A Privacy-Preserving Benchmarking Platform

zur Erlangung des akademischen Grades eines

Doktors der Ingenieurwissenschaften

der Fakultät für Informatik
des Karlsruher Instituts für Technologie

genehmigte

Dissertation

von

Florian Kerschbaum

aus Erlangen

Tag der mündlichen Prüfung: 16. Juni 2010
Erster Gutachter: Prof. Dr. Jörn Müller-Quade
Zweiter Gutachter: Prof. Dr. Günter Müller

Contents

List of Figures

List of Tables

Abstract

A privacy-preserving benchmarking platform is practically feasible, i.e. its performance is tolerable to the user on current (year 2007) hardware while fulfilling a compromise of functional and security requirements.

Benchmarking is the process of comparing one's own key performance indicators (KPI) to statistics of one's peer group. KPIs are quantitative measurements of business process performance, e.g. cash flow, machine lead time, or employee fluctuation rate. A peer group is a group of similar companies, usually competitors, wanting to compare against each other.

A benchmarking platform is a software service facilitating the benchmarking process. It is offered by a service provider who provides to its subscribers the statistics of their peer groups.

Such a platform can work based on the community concept. Everybody who is retrieving statistics is donating his KPIs as input to their computation, but most companies would prefer to keep their KPIs confidential and private. A privacy-preserving benchmarking platform can compute the statistics of the KPIs without revealing the KPIs to anyone.

This dissertation designs, architects, and evaluates an implementation of such a privacy-preserving benchmarking platform. First requirements are gathered and a general outline of the solution space is performed. Then a privacy-preserving benchmarking protocol that meets all requirements is designed. This is embedded into an architecture addressing the needs of a software system. Finally an implementation of the system is evaluated under real-world requirements and conditions.

In particular, this dissertation contributes:

- a novel (secure computation) benchmarking protocol in the central communication model
- a novel method for computing peer groups, such that all companies are assigned and the minimum size requirement of each peer group is met
- a realistic evaluation of the first ever benchmarking platform

The evaluation shows that the implemented privacy-preserving benchmarking platform is practically feasible.

Zusammenfassung

Ein vertraulichkeits-erhaltender Benchmarking Dienst ist praktisch umsetzbar, d.h. seine Ausführungsgeschwindigkeit ist für den Benutzer auf gegenwärtiger Hardware (Jahr 2007) akzeptabel und die funktionalen und Sicherheitsanforderungen können abgewogen erfüllt werden.

Benchmarking ist der Vergleich eigener Kennzahlen (KPI) mit Statistiken der Vergleichsgruppe. KPIs sind quantitative Masszahlen für die Leistung eines Geschäftsprozesses, z.B. Liquidität, Maschinenzyklus oder Angestelltenwechselrate. Eine Vergleichsgruppe ist eine Gruppe ähnlicher Unternehmen, häufig Konkurrenten, die sich gegenseitig vergleichen möchten.

Ein Benchmarking Dienst ist ein Dienstplattform, die den Benchmarkingprozess ermöglicht. Er wird von einem Dienstleister betrieben, der seinen Kunden die Statistiken ihrer Vergleichsgruppe zur Verfügung stellt.

Ein solcher Dienst kann nach dem Gegenseitigkeitsprinzip funktionieren. Jeder, der Statistiken abruft, stellt seine KPIs als Eingabe zur Berechnung zur Verfügung. Allerdings würden die meisten Unternehmen ihre KPIs vorzugsweise geheim halten. Ein vertraulichkeits-erhaltender Benchmarking Dienst berechnet die Statistiken ohne die KPIs irgendjemand offenzulegen.

In dieser Dissertation wird ein solcher vertraulichkeits-erhaltender Benchmarking Dienst entworfen, entwickelt und evaluiert. Zuerst werden die Anforderungen gesammelt und eine Übersicht über den Lösungsraumes präsentiert. Dann wird das entsprechende vertraulichkeits-erhaltende Benchmarking Protokoll entwickelt. Dies wird in eine umfassende Software Architektur eingebettet. Zum Abschluss wird eine Implementierung unter realistischen Anforderungen und Bedingungen evaluiert.

Die Beiträge dieser Dissertation sind insbesondere

- ein neues Benchmarking Protokoll zur sicheren Mehrparteienberechnung in einem zentralisierten Kommunikationsmodell.
- eine neue Methode zur Berechnung von Vergleichsgruppen, so dass alle Unternehmen einer Gruppe zugewiesen werden und jede Gruppe eine Mindestgröße erreicht.
- eine realistische Evaluierung des ersten Benchmarking Dienstes

Die Evaluierung zeigt, dass der implementierte vertraulichkeits-erhaltende Benchmarking Dienst praktisch umsetzbar ist.

Acknowledgements

First and foremost I would like to thank my advisor Prof. Jörn Müller-Quade for guiding the final phases of this dissertation. I learnt a great deal from him including to a very positive attitude towards even notoriously difficult problems. His quick understanding and trust into my abilities finally made the dissertation possible. A great thank you also to my second advisor, Prof. Günter Müller, who provided helpful guidance on political issues. His continuous support helped me withstand some difficult discussions.

I would also like to thank Dr. Orestis Terzidis, my initial supervisor at SAP, for suggesting the topic, granting the necessary freedom to complete it on time and supporting my professional development. Additional thanks go to Dr. Elmar Dorner, my second supervisor at SAP, for careful direction advising on the completion of this dissertation. Many thanks also go to Dr. Zoltan Nochta, my final supervisor at SAP, for picking up the topic and disseminating it throughout SAP as part of his job.

The SAP A1S Solution Management "Outside-In" team deserves a special mention for their close collaboration on the topic, help in defining the requirements and general insights into the topic of benchmarking and how to market it. I benefited a great deal from the discussions with them.

Another great source of support were the SAP Research CEC Karlsruhe PhD students who formed a lively group that was enjoyable to hang around.

My warmest wishes go to my family, and especially my aunt Hanne Kerschbaum and my mother Christa Kerschbaum. Without their support in difficult times this dissertation would have never been written.

This dissertation is dedicated to my parents Christa Kerschbaum and Peter Kerschbaum.

1 Introduction

Since the Internet economy has been establishing itself, more and more forms of collaboration over the Internet are emerging. Service providers are offering novel kinds of services and interest groups are gathering around them.

This dissertation describes the attempt at building a system for such a new service. It defines a new form of collaboration, designs a system, implements and evaluates it. The dissertation spans a wide point of view on the topic from purely theoretical proofs of security over design aspects in the system architecture to practical distributed system evaluation.

The topic of investigation and the purpose of collaboration is benchmarking. Benchmarking is the comparison of one company's key performance indicators (KPI) to the statistics of the same KPIs of its peer group. A key performance indicator (KPI) is a statistical quantity measuring the performance of a business process. Examples from different company operations are make cycle time (manufacturing), cash flow (financial) and employee fluctuation rate (human resources). A peer group is a group of (usually competing) companies that are interested in comparing their KPIs based on some similarity of the companies. Examples formed along different characteristics include car manufacturers (industry sector), Fortune 500 companies in the United States (revenue and location), or airline vs. railway vs. haulage (sales market).

Benchmarking is an important practice in managing businesses. Often it is performed by consulting companies which have access to KPIs of several companies; however new forms of benchmarking are emerging, e.g. databases for sale with KPI data. The benchmarking platform developed in this dissertation allows a company to buy a benchmarking service. The service provider is selling the statistics of the KPIs and in exchange the companies are supplying their KPIs to the statistics computation.

The benchmarking platform is a system run by the service provider that provides the statistics of the KPIs and runs a protocol for their computation from the KPIs with its customers. It is a server that acts as single focus point of the service, usually placed near the center of the network. From a customer's perspective the service provider is the only entity to interact with. Therefore the service provider can serve as the seller of the service,

although it is actually a collaboration of all customers.

The benchmarking service works at its full potential if a large number of companies is actively using it. The more companies supply their KPIs, the more accurate the statistics get, and the more useful the service is to its customers.

Another differentiating feature of the benchmarking platform is that it is designed to serve a variety of customers from very different industrial sectors. It is not targeting a uniform set of customers that forms one single peer group, but a diverse variety of them. This strengthens the observation that the more customers join the platform, the more useful is its service to its customers.

Privacy is of the utmost importance in benchmarking. Companies are reluctant to share their business performance data due to the risk of losing a competitive advantage or being embarrassed. This implies a data sharing risk and obstacle that opposes the successful establishment of the benchmarking platform. An important aspect of the benchmarking platform is therefore that it is privacy-preserving. Privacy-preserving means that the KPIs of the individual company are kept confidential to that company, i.e. the company does not suffer any data sharing risk (risk from revelation of the data), and the associated obstacles are being removed, since the KPIs are never revealed to anybody.

1.1 The Term "Privacy"

Before elaborating on the advantages of privacy, a short discussion on the use of the term is necessary. Privacy is a term from the Anglo-Saxon vocabulary. It refers to the protection of personal space and guarantees a freedom from governmental powers within that space. In particular in the American culture, privacy is seen as a highly-ranked value.

In contrast in the German culture, privacy is ranked much lower and many governmental rights may invade privacy. The corresponding German term is "Privatssphäre" and is often connected to unobservability and secrecy.

The term privacy has found extensive use in the electronic society. It has been used in many different ways and most often refers to the handling of personal data. Personal data is information that has been gathered about a person. Protection of such data is necessary to protect privacy as in freedom from governmental powers in an electronic society. So far, Anglo-Saxon and American law has established regulations first for data privacy protection, but these are traditionally less strict.

In contrast in German law a strong data protection act was established. It regulates "informationelle Selbstbestimmung": the right to determine how one's personal data is handled.

There are many technical ways to protect personal data, and these techniques have often been termed "privacy-enhancing" technologies. They usually enhance an established practice with privacy-protecting features, that improve the gathering and handling of personal data. The simplest and technically strongest form of data privacy protection is when no personal data is revealed throughout the process, since then no technical process for self-determination is required. These technologies have often been termed "privacy-preserving", since they do not allow breaches of privacy using personal data in the first place, i.e. a privacy-preserving technique is a technique that completes the process without revealing personal data.

This leads to another use of the term privacy (as in data privacy). Privacy refers to the confidentiality of (personal) data against all other persons and entities. It is therefore a stronger form of confidentiality, as it includes the definition of who is kept from that information. In contrast, a confidential communication (e.g. a secure channel) involves two parties to whom the data is revealed.

In the context of benchmarking multiple companies collaborate. Companies are not natural persons and therefore have no personal data and no right to privacy in either the Anglo-Saxon and American law or in the German law. However, just as natural persons, companies have data that emerge within their operational boundaries. Such data also enjoys legal protection in the form of "business secrets" with the intention of regulating competition and excluding criminal practices.

KPIs are definitely data that emerge from within the company. They are unknown to outsiders when measured and deserve protection by the company. In line with the forms of data privacy protection, the strongest form of protection does not reveal them. A privacy-preserving technique therefore protects the confidentiality of this data against all outsiders.

In conclusion, the term "privacy" and "privacy-preserving" in this dissertation are used for the confidentiality (and protection thereof) of secret business data.

1.2 Hypotheses and Research Methodology

The main thesis of this dissertation is that

- a privacy-preserving benchmarking platform is practically feasible, i.e. its performance is tolerable to the user on current (year 2007) hardware while fulfilling a compromise of functional and security requirements.

This involves two distinct aspects of research.

First, the design and analysis of a secure multi-party computation protocol for privacy-preserving benchmarking is performed. The underlying hypothesis is that

- current secure computation protocols do not address the distributed systems and network requirements of practical systems.

This is for example evidenced by the fact that no protocol for secure computation in the necessary central communication pattern has been published. Other examples include supply chain networks or publish-subscribe networks. Therefore this dissertation first defines the requirements for a practical system and compares existing theoretical approaches to it. Then it proceeds to define a benchmarking protocol meeting all requirements. This benchmarking protocol improves on the complexity of state-of-the-art protocols by using a common key among the subscribers of the benchmarking platform. The common key can be replaced by using more complex, less fit-for-purpose protocols that provide higher theoretical security guarantees. The research methodology is theoretical analysis and a security proof based on existing approaches is given. Current security analysis approaches are extended by matching them to the economic motivation. This yields a better expected performance of the protocols than purely theoretical approaches that neglect economics. Chapter 4 reports the results of this development. Chapter 8 describes related protocols that illustrate the trade-offs in designing protocols for benchmarking.

Second, the protocols have been placed in the greater context of a benchmarking system. The underlying hypothesis is that

- there is little to no experience in designing and implementing systems that use secure computation.

This is evidenced by the few publications for implementations of secure computation and the complete lack of commercial systems. A system architecture that accommodates the secure computation protocol is therefore given. The consequence of using secure computation are shown and special solutions are developed. The research methodology is software engineering and information systems research and a careful design analysis of the problem is given. Consideration is given to the user experience and other socio-economic factors. Result is a practically useful system that preserves the privacy of the subscribers. Chapter 3 outlines the solution space. Chapter 6 summarizes the results obtained by this work.

Third, the developed protocol and system is implemented and evaluated under real-world conditions. The underlying hypothesis is that

- current secure computation protocols are insufficiently evaluated for practical performance.

This is evidenced by the few publications for evaluations of secure computation. As part of the work for this dissertation, the protocols have been implemented and thoroughly evaluated as a system. This dissertation breaks

new ground by developing a protocol and method for high-performance, yet randomized privacy-preserving comparison. The research methodology is computer systems research by practical experimentation. Not only its performance, but also its security has been evaluated using practical experimentation which is a new concept for privacy-preserving security evaluation. The entire system has been implemented, evaluated and practically used. Very few evalutations have been done to estimate the performance under real-world conditions. This is one of the first secure computation systems evaluated for practical evaluation. Chapter 5 describes the tradeoff between security and performance in comparison. Chapter 7 shows the results in greater detail.

1.3 Structure of the Dissertation

This dissertation is divided into nine chapters. The chapters usually build on top of each other, such that sequential reading is best. The reader knowledgeable in the subject can, of course, skip Chapter 2. Also the main chapters (Chapter 4 and Chapter 6) approach the problem with two different research methodologies. They can therefore be best understood by experts in the different areas of computer science theory and systems research and can be read independently. Chapter 5 bridges the orthogonal goals of security (Chapter 4) and practical performance (Chapter 7) by developing an important protocol step that trades performance for security. Nevertheless both aspect, security and performance, are experimentally evaluated. It is anticipated that this could open a new research field, since the current alternative to secure multi-party computation is no protection at all. The following list gives a short abstract of the subject of each chapter.

Chapter 1 is this introduction. It introduces the motivation behind the problem and explains the general setting of the problem. It outlines the structure of the dissertation and lists all contributions including the practical realization of the work.

Chapter 2 serves two purposes: It reviews existing work, compares it to the solution developed in this dissertation and describes the building blocks upon which this solution builds. It describes other benchmarking systems that have been built, none of which is privacy-preserving. It lists secure computation protocols that can solve the problem of privacy-preserving computation. A distinction is made between general protocols that can solve any problem and specific protocols for statistics. No protocol satisfies all the requirements yet. The cryptographic building blocks used in the construction of the protocol and the concepts of web services used in the implementation conclude the chapter.

Chapter 3 presents the requirements for the benchmarking platform in detail. It also contains a first analysis of the solution space and presents

the first practical limitation on the protocol. Its main conclusion is that a subscriber can be in one peer group at most. This has an impact on the protocol design and the system architecture which then requires a special peer group formation algorithm.

Chapter 4 describes and analyzes the benchmarking protocol. It starts by describing the setup and key distribution. A practical procedure to reduce the key distribution to a public key infrastructure is described. It then describes the protocol in great detail. A textual and a formal description are given. Then the protocol is proved secure in the semi-honest and a newly developed constrained malicious model. The constrained malicious model allows arbitrary protocol deviations, as long as the correct result is delivered to the subscribers. It fits to the economic motivation of the benchmarking platform.

Chapter 5 introduces a randomized protocol for comparison. It first describes the underlying hiding technique in a two-party example. A performance measurements underpins its superior performance. Then the technique is integrated into the benchmarking protocol. In a series of experiments the leakage of the randomization is evaluated. These experiments cover the application in the benchmarking protocol. The conclusion is that the protocol provides practically acceptable leakage at superior performance.

Chapter 6 describes the design and architecture of the entire system. It starts with a use case analysis for the benchmarking platform. The key insight is to separate the protocol for statistics computation from the statistics retrieval. This allows the subscriber to retrieve statistics as soon as he becomes registered. Then it describes the novel peer group formation algorithm. The peer group formation algorithm builds peer groups, such that no subscriber is left out and all peer groups satisfy the minimum size requirement for privacy. The chapter concludes with a comparison of the notification versus the polling model of implementation. The conclusion is that a protocol can be completed within one polling interval in the polling model.

Chapter 7 presents the results of the experiments done with the implementation. Three main experiments have been conducted. The first measured the overall performance under different network conditions, the second measured the network traffic in bytes sent and the third the local performance with different key lengths. The conclusion is that the performance of the benchmarking protocol is sufficient to support practical realization.

Chapter 8 describes three problems which have been solved along the way. Constant-cost benchmarking is the same problem as the main thesis, but under stronger network performance constraints. This is the first solution for this important problem. The proposed benchmarking platform uses a common, shared key. The security implications of this choice have been minimized by two additional protocols presented in this chapter. The first replaces the static common key with a common session key at the expense

of anonymity. The second, coalition-safe protocol, replaces the common key entirely with a threshold variant of the encryption scheme at the expense of higher complexity.

Chapter 9 combines the conclusions from the different chapters and summarizes them to support the main thesis of this dissertation: A privacy-preserving benchmarking platform is practically feasible. It also discusses possible future work.

1.4 Contributions

This dissertation contributes an extensive treatment of privacy for benchmarking systems. In particular it contributes

- a privacy analysis of collaborative benchmarking in the service provider model
- novel benchmarking protocols in the service provider model
- a new security model matching the economic motivation of the service provider
- an efficient, randomized comparison technique with verified privacy
- design and system architecture for a benchmarking platform using privacy-preserving protocols
- novel peer group formation algorithm that addresses privacy needs
- evaluation of the system under real-world non-functional requirements

In summary, this dissertation is the first attempt at building a privacy-preserving application end-to-end. It does not stop at a theoretic treatment of possible protocols, but investigates the intricacies of building such a system as a distributed system. The author therefore believes that this dissertation can provide great insight into the problems of building such applications even beyond benchmarking.

1.4.1 Publications

Many insights gained have been published throughout the process of writing this dissertation. In particular, directly related results are

1. Florian Kerschbaum, Daniel Dahlmeier, Axel Schröpfer, and Debmalya Biswas.
   On the Practical Importance of Communication Complexity for Secure Multi-Party Computation Protocols.
   Proceedings of the 24th ACM Symposium on Applied Computing, 2009.

2. Florian Kerschbaum.
   Building A Privacy-Preserving Benchmarking Enterprise System.
   Enterprise Information Systems 2 (4), 2008.

3. Florian Kerschbaum.
   Practical Privacy-Preserving Benchmarking.
   Proceedings of the 23rd IFIP International Information Security Conference, 2008.

4. Florian Kerschbaum.
   Building A Privacy-Preserving Benchmarking Enterprise System.
   Proceedings of the 11th IEEE International EDOC Conference, 2007.

5. Florian Kerschbaum, and Orestis Terzidis.
   Filtering for Private Collaborative Benchmarking.
   Proceedings of the International Conference on Emerging Trends in Information and Communication Security, 2006.

The paper 4 has been awarded the Best Paper Award. Indirectly many papers have benefited from the results gained. These include:

1. Florian Kerschbaum.
   A Verifiable, Centralized, Coercion-Free Reputation System.
   Proceedings of the ACM Workshop on Privacy in the Electronic Society, 2009.

2. Florian Kerschbaum.
   Adapting Privacy-Preserving Computation to the Service Provider Model.
   Proceedings of the 1st IEEE International Conference on Privacy, Security, Risk and Trust, 2009.

3. Rafael Deitos, and Florian Kerschbaum.
   Improving Practical Performance on Secure and Private Collaborative Linear Programming.
   Proceedings of the 1st International Workshop on Business Processes Security, 2009.

4. Florian Kerschbaum, Debmalya Biswas, and Sebastiaan de Hoogh.
   Performance Comparison of Secure Comparison Protocols.
   Proceedings of the 1st International Workshop on Business Processes Security, 2009.

5. Florian Kerschbaum, Andreas Schaad, and Debmalya Biswas.
   Practical Privacy-Preserving Protocols for Criminal Investigations.
   Proceedings of the 7th IEEE International Conference on Intelligence and Security Informatics, 2009.

6. Axel Schröpfer, Florian Kerschbaum, Dagmar Sadkowiak, and Richard Pibernik.
Risk-Aware Secure Supply Chain Master Planning.
Proceedings of the 7th International Workshop on Security in Information Systems, 2009.

7. Florian Kerschbaum, and Alessandro Sorniotti.
RFID-Based Supply Chain Partner Authentication and Key Agreement.
Proceedings of the 2nd ACM Conference on Wireless Network Security, 2009.

8. Rafael Deitos, and Florian Kerschbaum.
Parallelizing Secure Linear Programming.
Concurrency and Computation: Practice and Experience 21 (10), 2009.

9. Florian Kerschbaum, and Philip Robinson.
Security Architecture for Virtual Organizations of Business Web Services.
Journal of Systems Architecture 55 (4), 2009.

10. Florian Kerschbaum, and Julien Vayssiere.
Privacy-Preserving Data Analytics as an Outsourced Service.
Proceedings of the ACM Workshop on Secure Web Services, 2008.

11. Florian Kerschbaum, and Andreas Schaad.
Privacy-Preserving Social Network Analysis for Criminal Investigations.
Proceedings of the ACM Workshop on Privacy in the Electronic Society, 2008.

12. Leonardo Weiss Ferreira Chaves, and Florian Kerschbaum.
Industrial Privacy in RFID-based Batch Recalls.
Proceedings of the IEEE International Workshop on Security and Privacy in Enterprise Computing, 2008.

13. Octavian Catrina, and Florian Kerschbaum.
Fostering the Uptake of Secure Multiparty Computation in E-Commerce.
Proceedings of the International Workshop on Frontiers in Availability, Reliability and Security, 2008.

14. Florian Kerschbaum, and Julien Vayssiere.
Privacy-Preserving Logical Vector Clocks using Secure Computation Techniques.
Proceedings of the 13th IEEE International Conference on Parallel and Distributed Systems, 2007.

15. Florian Kerschbaum.
Distance-Preserving Pseudonymization for Time-stamps and Spatial Data.
Proceedings of the ACM Workshop on Privacy in the Electronic Society, 2007.

16. Florian Kerschbaum.
Simple Cross-Site Attack Prevention.
Proceedings of the 3rd IEEE International Conference on Security and Privacy in Communication Networks, 2007.

17. Florian Kerschbaum, Rafael Deitos, and Philip Robinson.
Securing VO Management.
Proceedings of the 4th International Conference on Trust, Privacy & Security in Digital Business, 2007.

18. Florian Kerschbaum.
A new way to think about Secure Computation: Language-Based Secure Computation.
Proceedings of the 5th International Workshop on Security in Information Systems, 2007.

19. Rafael Deitos, Florian Kerschbaum, Philip Robinson, and Jochen Haller.
A Comprehensive Security Architecture for Dynamic, Web Service Based Virtual Organizations for Businesses.
Proceedings of the ACM Workshop on Secure Web Services (Poster), 2006.

20. Yücel Karabulut, Florian Kerschbaum, Fabio Massacci, Philip Robinson, and Artsiom Yautsiukhin.
Security and Trust in IT Business Outsourcing: a Manifesto.
Proceedings of the 2nd International Workshop on Security and Trust Management, 2006.

21. Philip Robinson, Florian Kerschbaum, and Andreas Schaad.
From Business Process Choreography to Authorization Policies.
Proceedings of the 20th Annual IFIP WG 11.3 Working Conference on Data and Applications Security, 2006.

22. Florian Kerschbaum, Jochen Haller, Yücel Karabulut, and Philip Robinson.
PathTrust: A Trust-Based Reputation Service for Virtual Organization Formation.
Proceedings of the 4th International Conference on Trust Management, 2006.

23. Florian Kerschbaum.
    Practical Private Regular Expression Matching.
    Proceedings of the IFIP WG 11.4 I-NetSec Workshop, 2006.

An invited talk summarizing some of the results has been given.

1. Florian Kerschbaum, and Rafael Deitos.
   Security Against the Business Partner.
   Proceedings of the ACM Workshop on Secure Web Services, 2008.

1.4.2 Practical Realization

This dissertation and in particular its implementation part has been developed while the author was employed at SAP Research. These research results have been transferred to the solution management (product definition) department within SAP. In fact, the development and in particular the requirements gathering were done in close collaboration. The assumptions made about real-world requirements within this dissertation are therefore based on real market research and product requirements.

Furthermore, the practicality of the approach and its economic impact has been validated by a real product definition group. Again assumptions made about the economics of security and the service provider model are therefore based on real market requirements.

The capabilities in privacy protection thanks to the protocol definition and the usability of the client software thanks to the architecture have convinced the solution management group to pursue this approach further. It is anticipated that a second generation version of SAP's benchmarking product will be privacy-preserving and be based on the work of this dissertation. While writing this dissertation, a customer pilot is planned with the prototype software evaluated in this dissertation. Therefore this work clearly has a strong impact on future computer systems and the economy.

1.5 Summary

This chapter introduced the necessity for a privacy-preserving benchmarking platform. It briefly explained the usage of "privacy" in this dissertation and presented the thesis to be supported by this dissertation: *A privacy-preserving benchmarking platform is practically feasible.*

Following this, the chapters of this dissertation are briefly outlined. Chapter 4 and Chapter 6 contain the main contribution of the dissertation. Chapter 4 is a theoretical development of the protocol and Chapter 5 extends it with an efficient, randomized technique for comparison. Chapter 6 builds the architecture following an information systems approach. The next section summarized the contributions including a list of publication of

this dissertation besides acting as a proof the thesis statement. Furthermore the efforts to turn the benchmarking platform into a product have shortly been mentioned. The process is quite advanced and it is anticipated that future (year 2007) versions of SAP's benchmarking product will be based on this work.

2 Related Work

2.1 Internet-Based Benchmarking

Literature on benchmarking in the business community is vastly available. It views benchmarking as a decision making process and models its effectiveness by the improvements of the decisions' effects. In general one can distinguish two forms of benchmarking:

- individual benchmarking where the focus is on the detailed analysis of single company and its improvement possibilities.
- overall benchmarking where the whole population of companies is analyzed for a common improvement perspective.

The benchmarking platform of this dissertation focuses on individual benchmarking giving a decision maker in one company the tools to improve his performance.

The advantage of an Internet-based benchmarking platform has been recognized by other researchers [9]. Bogetoft and Nielsen describe two benchmarking methods for such platforms. Neither of them is privacy-preserving, but both have been implemented. They mention the protection of information and anonymity as future work for their platforms [9].

The problem of peer group formation has not been addressed in their platform. Instead the authors assume that only one homogeneous peer group, e.g. banks, is using the platform. This is opposed to the privacy-preserving benchmarking platform which is supposed to cater for a very large and diverse population of companies.

A particular case study of an Internet-based benchmarking system has been done for the hospitality industry [18]. They implemented and evaluated their platform in a practical usage scenario. They reported problems with technology use as their main obstacle to future adoption, e.g. they mention the reluctance of personnel to enter data in regular intervals as a major problem.

They do not report on privacy problems or developments in their application. Peer group formation is not an issue for them, since they also have a very homogeneous user population.

In summary, one can conclude that currently (year 2007) Internet-based benchmarking applications are emerging. Two out of two existing implementations use a central platform or service as their modus of implementation. One implementation mentions privacy as an important aspect for future work.

2.1.1 Peer Group Formation

As mentioned in the previous chapter there is no previous work that particularly considers the problem of peer group formation. The two existing benchmarking applications both assume a homogeneous peer group with respect to the characterstics of the companies in the group. As a result they form one peer group in the entire application. The privacy-preserving benchmarking platform of this dissertation in contrast is supposed to cater for a very large and diverse user population.

Then the problem of peer group formation arises. The companies are mapped to multi-dimensional discrete data points along certain characteristics. The goal is to identify groups of companies with similar characteristics which are suitable for benchmarking. Section 6.2.1 gives the details about possible characteristics and an elaborated formulation of the problem.

This peer-group formation problem is seen as a data clustering problem where similar companies (users) need to be grouped together. Data clustering is defined as the assignment of objects into groups (called clusters) so that objects from the same cluster are more similar to each other than objects from different clusters. Usually similarity is assessed according to a distance measure. Data clustering algorithms can be classified into hierarchical and partitional methods. For more information about data clustering see the survey of [44].

For peer group formation, the requirements for the data clustering algorithm are that the clusters are partitional and the computational effort must be low. The k-means clustering algorithm is the fastest partitional data clustering algorithm [44] and also quite popular. The next section will review the k-means data clustering algorithm which is later modified to be used for peer group formation.

K-Means Clustering

K-means clustering was introduced in [54]. The algorithm of k-means clustering is iterative and always features grouping into k clusters.

The algorithm follows the following steps:

1. k distinct cluster centers are chosen randomly.

2. Each data point is assigned to its closest cluster center.

3. Each cluster center is recomputed as the center of its assigned data points.

4. If the change in location of the cluster centers is above a threshold δ, then iterate from step 2.

5. Output the clusters.

Several parameters influence the performance of the clustering and need to be chosen by the user of the algorithm. The number of clusters k influences running time and resulting cluster size and the convergence threshold δ influences running time and quality.

The initially chosen cluster centers also influence the resulting clusters and not each starting set leads to the same result, i.e. not each run of k-means clustering leads to the same result. K-means clustering reduces the intra-cluster variance, but does not yield a global minimum in variance. Its advantage is speed, such that it can be run over very large data sets, such as a subscriber database.

A variation of k-means clustering is constrained k-means clustering [6]. In constrained k-means clustering the data points are assigned, such that no cluster is below a certain size. This ensures that in the resulting clustering all clusters also have a minimum size.

They view the cluster assignment step (step 2 in the above algorithm) as a linear programming problem. The assignment is a $0, 1$ matrix of variables where 1 indicates that this data point is assigned to this cluster. The optimization objective of the linear programming problem is to minimize the element sum of the matrix product of the assignment matrix times the distance to the cluster centers matrix. The constraints on the assignment variables are that each data point can be assigned at most once and that no assignment variable can be negative.

The constraint additionally introduced in constrained k-means clustering is that each cluster must be of a minimum size. This is just another constraint on the assignment matrix and can be introduced into the linear programming problem. Then the cluster assignment step is a solution to the linear programming problem.

The linear programming problem has size kn variables (the assignment variables) where k is the number of clusters and n the number of data points. It has $kn + k + n$ constraints: kn constraints that no assignment variable is negative, k constraints for the minimum cluster size, and n constraints that no data point is assigned more than once.

In summary, data clustering is an established technique and k-means clustering works effectively for large data sets. The problem of adding a minimum cluster size has been considered before in constrained k-means clustering, but the linear programming problem that must be solved can grow very large for large data sets.

2.2 Database Privacy

The privacy-preserving protocol competes with the protection of data at the database level. The most prominent approach of privacy in databases is k-anonymity.

2.2.1 K-Anonymity

K-anonymity [70, 77] tries to prevent privacy-sensitive inferences between database tables. Let there be a table of the population under investigation, e.g. in our case companies subscribed to the benchmarking platform. This table has a set of quasi-identifiers, i.e. attributes whose combination uniquely identifies an entry. In the case of benchmarking this could be criteria to identify a company. Then in a k-anonymous table each combination of these quasi-identifiers must appear at least k times. In the table of all KPIs gathered by the service provider this would correspond to the peer groups, i.e. each combination of criteria would form a peer group. The parameter k then corresponds to the minimum peer groups size.

In the described form k-anonymity could protect benchmarking data, but updates as necessary for a growing benchmarking platform present an unsolved challenge. Therefore the secure computation approach is better suited for continuous protection of the data compared to a one-time release as assumed in the k-anonymity model.

2.2.2 L-Diversity

Two attacks on k-anonymity have been identified in [53]. First, k-anonymous groups can leak information due to the lack of diversity in the sensitive attribute, e.g. if all attributes have the same value. Second, background knowledge can be used to infer sensitive information even if the data is k-anonymous. In order to prevent that k-anonymous groups need to be chosen, such that the sensitive data items are l-diverse. Loosely speaking l-diverse means that the diversity of the sensitive attributes is sufficiently close to the distribution of the real data.

A micro-data release is the release of individual, potentially anonymized tuples of a database. The diversity requirement of l-diversity would apply to a micro-data release of benchmarking data, but since the benchmarking platform is releasing only aggregate statistics, it does not apply to it. This is fortunate, since a diversity requirement for peer groups could negatively impact its usefulness for benchmarking.

2.3 Secure Computation

This section describes protocols and implementation of secure computation. Secure (multi-party) computation is a cryptographic technique that can compute any function of distributed inputs with finite input and output length, such that nothing is revealed except the result. Or more precisely, let $f(\vec{x})$ be an n-ary function (i.e. $|\vec{x}| = n$) with up to n outputs, and denote the i-th output at party X_i as $f_i(\vec{x})$. Then, after the secure computation protocol execution party X_i knows its input x_i, and the result $f_i(\vec{x})$ (and everything that can be inferred from it), but nothing else.

This section presents three aspects of secure computation. First, general protocols are introduced, second, some special protocols for statistics and third, implementations of secure computation protocols are described. General protocols can realize any functionality $f(\vec{x})$, while special protocols optimize the protocols for special functions. The idea is that domain-specific knowledge can be used to create better functions for important problems. This has been suggested in [38].

Recently (year 2007) several implementations of secure computation protocols and applications have emerged. Section 2.3.3 reviews them. The privacy-preserving benchmarking platform is an implementation of a special protocol for statistics computation.

2.3.1 General Protocols

Secure computation has been introduced and solved for any function with finite input and output domain in the two-party case by Yao [80]. It introduced the idea of circuit construction to provide completeness for any functionality. The realized function $f(\vec{x})$ with fixed input length is represented as a circuit C and then this circuit is emulated in the protocol. A function $f(\vec{x})$ with variable input length is represented as a (non-uniform) circuit family, but in this section the focus is on one concrete, implementable member of that family for one fixed input-length function. Any function $f(\vec{x})$ (with finite fixed-length input and output domain) can be represented as a circuit with very few gate types, e.g. only "not-and" gates or only "and" and "exclusive-or" gates. Given protocols to emulate these few gate types and a composition theorem that shows that their combination into any circuit is still secure, the circuit can be emulated as a composition of several gate protocols. By this circuit construction technique the general protocol has been reduced to a (composable) gate protocol (or protocols) for a binary boolean function.

Yao's seminal paper [80] also introduced Yao's millionaires' protocol as an important problem and solved it. Imagine two millionaires that want to compare their riches, but do not want to reveal to each other the exact amount. Yao's millionaires' protocols implement the greater-than function-

ality in a privacy-preserving manner.

Yao's protocol starts by Alice scrambling the circuit. She chooses a random bit for each wire and "exclusive-or"s it with the bit sent over that wire into a scrambled bit. A key is chosen for each wire and each bit value, such that the scrambled bit value can be deduced from the key. Each gate entry is represented as a table of four entries of the key values for the resulting scrambled bit values. Then each key is encrypted with a combined key of the two input wires. The scrambled circuit is sent to Bob.

Oblivious Transfer is a cryptographic protocol in which the sender Alice has n messages $m_i (i = 1, \ldots, n)$ and the receiver Bob has a selection $b (1 \leq b \leq n)$. After the protocol Bob has received m_b and nothing about $m_{\neg b}$ and Alice has learnt nothing (about b) at all. An Oblivious Transfer with n messages is called a 1-out-of-n Oblivious Transfer. A detailed description of Oblivious Transfer is given in Section 2.4.2. Then for each of Bob's input wires, Alice and Bob engage in an Oblivious Transfer protocol, such that Bob only obtains the key corresponding to his input bit and Alice learns nothing about Bob's input.

Bob can then execute the circuit. Since he has only one key for each wire, he can only decrypt one entry in each gate and only obtain one result. This result can then be shared with Alice.

The first secure computation protocol in the multi-party setting was [37]. It provides security in the computational setting where adversaries are polynomially bound. It used a construction based on permutations of five elements and was later simplified by Goldreich [34]. In the simplified version all bits are shared using secret sharing. Circuits are represented only with addition ("exclusive-or") and multiplication ("and") gates. Exclusive-or gates can be executed locally due to the linearity of the secret sharing scheme. Multiplication gates are reduced to 1-out-of-4 Oblivious Transfer protocols between two parties for the four possible values of the two shares of the receiver.

Secure computation in the multi-party setting with information-theoretic security was introduced in [7]. In information-theoretic security the adversary is computationally unbounded. The protocol divides each bit into secret shares as well and addition gates are locally evaluated as well. Multiplication gates are evaluated by polynomial reduction on the number of variables. Due to the (t, n)-secret sharing $t > \frac{n}{2}$ honest parties suffice to execute the protocol properly.

Yao's protocol has been extended to the multi-party setting in [4]. This yields a constant round protocol, but still with communication complexity linear in the size of the circuit, since the entire circuit needs to be transferred. In [19] malicious model security is implemented, such that random number generators can be used as black boxes.

All presented secure multi-party computation protocols and many subsequent improvements rely on a complete mesh communication pattern, i.e.

each party communicates with each other party during the protocol. This communication model is ideal if no third party service provider is available, but in most business applications, e.g. benchmarking, such a platform is available.

Naor et al. [60] first introduced secure computation of any function in a server model. Clients only submit their inputs to the servers which then compute the function among them. In [60] two mutually distrustful, but honest servers are required. In [45] a fault in the zero-knowledge proofs was later fixed. The two servers execute Yao's protocol [80] between them after receiving the inputs via a special Oblivious Transfer protocol.

The server model was also picked up by other general protocols. The basic idea is to separate input, computation and output functionality. The clients split their input into shares or submit their encrypted input to the computation server which forward the result to the output server. This has been described for the protocols [15, 16] where [15] is an optimization of [7] which can operate with multiple servers and (t, n) secret sharing again. In this case $t > \frac{n}{2}$ servers must be honest.

In summary, secure computation is possible for any function in the two-party and in the multi-party setting. In the multi-party setting it can be secure even against computationally unbounded adversaries. Initial protocols relied on a full mesh communication pattern. While general server-assisted solutions exist, they require at least two mutually distrustful servers.

2.3.2 Special Protocols for Statistics

The problem of privacy-preserving benchmarking was first mentioned in [2]. They provide special protocols for advanced statistics, such as moving average, exponential smoothing, and linear regression. Their building blocks equal the building blocks of the benchmarking platform: homomorphic encryption, secret sharing and Oblivious Transfer. All protocols for the advanced statistics base on protocols for privately computing division. In their secure division protocols the players X_i each hold two inputs x_i and y_i. The output of the protocol is $\frac{\sum_i x_i}{\sum_i y_i}$. Four protocols are given for the multi-party and the two-party case. Their performance characteristics in relation to the number of participants n and the number of significant bits l in the solution are summarized in Table 2.1.

All secure division protocols are computed in a full mesh network architecture where each participant communicates with each other participant. Nevertheless in the most practical multi-party protocol DIV1 they achieve security also only without collusion with a special single party. None of the protocols has been implemented and been practically evaluated.

The problem of privacy-preserving statistics has been introduced in [24]. They also provide protocols for linear regression and correlation coefficient. Secure division also plays an important role in [24], but differently from [2]

Name	No. Parties	No. Rounds	Communication per party
DIV1	n	5	$O(n)$
DIV2	2	$O(\log l)$	$O(\log l)$
DIV3	2	2	$O(1)$
DIV4	n	5	$O(n^2)$

Table 2.1: Overview of secure division protocols by Atallah et al.

they also provide simplified protocols for the case where data is shared, such that Alice holds x and Bob holds y and they want to compute $\frac{x}{y}$. All given protocols are for the two-party case only, but multiple data items at each party are addressed.

The main building block for the protocols is a protocol for privately computing a two-party dot product of vectors. In this protocol Alice's vector $\vec{x}$ is divided into several vector shares $\vec{x}_i$, such that $\sum_i \vec{x}_i = \vec{x}$. Then each $\vec{x}_i$ is randomly permuted by Alice with a different permutation Φ_i. Bob's vector $\vec{y}$ is permuted by each Φ_i by Alice and each element of $\Phi_i(\vec{y})$ is blinded by adding a random variable, such that Bob does not get to know Φ_i. To hide $\vec{y}$ from Alice during this permutation it is encrypted using homomorphic, semantically secure encryption. Bob computes the dot product for each set of permutated vector and adds the results. The final result is the dot product blinded by a random number chosen by Alice.

The probability that Bob will learn an entry of Alice's vector is $\frac{1}{n^m}$ where n is the size of the vector and m is the number vector shares. Bob would need to guess the element of vector shares that sums to the element in $\vec{x}$. The probability of guessing one entry is $\frac{1}{n}$ and, since each of the m permutations is chosen independently, the resulting probability is $\frac{1}{n^m}$.

This protocol was applied to the problem of privately computing division as in [2]. Alice has x_1 and y_1 and Bob has x_2 and y_2 and they want to compute $\frac{x_1+x_2}{y_1+y_2}$. Bob chooses two random number r_1 and r_2 and sends $r = \frac{r_2}{r_1}$ to Alice. Alice and Bob use the scalar dot product protocol twice, but such that only Alice learns the result: First to compute $(x_1, 1) \cdot (r_1, r_1 x_2) = r_1(x_1 + x_2)$ and second to compute $(y_1, 1) \cdot (r_2, r_2 y_2) = r_2(y_1 + y_2)$. Alice computes the result as $r\frac{r_1(x_1+x_2)}{r_2(y_1+y_2)} = \frac{x_1+x_2}{y_1+y_2}$.

Later a flaw was found in the protocol by [51]. The problem is that r as sent by Bob is likely to reveal r_1 and r_2. A floating-point number reveals its quotient and divisor if they are co-prime which approximately happens with probability $\frac{6}{\pi^2}$ if they are uniformly chosen. [51] provides a very complex secure two-party protocol that is provably secure for approximating $\frac{x_1+x_2}{y_1+y_2}$. Their protocol requires a constant number of rounds and communication costs cubic in the size of the input (number of bits).

The first protocol to compute the median or more generally the k-th ranked element was given in [1]. They give protocols for the two-party

case and the multi-party case and versions secure against semi-honest and malicious attackers. A precise definition of a semi-honest attacker is given in Section 4.3.2 and precise defintion of a malicious attacker can be found in [34]. All protocols use general secure computation as a sub-protocol in order to compute certain steps of the protocol.

The two-party protocol to compute the median secure against semi-honest attackers proceeds as follows:

1. Let the median be the k-th ranked element. Both Alice and Bob prepare a set of their k smallest elements. Let $\mathbb{S}_A$ denote Alice's set and $\mathbb{S}_B$ Bob's. Without loss of generality assume that $|\mathbb{S}_A| = |\mathbb{S}_B| = 2^j$.

2. Alice and Bob engage in a general secure computation protocol to compute $m_A < m_B$ where m_A and m_B are the medians of their respective sets $\mathbb{S}_A$ and $\mathbb{S}_B$.

3. If $m_A < m_B$, then Alice removes all elements less than or equal to her median m_A from $\mathbb{S}_A$ and Bob removes all elements strictly greater than his median m_B from $\mathbb{S}_B$. If $m_A \geq m_B$, they do vice-versa.

4. Steps 2 and 3 are repeated until the sets are of size 1 and then the smaller element is chosen as the median.

This protocol requires $O(\log k)$ rounds and has $O(\log k)$ communication complexity. The advantage compared to general secure computation stems from the insight that the result of the private comparison can be public, i.e. known to Alice and Bob, since it can be deduced from the result. None of the protocols has not been implemented or practically evaluated. The protocol only guarantees privacy if all elements in the union of $\mathbb{S}_A \cup \mathbb{S}_B$ are unique. Furthermore no implementations for the protocols necessary for malicious security have been given in [1], such that one needs to use general constructions which can be very costly.

The multi-party protocol for computing the median guesses the median element by binary search in its domain. Let $[\alpha, \beta]$ be the domain of possible values in the sets $\mathbb{S}_i$, i.e. the values need to be integers or appropriately scaled.

1. Initialize $[a, b]$ to $\text{dom}(N) = [\alpha, \beta]$.

2. Set $m = \lceil \frac{a+b}{2} \rceil$ and let l_i be the number of values in set $\mathbb{S}_i$ strictly smaller than m and g_i be the number of values strictly greater than m.

3. The participants engage in a general secure computation that computes $l = \sum_i l_i$ and $g = \sum_i g_i$ and compares them to k. Again, let the median be the k-th ranked element. If $l \geq k$, then set $b = m - 1$. If $g \geq n - k + 1$, then set $a = m + 1$. Otherwise m is the median.

4. Repeat steps 2 and 3 until the median has been found.

The protocol requires $O(\log|\text{dom}(N)|)$ rounds and has communication complexity $O(\log|\text{dom}(N)|)$. Again the insight is that the result of the contained general secure computation can be public (in this case known to all parties), since it can be inferred from the result of the entire protocol. The communication pattern of the protocol depends on the communication pattern of the general secure computation protocol used. The advantage that this protocol provides is therefore limited to the selected disclosure of the intermediate protocol results.

A protocol for privately computing the maximum in a centralized communication model is presented in [21]. This protocol is based on number theoretic assumptions, namely quadratic residuosity. A quadratic residue x (modulo a constant m) is a number for which there exists a y, such that $y^2 \bmod m = x$.

The probabilistic (homomorphic) encryption scheme of [39] is based on quadratic residues. Let $Q(x|m)$ denote the quadratic residuosity of x modulo m, i.e. $Q(x|m) = 1$ if x is a quadratic residue and 0 if not. The Legendre symbol $(\frac{x}{p})$ for primes p implies $Q(x|p)$.

$$(\frac{x}{p}) = 1 \Longleftrightarrow Q(x|p) = 1$$

$$(\frac{x}{p}) = -1 \Longleftrightarrow Q(x|p) = 0$$

The Legendre symbol can be computed as $(\frac{x}{p}) = x^{\frac{p-1}{2}}$. For a more efficient algorithm to compute the Legendre symbol see [3]. Furthermore it holds that

$$(\frac{xy}{p}) = (\frac{x}{p})(\frac{y}{p})$$

The Jacobi symbol $(\frac{x}{n})$ is the extension of the Legendre symbol to composite numbers $n = p_1^{\alpha_1} p_2^{\alpha_2} \dots p_k^{\alpha_k}$. It is defined as

$$(\frac{x}{n}) = (\frac{x}{p_1})^{\alpha_1} (\frac{x}{p_2})^{\alpha_2} \dots (\frac{x}{p_k})^{\alpha_k}$$

An efficient algorithm for computing the Jacobi symbol without knowing the factorization of n can be found in [3]. The Jacobi symbol does not imply quadratic residuosity. There are so-called pseudo-primes for which there are quadratic non-residues with Jacobi symbol equal to 1, e.g. if $n = pq$, $(\frac{x}{p}) = -1$, and $(\frac{x}{q}) = -1$, then $(\frac{x}{n}) = 1$. There is no efficient algorithm known to compute quadratic residuosity for composite numbers. Computing the quadratic residuosity is believed to be as hard as factoring [39]. The encryption system of [39] encodes 0s and 1s as quadratic residues and quadratic non-residues (with Jacobi symbol 1), respectively.

The protocol of [21] uses the same bit-wise encoding for the numbers x_i to be compared for maximum. I.e. each party X_i holds a private key of a factorization of $n_i = p_i q_i$, while the composite n_i is a public-key. Each x_i will be represented as a set of vectors $\vec{v}_{i_j}$ of quadratic (non-) residues $v_{i_{j_k}}$ modulo n_j.

The protocol starts by defining a two-party subprotocol for Oblivious Transfer of a bit on the condition of quadratic residuosity. Alice has b_0, b_1 and y and Bob has n_j. After this special Oblivious Transfer Bob has b_0 if y is a quadratic non-residue $Q(y|n_j) = 0$ and b_1 if y is a quadratic residue $Q(y|n_j) = 1$. Details of the protocol can be found in [21].

Let m be the maximal size of any x_i in bits. Then the protocol defines a $m \times m$ matrix $\mathbf{M}_{i,j}$, such that $\mathbf{M}$ contains exactly one column with only quadratic non-residues if $x_i > x_j$. This matrix $\mathbf{M}_{i,j}$ is constructed as follows: Let $\vec{v}_{i_i}$ be the encoding of x_i modulo n_i and $\vec{v}_{j_i}$ be the encoding of x_j modulo n_i. The l-th column of $\mathbf{M}_{i,j}$ contains

1. $-v_{j_{i_k}} v_{i_{i_k}}$ for $k = 1, \ldots, l-1$.
2. $v_{i_{i_l}}$
3. $-v_{i_{j_l}}$ for $l+1, \ldots, m$

This l-th column contains only quadratic non-residues if

1. the first $l-1$ bits of x_i and x_j agree
2. the l-th bit of x_i is 1.
3. the l-th bit of x_j is 0.

The protocol in [21] also accounts for the case $x_i = x_j$ which has been omitted here.

Based on this matrix $\mathbf{M}_{i,j}$ a second special Oblivious Transfer is defined. Alice has b_0, b_1 and $\vec{v}_{i_i}$ and Bob has $\vec{v}_{i_j}$. They compute $\mathbf{M}_{i,j}$ and after this special Oblivious Transfer Bob has b_0 if $x_i > x_j$ and b_1 if $x_i < x_j$. Details of the protocol can again be found in [21].

The overall protocol then proceeds as follows. Let n be the number of clients X_i. In a registration phase the public key of each client X_i is distributed to each X_j. Then each X_i sends $\vec{v}_{i_j}$ for $j = 1, \ldots, n$ to the server. The server prepares a message mes, e.g. a signed "maximum", and splits it into n shares mes_i, such that $mes = mes_1 \oplus \ldots \oplus mes_n$ (where $\oplus$ denotes the exclusive-or operation). He prepares a random message mes' and mes'_i in the same way. He engages in n second special Oblivious Transfers with each X_i for mes_j, mes'_j and $\vec{v}_{i_i}$, $\vec{v}_{i_j}$ $(j = 1, \ldots, n)$. Only if X_i has the maximum she will receive all shares mes_j and be able to reconstruct mes which has been signed to distinguish it from random numbers. All others receive random messages.

After the protocol the holder of the maximum knows she has the maximum and can proof it by revealing the protocol transcript to the server. The protocol requires a constant number of rounds, but has a communication complexity per participant of $O(nm^2)$ (where m is the maximum size of the inputs in bits). Furthermore the protocol encodes the inputs x_i bit-wise which causes $O(m)$ modular exponentiations during the encryption.

The protocol is secure against collusion of up to $n-1$ clients (without the server), but only in the semi-honest model. Collusion with the server are excluded, since it can reveal an encoding $\vec{v}_{i_j}$ of party X_i's input x_i to party X_j. Security is only proven in the semi-honest model. Furthermore the protocol reveals the public keys of each party, such that parties are identifiable via them as pseudonyms.

A protocol for privately computing the sum in a centralized communication model has been presented in [22]. This protocol uses El-Gamal encryption [28] in its construction, but the main summation protocol can be presented without reference to a specific encryption system.

The setup consists of several parties X_i with input x_i and a central server SP. Each party has a public-, private-key pair $E_i(\cdot), D_i(\cdot)$. The public keys are known to all parties including the central server SP.

Let n be the number of parties. The summation protocol proceeds as follows:

1. Each party X_i splits her input x_i into n shares $x_{i,1}, \ldots, x_{i,n}$, such that $x_i = \sum_{j=1}^{n} x_{i,j}$. She computes $E_j(x_{i,j})$ for $j = 1, \ldots, i-1, i+1, \ldots, n$ and sends them all to the server SP.

2. The server SP sorts the messages and sends $E_j(x_{1,j}), \ldots, E_j(x_{j-1,j})$, $E_j(x_{j+1,j}), \ldots, E_j(x_{n,j})$ to subscriber X_j.

3. Each party X_i decrypts the received messages and computes $s_i = \sum_{j=1}^{n} x_{j,i}$ and send s_i to the server SP.

4. The server SP computes the sum

$$s = \sum_{i=1}^{n} s_i = \sum_{i=1}^{n} \sum_{j=1}^{n} x_{j,i} = \sum_{j=1}^{n} \sum_{i=1}^{n} x_{j,i} = \sum_{j=1}^{n} x_j$$

After the protocol the server holds the sum of the inputs x_i. The protocol can be made secure against malicious deviations by the subscribers X_i which is detailed in [22]. The protocol requires a constant number of rounds, but has a communication complexity per participant of $O(n)$, i.e. linear in the number of participants.

The protocol is secure against collusion of up to $n-1$ clients (without the server), even in the malicious model. The server is assumed to be semi-honest and collusion with it are explicitly excluded, although not required

for the summation protocol by itself. The protocol reveals the public keys of each party, such that parties are identifiable via them as pseudonyms.

2.3.3 Implementations

The first published implementation of secure computation was the FairPlay system [55]. It implements the two-party protocol of [80]. It has a simple programming language to specify the computed function. A program is first translated into a register-free, feed-forward binary circuit which is then used in the protocol between the two parties.

The restriction of the circuit to be oblivious also places some restrictions on the programming language. In an oblivious circuit each gate is executed exactly once which e.g. forbids the use of registers common in processor architectures. It only allows loops with a constant number of iterations which are unrolled in the circuit. Functions are completely inlined and therefore recursion is not possible. Both branches of "if" statements are always evaluated. Most complex are array accesses with a variable index [55]. The resulting circuit's size is of the order of the size of entire array with a large constant. The constructed circuit is essentially a series of "if" statements comparing the index to a constant and returning the array element if they match.

The restrictions on the programming language can lead to difficult programs for certain functions. Take the following problem: Alice has input a and Bob has input b and they want to compute $((a + b)!)$. The resulting program is depicted in Figure 2.1. It is likely that it is easier to write a program that outputs this program's source code than to write the program itself.

```
int fac(int a, int b)
s := a + b
if (s == 0)
    out := 1
if (s == 1)
    out := 1
if (s == 2)
    out := 2
if (s == 3)
    out := 6
...
if (s == max)
    out := max!
return out
```

Figure 2.1: Factorial function

The reported performance figures of [55] are disappointing from a practical perspective, e.g. the reported time for comparing two 32-bit integers (for maximum) is 1.25 seconds over LAN network conditions and 4.01 seconds over WAN network conditions.

The first multi-party secure computation implementation is [74]. It implements the protocols of [7], although it contains many tools for efficiently implementing distributed constraints problems. Unfortunately there is not much scientific literature about the implementation. The protocols of [7] uses a full-mesh communication pattern with a number of rounds linear in the size of the circuit. It calls itself an interpreter as opposed to a compiler of [55]. No performance figures are reported and the provided examples use key sizes that prohibit useful performance measures, since they are breakable using a pocket calculator. No information is available on the security model used.

Another multi-party secure computation implementation is currently being developed. It is partly based on the protocols in [15]. These protocols implement a full-mesh communication pattern with a number of rounds linear in the circuit size. It also compiles a domain-specific programming language into a circuit that is being executed. The circuit is arithmetic as opposed to binary in [55], i.e. it operates on integers. The programming language is described in [61] and extends the functionality of [55], e.g. it allows loops on values known to all participants.

Furthermore the system implements a server model, where many clients submit their input to a fixed number of servers. The privacy of the computation is maintained as long as only a minority of the server is corrupted. First performance figures for an auction site are reported in [8]. No overall performance figures are reported, but they benchmark individual operations, such as addition, multiplication, division and comparison. The authors indicate that they expect a practical implementation of an auction system to be feasible given the performance results. The system currently only implements the semi-honest security model.

The multi-party secure computation using two servers of [60] was implemented for a survey in [30]. The protocols of [60] use a constant number of rounds, but have communication cost linear in the size of the circuit. This system uses two computation servers which are mutually distrustful and must be (semi-)honest. The clients submit their inputs via a special Oblivious Transfer protocol to one of the servers. Computing the results of surveys is very similar to benchmarking in that both compute statistics, such as average, maximum and median of their input values. No absolute performance figures are reported, but the authors also expect their system to be practically feasible, although they anticipate memory to be the main limiting resource due to the storage of the entire circuit. They currently only implement the semi-honest security model.

A performance evaluation of computing statistics in a query over a

database has been conducted in [76]. They send an encrypted query to the database, essentially a 0 or 1 for each entry, and the server evaluates this query, such that it does not learn which values are queried, i.e. it multiplies each entry with either the encrypted 0 or 1 and sums the result using a homomorphic encryption system. The authors report that this method is infeasible compared to current database technology. Particularly handicapping is the fact that every element of the database must be evaluated.

In summary, all implementations of secure multi-party computation that are practically evaluated [8, 30] use a server model. They separate input clients from servers to simplify communication. [30] uses two servers which must be (semi-)honest and [8] uses 3, 5, or 7 servers where a minority can be faulty. All implement the semi-honest security model and none report figures on overall examples taking network conditions into account.

2.4 Cryptographic Building Blocks

This section describes the cryptographic primitives necessary to understand the protocol. In particular, the benchmarking protocol uses two advanced techniques: homomorphic encryption and Oblivious Transfer (OT).

2.4.1 Homomorphic Encryption

In homomorphic encryption one operation on the cipher texts maps into another operation on the plain texts. There are many homomorphic encryption schemes, including the popular RSA (Rivest-Shamir-Adleman) encryption scheme [67]. In RSA multiplying two cipher texts (modulo the RSA key modulus) results in an encryption of the product of the two plain texts (modulo the RSA key modulus), i.e. in the RSA case multiplication maps into multiplication.

For the benchmarking protocols the operation on the plain texts must be addition, i.e. one can secretly add two plain texts. The first encryption system with this property was by Goldwasser and Micali [39], but worked only for bit plain texts. Plain text length was at most one bit and addition was modulo 2, i.e. equal to the exclusive-or operation. The Goldwasser-Micali encryption system is also the first semantically secure encryption system. Goldwasser-Micali encryption with X's public key is denoted as $E^b_X(\cdot)$ and the corresponding decryption as $D^b_X(\cdot)$.

Encryption systems that have the additive homomorphic operation, but on (small) integers are [5, 20, 56, 62, 63]. In particular, we use Paillier's [63] encryption system which was later generalized to a threshold encryption system by [20]. Let $E_X(\cdot)$ denote the encryption with X's public key and $D_X(\cdot)$ denote the corresponding decryption. Formally the homomorphic encryption system used needs to have the following property:

$$D_X(E_X(x) \cdot E_X(y)) = x + y$$

From which the next property can be easily derived:

$$D_X(E_X(x)^y) = x \cdot y$$

The modulus of the two operations (on the plain text and on the cipher text) is different, but omitted here. In Paillier's encryption systems [63] it can be chosen large enough, such that it does not affect the calculations made in the protocol. These operations allow a party, in the case of the benchmarking protocols the service provider, to operate on the plain texts without knowing them.

The encryption system used in the benchmarking protocol does not only need to be homomorphic, but also semantically secure (indistinguishability against the chosen plaintext attack – IND-CPA [36]) in order to be provably secure and public-key in order for the service provider to be able to encrypt values and participate in the protocol. This implies that each plaintext can be encrypted as many (randomized) ciphertexts and it then becomes computationally hard to distinguish one encryption of the ciphertext from any other such encryption, since they are not efficiently enumeratable. Public-key encryption systems were introduced in [23] and have the property that one can encrypt without being able to decrypt. Since the invention of RSA [67] public-key encryption systems have become very popular and its details are not described here.

The security properties of semantic security are explained in more detail in Section 4.3.1. Semantic security is usually achieved by randomizing the cipher text, i.e. one plain text has many possible cipher texts. The following previously cited encryption systems [20, 56, 62, 63] are all public-key and semantically secure.

Homomorphic, semantically secure encryption schemes have another important operation: re-randomization. In re-randomization a party is given a cipher text (possibly without the private decryption key) and its goal is to find another encryption of the same plain text. In homomorphic encryption this can be achieved by "adding" the neutral element (0).

$$E_X(x) \cdot E_X(0) = E'_X(x)$$

However it holds with very high probability that

$$E_X(x) \neq E'_X(x)$$

For the two cipher texts $E_X(x)$ and $E'_X(x)$ the cipher text indistinguishability property holds. This operation can be very helpful when cipher texts are sent in the protocol, but it should not be known that a plain text identical to a previous one is returned.

2.4.2 Oblivious Transfer

Oblivious Transfer (OT) is a cryptographic protocol, with which a sender can transfer one of several messages to a receiver without the sender knowing which message was sent. OT was introduced in [66] and generalized to 1-out-of-2 OT in [29] which has been proven equivalent in [17]. In 1-out-of-2 OT Alice (the sender) has two bits a_0 and a_1 and Bob has one bit $b \in \{0, 1\}$. After the execution of the 1-out-of-2 OT protocol Bob knows a_b, but nothing about $a_{\neg b}$ and Alice has learnt nothing, in particular not b. The notation for an OT protocol between Alice as a sender and Bob as a receiver is:

$$A \xrightarrow{\text{OT}} B$$

OT is a very powerful protocol and secure computation [34, 37] and cryptographic primitives [50] can be based on it. Nevertheless it is very computation-intensive [59], since it requires modular exponentiation, and is the main reason for the low performance of the practical implementation of secure computation in [55]. It therefore should be used sparingly.

In 1-out-of-n OT Alice has n messages for Bob to choose from. It can be implemented using 1-out-of-2 OT [58], but many practical OT protocols have natural extensions for 1-out-of-n OT, e.g. [59]. The fastest known implementation of OT is described in [59]. Its most efficient version was proven secure under the (computational and decisional) Diffie-Hellman assumptions in the random oracle model and the benchmarking protocol implementation uses it.

2.4.3 Cryptographic Hash Functions

A hash function is a function that is easy to compute and compresses an input x of finite arbitrary length to an output $H(x)$ of fixed finite length [65]. Cryptographic hash functions provide pre-image resistance, i.e. given a cryptographic hash $H(x)$ (with unknown x) it is computationally infeasible to find a value for x. In collision-resistant hash function it is computationally infeasible to find any two distinct inputs x and x', such that $H(x) = H(x')$.

A message authentication code (MAC) is a cryptographic hash function parameterized by a secret key k. More importantly, MACs provide computation-resistance, i.e. given any number of authenticated texts $\langle x_i, MAC(x_i, k)\rangle$ it is computationally infeasible to compute another authenticated text $\langle x, MAC(x, k)\rangle$ $(x \neq x_i)$ without knowing the key k. A successful attempt of producing an authenticated text is called MAC forgery.

2.5 Service Oriented Architecture

Service Oriented Architectures [43] (SOA) are concerned with the composition and management of services into distributed applications. Services are

high-level abstractions of application components independent of their implementation details, e.g. programming language. The architecture of the benchmarking platform is a service oriented architecture that is composed of several services.

This chapter describes the architecture of the benchmarking platform as a SOA and its components (services). A specific aspect of the architecture, the benchmarking protocols, have been described in Chapter 4. The participants of the protocol, i.e. subscribers and service provider, suitably map to services in a SOA. Parties in a cryptographic protocol are autonomous entities that (distrustfully) interoperate, just as services do in a SOA. The individual steps performed locally at a party are therefore combined and their interface is offered as a service.

The composition of those services is guided by the protocol specification of Section 4.2. The semantics of the service interfaces is also strictly given by the protocol specification. SOA helps in locating and interoperating the services. The details of service implementation are described in the next section.

Due to the cryptographic requirements of the benchmarking application, services are rather tightly coupled. SOAs call for loose coupling [43], but cryptographic services cannot simply be replaced by another service unless that service implements the same protocol. Although individual services as they are deployed at different locations (companies) are loosely coupled in the SOA, the semantics of the service are rather fixed. The architecture of the benchmarking application does not use advanced service composition of SOAs.

2.5.1 Web Services

Web services are an implementation of the service concept over Internet protocols and standards. A web service is an application component identified by a Universal Resource Identifier (URI) and capable of exchanging messages via Extensible Markup Language (XML).

XML is a standardized document description format that can be used to structure data and message exchanges. It groups the document's contents into hierarchical elements by markups. Figure 2.2 shows an example of an XML element with an attribute, value pair.

```
<element attribute="value">data</element>
```

Figure 2.2: XML data element

XML is extensible, because it allows the creation of customized markups. More details of XML are contained in its specification [10].

Web service interfaces are described using the Web Service Definition Language (WSDL), an XML extension. A WSDL description contains the

methods (operations) a web service offers and their parameters (messages). It also describes the bindings, i.e. names the protocols over which the service can be contacted, e.g. SOAP. The details of WSDL can be found in its specification [12].

Simple Object Access Protocol

The Simple Object Access Protocol or Service Oriented Architecture Protocol (SOAP) is the protocol for encoding messages for web services. It is usually a layer on top of the Hypertext Transfer Protocol (HTTP) and is therefore an extension to the network stacks with application as the top layer (HTTP). It is XML-based. SOAP messages are divided into header and body. The body contains the encoded parameters of the message and the header contains meta information about the message. Detailed information about SOAP can be found in [41].

2.5.2 Web Service Security

Web Service Security [57] (WS-Security) is a standard for establishing secure, i.e. confidential, and authenticated channels via SOAP. It specifies the encryption and signatures methods and procedures for web service calls. It is the main means of communication in the benchmarking protocol implementation.

WS-Security uses XML Encryption to encrypt the message contents and achieve confidentiality. Only the SOAP body (or even only a part of it) is encrypted and the SOAP header is left in plain text for message handling, e.g. routing. Authenticity and integrity of the message are achieved using XML Signature. By default the entire SOAP body is signed and the signature is stored in the header as a detached signature. Furthermore, WS-Security specifies how to include cryptographic tokens, e.g. certificates, in the SOAP message. These are included in the header as well.

XML Encryption

XML Encryption can be used to encrypt entire XML documents, selected elements (and their descendants), or the data contents of an XML element. XML Encryption replaces the encrypted content with an `<EncryptedData>` element. Figure 2.3 shows the minimal structure of such an element.

The `<CipherValue>` element contains the cipher text. It may be replaced with a reference to the cipher text in the XML document. Further optional elements indicate the encryption method, i.e. the encryption algorithm, and the key used. XML Encryption allows for several key management methods, e.g. encrypting the contents with a symmetric key and then encrypting the symmetric key with a public key. This speeds up encryption, since symmetric encryption is considerably faster than public key encryption.

```
<EncryptedData>
    <CipherData>
        <CipherValue>...</CipherValue>
    </CipherData>
</EncryptedData>
```

Figure 2.3: XML encryption

All details of XML Encryption can be found in its specification [26].

XML Signature

XML Signature can be used to sign entire XML documents, selected elements (and their descendants), or even external objects. Signatures in XML therefore are a two-step process: first the to-be-signed contents is hashed and these hashes (along with some meta information) are hashed again. The final hash is then signed.

The signed contents is referenced from the signature by an URI. URIs can reference outside the document of the signature or elements inside that document. For elements inside the document those elements are furnished with an identifier. An example is depicted in Figure 2.4.

```
<element id="123">data</element>
```

Figure 2.4: XML identifier

Such an element is later referenced via an `<Reference>` element. Figure 2.5 shows a reference to the element from Figure 2.4.

```
<Reference URI="#123">...</Reference>
```

Figure 2.5: XML reference

The minimal structure of an XML Signature is depicted in Figure 2.6. The reference contains the hash (digest) of its contents, such that as described above all hashes can be hashed again. The signature is then stored in the `<SignatureValue>` element.

Since XML separates contents from structure, the exact format, e.g. white spaces, line breaks, etc., of an XML document are arbitrary and do not contribute to its contents. XML Signature therefore enforces a strict format, called Canonicalized XML, before signing the contents. More details on XML Signature can be found in its specification [27].

```
<Signature>
     <SignedInfo>
          <Reference>
               <DigestValue>...</DigestValue>
          </Reference>
     </SignedInfo>
     <SignatureValue>...</SignatureValue>
</Signature>
```

Figure 2.6: XML signature

2.6 Summary

This section has described the related work for the privacy-preserving benchmarking platform.

First, similar applications have been considered. There are a few systems for Internet-based benchmarking for special groups. None of the systems implements privacy protection for the KPIs, although some mention privacy as important.

All systems have been implemented for specialized group, such that the problem of peer group formation has not been considered yet. The closest related data clustering algorithm (k-means clustering) has been explained.

Second, literature on secure computation has been reviewed. There is a large number of general protocols that can implement any function. In particular, protocols that can be implemented in a centralized client-server model have been considered. There is no general protocol which currently works with only one central server as economically required by a single service provider. General protocols tend to be slow and [38] suggests that special protocols should be built for important problems.

The next section has reviewed special protocols for statistics computation. There are protocols for sophisticated statistics, in particular division has been considered in many papers. Of these protocols none implements a central server model. For the central server models protocols for maximum and sum have been discovered. These protocols require linear communication cost per participant. No central server protocols for median has been described yet. Anonymity of the subscribers as facilitated by the central communication pattern has not been considered in any publication yet.

Third, some building blocks used in the protocols have been described. Important building blocks are homomorphic encryption, Oblivious Transfer, and cryptographic hash functions. Homomorphic encryption maps multiplication of cipher texts $E(x)$ and $E(y)$ to addition of plain texts x and y: $E(x) \cdot E(y) = E(x + y)$. The homomorphic encryption systems used ($E_X(\cdot)$ and $E^b_X(\cdot)$) are public-key and semantically secure. $E_X(\cdot)$ operates on integers while $E^b_X(\cdot)$ operates on bits. Oblivious Transfer is a protocol for

transmitting one out of n bits, such that the sender does not know which bit has been transmitted and the receiver only learns the selected bit. The communication complexity of oblivious transfer is $O(n)$. Cryptographic hash functions map arbitrary-length inputs to fixed-length outputs, such that the process cannot be reversed, i.e. given a hash $H(x)$ it is impossible with very probability to find a valid pre-image x.

Fourth, the architecture components of the implementation have been described. The implementation is realized as a service-oriented architecture based on web services. Web services are implementations of services over Internet protocols and standards. The encoding protocol for web service calls is the Simple Object Access Protocol (SOAP).

Web services offer the establishment of secure, authenticated channels using the WS-Security standard. WS-Security uses XML encryption and signature to modify the elements of the SOAP message, such that its confidentiality and integrity are protected.

3 Problem Outline

This chapter describes the problem of a privacy-preserving benchmarking platform in detail. It covers functional and non-functional requirements, the service provider model, privacy requirements and their implications. It also outlines the limitations of practically feasible privacy in benchmarking protocols.

3.1 Non-Functional Requirements

The benchmarking platform should be designed for practical, real-world application. It needs to support a realistic number of customers, KPIs and peer groups. It should support $100,000$ customers and 200 KPIs. These numbers are based on current (year 2007) market conditions and business applications.

Communication and computation cost should be aligned to today's (year 2007) network conditions and computer hardware. The entire communication cost for one customer should not exceed 10 MB and the entire protocol should be able to finish in less than 24 hours, if it does not interact with a user. These numbers are no fixed requirements, but rather a guideline. The lower communication and computation cost can be kept, the more realistic is the use of the protocol in real-world applications.

3.2 Statistics

A KPI is a floating-point number computed according to a common basis, such that their inter-company comparison is useful. Such KPIs can be readily extracted from modern Enterprise Resource Planning (ERP) systems. Let

v_i	denote the i-th element of vector $\vec{v}$, e.g. γ_{j,k_i} the i-th element of $\vec{\gamma}_{j,k}$.
X_i $(i = 1, \ldots, n)$	denote one of n customers.
SP	denote the service provider.
$x_{i,k}$	be customer X_i's value of the $k-th$ KPI.
$\vec{\beta}_k$	$= (x_{1,k}, \ldots, x_{n,k})$.

Recall that a peer group is the group of companies that wants to benchmark against each other. Let

$Q_j \subset \{1, \ldots, n\}$	denote the participants of the $j-th$ peer group.
$q_j = \lvert Q_j \rvert$	denote the size of the $j-th$ peer group.
$\vec{\gamma}_{j,k}$	be the subsequence (as a vector) of $\vec{\beta}_k$.
$\vec{\gamma}_{j,k}$	$= (x_{i,k} \lvert i \in Q_j)$.
$\overrightarrow{\gamma_{j,k}}$	have the same elements as $\vec{\gamma}_{j,k}$ and be sorted in ascending order.

Each customer X_i provides his KPIs $x_{i,k}$ as input. The benchmarking platform should provide the following statistics of each KPI to its customers to compare to their KPIs. That means the benchmarking platform provides the statistics mean, variance, maximum, median and best-in-class for each $\vec{\gamma}_{j,k}$ to all $X_i \in Q_j$ as output. The benchmarking platform also receives the statistics for each peer group as output. A description of the model of the subcomponent benchmarking protocol can be found in Section 4.2.1. The formulas for the statistics are given below:

1. Mean: $\mu_{j,k} = \frac{\sum_{i=1}^{q_j} \gamma_{j,k_i}}{q_j}$
2. Variance: $\sigma_{j,k}^2 = \frac{\sum_{i=1}^{q_j} (\gamma_{j,k_i} - \mu_{j,k})^2}{q_j - 1}$
3. Maximum: $max_{j,k} = \max(\vec{\gamma}_{j,k})$
4. Median: $median_{j,k} = \overrightarrow{\gamma_{j,k}}_{\lceil \frac{q_j}{2} \rceil}$
5. Best-In-Class: $bic_{j,k} = \frac{\sum_{i=\lceil \frac{3}{4} q_j \rceil}^{q_j} \overrightarrow{\gamma_{j,k}}_i}{\lceil \frac{1}{4} q_j \rceil}$

The mean is the arithmetic average of the KPIs $\vec{\gamma}_{j,k}$. The variance is a statistical measure for the deviations in the KPIs $\vec{\gamma}_{j,k}$. The maximum is the maximum value of the KPIs $\vec{\gamma}_{j,k}$. The median is the middle element in a sorted list, i.e. half of the elements are larger and half of the elements are smaller. Then, $median_{j,k} = \overrightarrow{\vec{\gamma}_{j,k}}_{\lceil \frac{q_j}{2} \rceil}$. Best-in-class is the mean of the top 25%. Best-in-class computation has elements from mean and median computation.

3.3 Service Provider Model

A service provider is an organization offering a benchmarking service to his customers. The service provider may control several servers running the service or parts of it, but all of them form one administrative domain. The characteristic property of an administrative domain is that one (legal) person has access to all data and computations made in that domain. Privacy-preserving computation in multiple administrative domains can be addressed

using existing secure multi-party computation protocols. Customers are organizations in a different administrative domain that are interested in using the benchmarking service. A set of protocols describe the interactions and messages sent between customers and service provider. Customers receive only the messages specified in the protocols and vice versa the service provider only receives messages as specified in the protocols.

The benchmarking service includes providing the subscriber X_i statistics of the KPIs of the peer group or peer groups she is a member of (i.e. $X_i \in Q_j$), such that she is able to perform the benchmarking process by comparing these statistics to their KPIs. The service provider initiates a statistics computation protocol at his discretion, such that customers can retrieve the statistics, i.e. result of the computation protocols, from the platform in a separate retrieval protocol.

The service provider model is characterized by two restrictions that are placed on the protocols and their communication between the participants.

First, the customers are only to communicate with the service provider. The only communication supported is point-to-point, i.e. between two participants, and one of the participants must be the service provider. This leads to a centralized communication pattern.

Second, the customers are to remain anonymous amongst each other. Two notions of anonymity can be distinguished: strong and weak anonymity. The requirement for strong anonymity is that customers do not know or refer to any static identifier of other customers. Such an identifier can be the name of the organization, a (static) IP address, a (static) pseudonym or similar information that uniquely identifies the organization. Using the terminology for anonymous communication from [64] unobservability is defined as not being able to determine whether a customer participates in the benchmarking platform or not. The unobservability set is the set of all customers and the attacker is a (subset of) customer(s). The service provider and parties with direct network access to other parties' communication links are excluded from the set of possible attackers. This definition is most closely related to our definition of strong anonymity. [64] defines anonymity as the state of being not identifiable within a set of subjects. This definition is not strong enough for the benchmarking platform's purposes, since it only protects the identity of the subjects. In [64] unlinkability of two or more items means that within a system, these items are no more and no less related than they are related concerning the a-priori knowledge. A particular concern for the benchmarking platform is the unlinkability of two customers after two runs of the benchmarking protocol. This is only included in the unobservability definition of [64] and not its anonymity definition. For weak anonymity a customer may know and refer to each with a (static) pseudonym that is not linkable to any other identifying information except by a trusted third party. It is most similar to a role pseudonym in [64], which is initially unlinkable (except by the service provider). According to [64] pseudonymity provides

some anonymity, but to a varying degree. If the pseudonyms remain unlinkable to the identity of the subscribers, it at least protects the anonymity (as in the definition of [64]) of the customers.

Strong anonymity implies weak anonymity, but not vice versa. Given the need for anonymity a strongly anonymous system is therefore preferable.

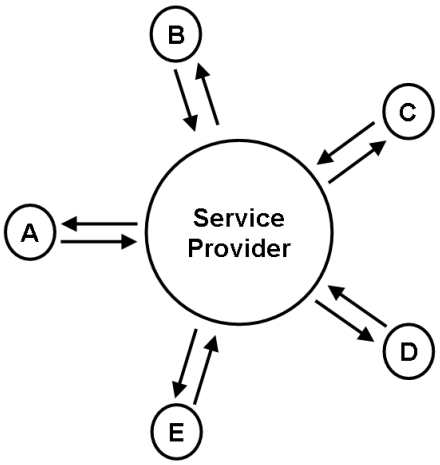

Figure 3.1: Service provider model

Figure 3.1 provides a visualization of the service provider model in an example with five companies (A-E) and the service provider.

3.4 Privacy Requirements

A feature of the benchmarking service is that it protects the privacy of the customers. Not only do they remain (strongly) anonymous amongst each other, but also their (individual) KPIs are not leaked to either the service provider or another customer. All participants only learn what can be inferred from their input and the output. More formal definitions follow in Section 4.3.2.

Let protocol Π denote a protocol for computing benchmarking statistics in the service provider model. Without loss of generality assume there is only one peer group. The definition of privacy follows standard cryptographic techniques. For reference see [34, 36]. The privacy obtained when using protocol Π is compared to the privacy in the ideal model. X_i's input (KPI) is denoted by $x_{i,k}$ and let $\vec{\beta}_k = (x_{1,k}, \ldots, x_{n,k})$ be the inputs for the k-th KPI. The service provider has no input. In the ideal model there exists a trusted third party TTP and all X_i send $x_{i,k}$ to TTP which sends the statistics to the service provider SP. The customers may then retrieve the statistics in a separate protocol. An attacker may subvert any number of

parties in the real model of protocol execution or in the ideal model (except TTP). We compare the abilities of an attacker when he subverts the same set of parties in the real and ideal model.

Definition 3.1 *Protocol* Π *privately computes the benchmarking statistics if an attacker in the real execution of* Π *can compute the same information as in the ideal model.*

3.4.1 Database View

The inputs and outputs of the ideal model can be viewed as database tables. The customers each have one tuple of a horizontally partitioned, distributed database table of individual KPIs. Such a tuple consists of $\langle id_{customer}, id_{peergroup}, id_{KPI}, KPIvalue \rangle$. Let $stat_1$, $stat_2$, $stat_3$, ... denote the statistics computed. After the statistics computation protocol the service provider has a database table with the following tuples: $\langle id_{peergroup}$, id_{KPI}, $stat_1$, $stat_2$, $stat_3$, $\ldots \rangle$. This database table can be considered public, while the distributed table of KPIs is considered sensitive and private. The achievable privacy properties of peer group participation is discussed in Section 3.5. Figure 3.2 shows an overview of the databases in the system.

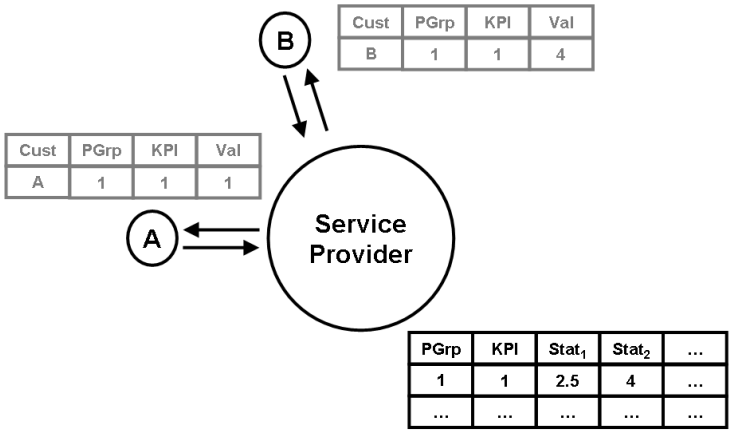

Figure 3.2: Database view

Recall that micro-data release is the release of individual, potentially anonymized tuples of a database. One can compare the privacy-preserving benchmarking statistics computation approach of revealing the peer group identifier ($id_{peergroup}$) to a k-anonymous [70, 77] micro-data release where the quasi-identifiers characterize the customer ($id_{customer}$). Both provide at least the same degree of anonymity and privacy protection where k is equal to the size of the smallest peer group, since the peer group identifier

is equivalent to the anonymized quasi-identifiers. Furthermore our privacy-preserving benchmarking statistics computation replaces the values of the individual KPIs with aggregated statistics, such that at least the same degree of privacy is achieved.

3.4.2 Economic Motivation

The privacy requirement of the benchmarking platform is designed for the economic advantage of the service provider. Two advantages can be separated: customer acceptance and competitive advantage.

Privacy is anticipated to increase customer acceptance. The intuition is that customers are reluctant to disclose business critical data and private benchmarking can alleviate the risk. This in turn leads to a larger market size and in the last consequence to larger revenue.

Privacy can also provide a competitive advantage. The risk and cost of sharing KPIs to engage in benchmarking can be lowered by privacy, thereby offering a higher benefit to customers, and justifying a higher price or increasing market share.

The difference between increased customer acceptance and competitive advantage is that increased customer acceptance increases the overall market while the competitive advantage increases market share.

Also given the possibility of privacy-preserving benchmarking with similar results to and the same price as non-privacy-preserving benchmarking, there is no reason to engage in non-privacy-preserving benchmarking [46].

3.4.3 Privacy against Service Provider

The participants of the benchmarking platform are the customers and the service provider. Privacy as a competitive advantage between service providers requires privacy against the service provider in the statistics computation protocol, since the customers are to retrieve only the statistics (and not the individual) KPIs anyway. In relation to definition 3.1 the service provider is a potential attacker.

The service provider model's central communication pattern complicates resistance against a faulty or malicious service provider. If the service provider stops the protocol before it completes, there is no way for the customers to recover, since they are not to communicate with each other directly.

Service Provider as an Attacker

Privacy against the service provider implies that the service provider can be subverted by an attacker. At the very low level of the protocol this means that message security must be ensured in the presence of an attacking service provider.

The central communication pattern also implies that a service provider subverted by an attacker controls the entire network. The attacker can read, write, alter or forge any message. This setup corresponds to the setup in the cryptographic problem of establishing secure channels over an insecure network. Such a secure channel can be used to exchange messages between customers which cannot be read or altered by the service provider (or any other customers).

If the customers are to remain strongly anonymous, they cannot establish such a channel, since they would not know where to send the message to. This rules out any existing solution that requires pair-wise secure channels between the participants for a strongly anonymous system. In a system with weak anonymity, cryptography still provides no tools or protocols with which two customers could establish a secure channel without a third party trusted by both of them. This implies that every solution that requires pair-wise secure channels for a weak anonymous system requires another trusted third party besides the service provider.

3.5 Peer Group Models

Peer group formation is the process of computing the peer groups from the set of customers at a given point in time. Peer group formation creates a mapping between customers and peer groups. A peer group has a minimum size for its statistics to be meaningful in the benchmarking process. This minimum size is larger than one, since a customer wants to compare to its competition and not just itself. Therefore a peer group always maps to multiple customers. A customer can be a member of one or more peer groups. Two peer group models can be distinguished: single and multiple.

In the single peer group model the customer is part of exactly one peer group. In the multiple peer group model the customer can be part of one or more peer groups. The peer group model has implications on the communication pattern and the privacy of the KPIs.

3.5.1 Single Peer Group Model

In the single peer group model the customer maps to one and only one peer group. The privacy of the KPI after statistics computation is protected by the size of the peer group. If the statistics computation is private, no individual KPI is being leaked during computation.

As in every privacy-preserving system there is a trade-off between utility and privacy. In Section 3.4 the privacy requirements according to the definition of [34] have been outlined, but what can be inferred by the output of the system cannot be protected. Given the five statistics computed over $\overline{\vec{\gamma}_{j,k}}$ one can build a system of five equations. If the size of the peer group is

at least six, there always remains one degree of freedom for an attacker (including the service provider) to determine $\overline{\vec{\gamma}_{j,k}}$, such that the inputs always remain private. Some elements of $\overline{\vec{\gamma}_{j,k}}$ are leaked by the output, such as the maximum and the median, but not the party that provided them as input. This implies that the output of a benchmarking protocol does not breach privacy.

It remains to show that the output of the entire system does not breach privacy in the single peer group model. In the single peer group model the service provider who is the only participant given the output of all peer groups is also only given those five equations for each peer group. No additional information about any KPI $x_{i,k}$ of any participant X_i can be inferred by the output of the system. Therefore no one including the service provider can determine $\overline{\vec{\gamma}_{j,k}}$ in the single peer group model. The conclusion is that the single peer group model is sufficient for privacy. The privacy of the benchmarking platform reduces to privacy of the benchmarking protocol, i.e. the computation of the statistics.

The service provider may know the peer group of a customer without a privacy breach. The service provider only needs to contact the customer of a peer group to compute its statistics, since the non-inclusion does not reveal information the service provider does not have already.

Let o be the number of KPIs, p the number of peer groups, n the number of customers and q_j the number of customers in the j-th peer group. In the single peer group model, it holds that

$$n = \sum_{j=1}^{p} q_j$$

The single peer group model's statistics computation protocol communication cost is lower bounded by $\Omega(no)$, since every customer needs to send its KPIs (or a value computed with it) to the service provider.

3.5.2 Multiple Peer Group Model

In the multiple peer group model the customer can be a member of more than one peer group. The privacy of the customer's KPI is at risk, if the service provider knows which customer participates in which peer group.

Denote customer's X_i participation in peer group j by $\lambda_{j,i} = 1$, else $\lambda_{j,i} = 0$. Let Λ denote the matrix of $\lambda_{1,1}, \ldots, \lambda_{p,n}$. Figure 3.3 shows an example of such a peer group participation matrix.

Let $x_{i,k}$ be customer X_i's value of the $k-th$ KPI and Q_j be the members of the j-th peer group. Recall $\vec{\beta}_k = (x_{1,k}, \ldots, x_{n,k})$. The computation of the sum of a KPI per peer group can be written as

$$\vec{sum}_k = \Lambda \vec{\beta}_k$$

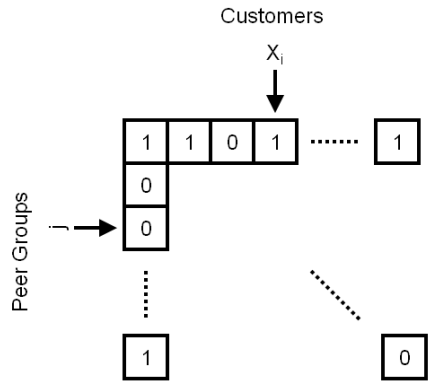

Figure 3.3: Peer group participation matrix

The computation of the sums $s\vec{u}m_k$ for all peer groups for the $k-th$ KPI is equivalent to the computation of average, if the values in Λ are divided by the peer group size or the peer group size is known and the results of the matrix multiplication are later divided by the peer group size locally by each subscriber.

If Λ (or a subset of rows of Λ) are invertible, then

$$\Lambda^{-1} s\vec{u}m_k = \vec{\beta}_k$$

Since $s\vec{u}m_k$ is public, an invertible Λ must remain private for $\vec{\beta}_k$ to remain private. The communication pattern of the statistics computation protocol then must also not reveal an invertible Λ. In the service provider model with one service provider during statistics computation every customer therefore has to participate in the protocol for computation of every peer group. The multiple peer group model's statistics computation protocol communication complexity is thus lower-bounded by $\Omega(nop)$.

Other methods to prevent inversion of Λ may have lower communication complexity. Also other methods to maintain privacy of the peer group participation matrix may have lower communication complexity. E.g. [52] suggests two service providers during statistics computation, one which computes the statistics and one which maintains the peer group participation.

3.5.3 Conclusion

A necessary condition for Λ to be invertible (or pseudo-invertible) is $p \geq n$. This never holds in the single peer group model, but may hold in the multiple peer group model. The output of the multiple peer group model therefore may violate privacy, while the output of the single peer group is private. In summary, the possibility of invertability is sufficient for non-privacy, while

the single peer group model is sufficient for privacy. Peer group formation uses non-sensitive criteria that might even already be public, as input. A regular, non privacy-preserving computation of peer groups is therefore preferable, since its computation and communication cost is lower.

For the statistics computation the single peer group model offers a better lower bound on the communication cost. Considering the non-functional requirement for the number of customers and $p = o(n)$, the multiple peer group model places high burdens on the practicality of the protocols. Assume that only one encrypted value of size 256 bytes needs to be transferred per KPI, peer group and customer. For 200 KPIs and $100,000$ customers, one customer must transfer over 4 GB per peer group. This can be considered impractical under current network conditions. In this dissertation the single peer group model is chosen in order to maintain the single service provider during statistics computation and to avoid more complicated restrictions on the inversion of the peer group participation matrix.

3.6 Summary

This chapter explained the functional and non-functional requirements for the benchmarking platform. It described what is referred to as a benchmarking platform. A benchmarking platform is a benchmarking service offered in the service provider model, i.e. central communication pattern. The benchmarking platform requires a statistics computation protocol. The statistics for benchmarking included in this protocol should be mean, variance, maximum, median, and best-in-class.

The benchmarking platform is characterized by two features: practical and privacy-preserving. The parameters for the benchmarking platform to be practical in a real-world setting have been defined. It should support $100,000$ customers for 200 KPIs with limited available bandwidth and computation power. Privacy for benchmarking has been defined by comparison to the ideal model where parameters are sent to a trusted third party. Loosely speaking, privacy means keeping the individual KPIs secret from service provider and other customers. The implications of privacy on peer group formation have been shown and evaluated against the practicality requirement. In conclusion, in order to keep the platform practical and private, a customer may participate in only one peer group (referred to as the single peer group model).

4 Protocols

This chapter describes the protocols run by the service provider and subscribers to implement the privacy-preserving benchmarking service. At its core there is the benchmarking protocol in Section 4.2. The benchmarking protocol computes the required statistics in a privacy-preserving manner, i.e. without revealing the subscribers' inputs. It is the benchmarking protocol that provides the platform its privacy-preserving property. The benchmarking protocol's prerequisites in terms of key distribution and security model are described in Section 4.1. The key distribution can be achieved using a trusted third party, e.g. in the form of an enhanced certificate authority. Section 4.1.2 describes the protocols involved.

The theoretical analysis of the benchmarking protocol is conducted as security proofs in Section 4.3. The security models for the proofs and the proof techniques are aligned with the work of [34]. Clarification of the security assumptions necessary to prove the security has been emphasized. Section 4.3.1 lists the assumptions made in the proofs.

4.1 Security Model

The security model covers the result and process of key distribution and its implications. It describes the entities involved, the protocols between them to setup the system and the restrictions on allowed collusion. It builds the basis for the security analysis and proofs of the protocols in the later sections.

4.1.1 Key Distribution

Three keys play a central role in the benchmarking platform. First, the public and private key pairs corresponding to the operations $\langle E_{common}(\cdot), D_{common}(\cdot)\rangle$ and $\langle E^b_{common}(\cdot), D^b_{common}(\cdot)\rangle$ in the homomorphic encryption schemes where the first operates on integers and the second on bits and second, the symmetric key s_{common} for the message authentication code. All subscribers know $D_{common}(\cdot)$ and $D^b_{common}(\cdot)$, but the service provider does not, i.e. the service provider cannot decrypt values encrypted with $E_{common}(\cdot)$ or $E^b_{common}(\cdot)$. Similarly, all subscribers know s_{common}, but the

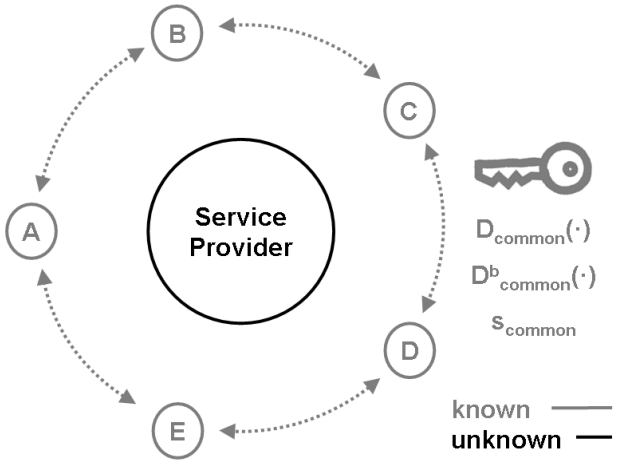

Figure 4.1: Key distribution

service provider does not, i.e. the service provider cannot compute message authentication codes. The common public keys for encryption $E_{common}(\cdot)$ and $E^b_{common}(\cdot)$ are known to all participants, i.e. the service provider can encrypt values. The key distribution is depicted for an example of five subscribers in Figure 4.1.

Attacks where the service provider and a subscriber collude are excluded, since it breaks the basic security assumption that the service provider does not have access to the common decryption keys $D_{common}(\cdot)$ or $D^b_{common}(\cdot)$. In case of such an attack the benchmarking platform provides no security. On the one hand, the intention of the service provider to collude can be excluded economically, since if it is detected, the harm may exceed all benefits of the benchmarking platform and spread to other areas of the service provider's business. On the other hand, the service provider may not intentionally be involved and a malicious subscriber or attacker may simply publish the key. In order to make key extraction harder some methods proposed in the literature may be used [13, 14]. Furthermore an effective re-keying procedure should be in place. Recall that all subscribers remain (strongly) anonymous amongst each other, i.e. they do not even refer to each other by an unique public key. Anonymity additionally raises the bar for collusion attacks of subscribers, such that such attacks do not necessarily need to be prevented. For those cases where the economics would favor a stronger secure solution we provide coalition-safe benchmarking in Chapter 8.

The communication between the service provider and the subscriber may contain values that are encrypted under the common encryption keys $E_{common}(\cdot)$ or $E^b_{common}(\cdot)$. Another subscriber wiretapping this communica-

tion could break the encryption and infer values she is not supposed to know. Therefore we assume secure, authenticated channels between all subscribers and the service provider.

These channels are established using standard techniques for secure communication over insecure networks. The session key establishment is performed using trusted public keys. Recall that the subscribers do not remain anonymous to the service provider for billing purposes. Trusted public keys are established using a public key infrastructure (PKI) [72, 75]. The standards used have been described in Chapter 2.

4.1.2 Registration

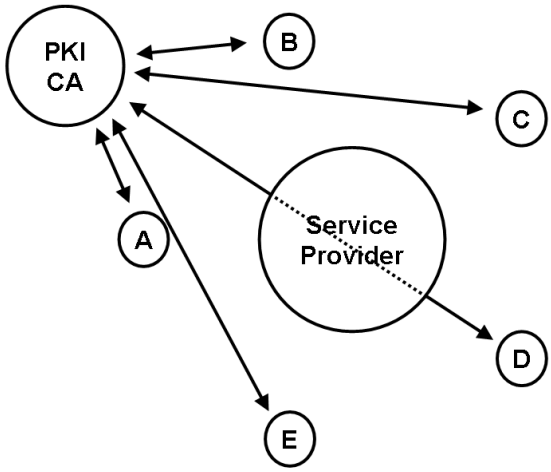

Figure 4.2: Registration with certificate authority

Registration is the process to be completed before the subscription to the benchmarking platform is completed. It involves establishing the trusted public keys $E_{X_i}(\cdot)$ and $E_{SP}(\cdot)$ between a subscriber X_i and the service provider SP. They use a certificate authority (CA) in the PKI to sign their public keys. The subscriber X_i needs to obtain his certificate during registration and therefore the subscriber and the CA need to communicate. This communication is limited to the registration phase only and no communication with the CA is necessary during the benchmarking process after registration.

Since the subscriber needs to contact the CA, the CA is also used to act as a dealer and distribute the common keys $D_{common}(\cdot)$, $D^b_{common}(\cdot)$ and s_{common} to the subscribers. Note that $E_{common}(\cdot)$ and $E^b_{common}(\cdot)$ are public parameters that can be posted on the service provider's web site. The CA has access to these keys and is therefore assumed not to collude with the

service provider. Fortunately many independent CAs currently offer their services, although their standard functionality needs to be extended.

Let $E_{CA}(\cdot)$, $D_{CA}(\cdot)$, $S_{CA}(\cdot)$ be encryption, decryption and signature with the CA's public or private key, respectively. Let $E(\cdot)$ and $D(\cdot)$ denote the encrpytion and decryption capability, i.e. in transport they denote the public or private keys. Let $Cert_{CA}(X_i, E_{X_i}(\cdot))$ be a public-key certificate issued by the CA to subscriber X_i. Figure 4.2 shows the one-time communication pattern for a registration with the CA.

Protocol

X_i	$\longrightarrow$	SP	$E_{SP}(X_i, E_{X_i}(\cdot))$
SP	$\longrightarrow$	X_i	$E_{X_i}(S_{SP}(timestamp, X_i))$
X_i	$\longrightarrow$	CA	$E_{CA}(S_{SP}(timestamp, X_i), E_{X_i}(\cdot))$
CA	$\longrightarrow$	X_i	$Cert_{CA}(X_i, E_{X_i}(\cdot))$,
			$S_{CA}(E_{X_i}(D_{common}(\cdot), D^b_{common}(\cdot), s_{common}))$

Figure 4.3: Registration protocol

The subscriber X_i sends his identity X_i along with his public key to the service provider SP. The service provider returns a signature with his public key of a fresh timestamp *timestamp* and the identity encrypted under X_i's public key. The subscriber then contacts the certificate authority CA transmitting the above signature (and its message contents) plus its public key $E_{X_i}(\cdot)$. The CA will now distribute a certificate to X_i. In order to do that the identity of X_i (and implicitly the authenticity of the message) is usually verified via second channel, e.g. e-mail. If this check has been successful, the CA sends the certificate (including identity X_i and public key $E_{X_i}(\cdot)$) and the signed and encrypted common keys $D_{common}(\cdot), D^b_{common}(\cdot)$ and s_{common} back to X_i.

The distribution protocol of the CA follows its regular distribution protocol of the generated certificate after verification of the identity of the subject (subscriber). Its only addition is that the CA also sends $D_{common}(\cdot)$, $D^b_{common}(\cdot)$ and s_{common} to the subscriber X_i. This message part can be encrypted with $E_{X_i}(\cdot)$ which has just been verified or alternatively a session key generated from it.

Before the CA issues a certificate for the use in the benchmarking protocol and reveals the associated secrets it should verify that the subscriber is indeed in the process of registration with the service provider. The subscriber presents a proof that the service provider agrees to the participation of the subscriber, i.e. she has paid the necessary subscription fee. The service provider, after completing the necessary verification and credit checks, issues

the signed token to the subscriber X_i. The token is the signed concatenation of the timestamp and the name of the subscriber X_i: $S_{SP}(timestamp, X_i)$. At this point of time, the service provider cannot rely on authenticated channels for the verification, but may perform the verification independently, e.g. based on credit card data. The subscriber X_i forwards the token to the CA when applying for a certificate. The CA verifies the subscriber X_i's identity and the freshness of the token and upon successful verification continues with its regular verification protocol and amended distribution from above. A summary of the interactions without details of the verifications by the individual parties is depicted in Figure 4.3.

4.2 Benchmarking Protocol

The benchmarking protocol is run to compute the statistics of the KPIs of the subscribers in the peer group, i.e. a protocol run is made for each KPI and each peer group. The protocol fulfills the functional requirements from Chapter 3 and the evaluation of non-functional requirements is described in Chapter 7.

4.2.1 Model

As stated before a benchmarking protocol is run for each peer group j and each KPI k. The benchmarking protocol is run between $q_j + 1$ parties: the service provider SP and the subscribers $X_i (i \in Q_j)$. The only allowed communication links are between each X_i and SP, i.e. no communication between any X_i and $X_{i'}$ $(i' \in Q_j)$ is allowed. Subscribers must remain strongly anonymous amongst each other, i.e. each message received by a subscriber should be free of identifiers of other subscribers.

The service provider SP initiates each protocol and determines the peer group j and the KPI k for which statistics are to be computed. The peer group formation, i.e. the fact that $i \in Q_j$, is discussed in Section 6.2. Also the size of the peer group q_j is determined by peer group formation. All this information is sent by the service provider SP to the subscribers X_i before the benchmarking protocol.

In the remainder of the exposition the index k is dropped for clarity, i.e. the KPI of X_i is x_i. Also the vector of all inputs is denoted as $\vec{x}$ ($\vec{\gamma}_{j,k}$ in Section 3.2) and its sorted version as $\overrightarrow{\overline{x}}$. Each subscriber X_i provides his KPIs x_i as input to the protocol. The service provider SP provides no input. Both the service provider SP and each subscriber X_i obtain the output. The outputs are the mean of $\vec{x}$ times q_j, the variance of $\vec{x}$ times q_j, the maximum of $\overrightarrow{\overline{x}}$, the median of $\overrightarrow{\overline{x}}$ and the best-in-class of $\overrightarrow{\overline{x}}$ times $\lceil \frac{q_j}{4} \rceil$. All statistics are computed according to the formulas in Section 3.2.

The security model of the computation is semi-honest. For a definition of semi-honest security see Section 4.3.2. It is later extended to constrained

malicious security which is described in Section 4.3.3.

4.2.2 Notation

This section reviews the notation used throughout the dissertation and, in particular, in the formal descriptions of the protocols. The benchmarking protocol is specified formally in Figure 4.4. The notation for cryptographic operations is similar to the symbolic notation of Dolev, Yao [25], although [25] does not provide notations for all necessary operations.

An indexed sequence or vector of values is written as $\vec{x} = (x_1, \ldots, x_{|\vec{x}|})$ where x_i is the i-th element of the vector. Matrices are written as $\mathbf{M}, \mathbf{N}, \ldots$ and individual elements as $m_{i,j}, n_{i,j}, \ldots$. Unordered sets are written as $\mathbb{S}, \mathbb{T}, \ldots$. The length of vectors and the size of sets is written as $|\vec{x}|$ and $|\mathbb{S}|$, respectively. Unless otherwise noted values are indexed as seen from a system-wide perspective throughout the dissertation, i.e. there are n subscribers, p peer groups, and o KPIs. Some protocol variables, e.g. the validation bit $b_{i,j,k}$, exist independently for each dimension.

The concatenation of strings a and b is written as $a|b$. No special notation is used when a numeric variable or symbol is referenced as a string. Instead standard conversion techniques are assumed.

Table 4.1 gives an overview of the variables names and their usage.

4.2.3 Protocol Description

The benchmarking protocol is a composition of several techniques that combined achieve the computation of the five statistics: average, variance, maximum, median, best-in-class. These techniques are summation, rank computation, selection, best-in-class and decryption. This section explains each technique separately in order to decrease the level of complexity for the reader. The description is rather informal focusing on the protocol mechanisms while a formal description is given in Figure 4.4.

Nevertheless, references to the line numbers of Figure 4.4 are given, such that the two can be easily related. The description of the techniques is followed by two sections: one on computing with floating-point numbers and one on the protocol composition.

The final protocol is composed from the techniques described next. The way of composition and the resulting properties are described in one section.

All values referenced in the protocols are integers or bits, since homomorphic encryption can only deal with integers and bits. Nevertheless, the statistics need to be computed as floating-point numbers. How this is realized is described in Paragraph “Computing with Floating-Point Numbers”.

b	validation bit
c	condition variable for comparison
f	function
g	function
i	index
j	index into peer groups
k	index into KPIs
$\vec{m}$	messages in the protocol $\phi = \lvert\vec{m}\rvert$
n	number of subscribers: $n = \lvert\vec{X}\rvert$
o	number of KPIs
p	number of peer groups
$\vec{q}$	sizes of peer groups: $q_j = \lvert\mathbb{Q}_j\rvert$
r	uniformly chosen random number
s	secret key
t	size of the attacker set
x	KPI value
$name$	named variable, e.g. sum
A, B	attackers (as polynomial-time circuits)
C	polynomial-time circuit
$D(\cdot)$	decryption function
$\mathbb{D}$	domain of hidden values
$E(\cdot)$	encryption function
E^{name}	named encrypted value
$H(\cdot)$	cryptographic hash function
I	set of attacker (indices)
$MAC(\cdot, s)$	message authentication code function
P	any party in the protocol
$\vec{\mathbb{Q}}$	peer group members (subsets of $\{1, \ldots, n\}$)
SP	service provider
$\vec{X}$	subscribers
$\vec{\beta}_k$	k-th column of $\mathbf{x}$, i.e. $k-th$ KPI of all subscribers
$\vec{\gamma}_{j,k}$	$k-th$ KPI of all members of the $j-th$ peer group
ϕ	number of messages
$\mathbf{\Lambda}$	peer group participation: $\lambda_{i,j} \in 0, 1$
Φ	(random) permutation: $\Phi(i)$ is the permuted element for i

Table 4.1: Variables

Round 1:					
1	X_i	$\longrightarrow$	SP	$E_{common}(x_i)$	
2				$E^b_{common}(x_i)$	
Round 2:					
3	SP	$\longrightarrow$	X_i	$E_{common}(sum + r_1) = E_{common}(\sum_{i=1}^{q_j} x_i) \cdot E_{common}(r_1)$	
4				$E^b_{common}(c_{\vec{\Phi(i)}}) = (\ldots,$	
				$E^b_{common}(c_{\Phi(i)_{\Phi'(l)}}) = E^b_{common}(x_{\Phi(i)} < x_{\Phi'(l)}), \ldots)$	
5	SP	$\xrightarrow{\text{OT}}$	X_i	$E_i^{median} = \begin{cases} E_{common}(x_{\Phi(i)} + r_{4_i}) & \text{if } pos(\vec{c}_{\Phi(i)}) = \lceil \frac{q_j}{2} \rceil \\ E_{common}(r_{4_i}) & \text{otherwise} \end{cases}$	
6				$E_i^{bic} = \begin{cases} E_{common}(x_{\Phi(i)} + r_{5_i}) & \text{if } pos(\vec{c}_{\Phi(i)}) \geq \lceil \frac{3}{4} q_j \rceil \\ E_{common}(r_{5_i}) & \text{otherwise} \end{cases}$	
7				$E_i^{max} = \begin{cases} E_{common}(x_{\Phi(i)} + r_{6_i}) & \text{if } pos(\vec{c}_{\Phi(i)}) = q_j \\ E_{common}(r_{6_i}) & \text{otherwise} \end{cases}$	
8	X_i	$\longrightarrow$	SP	$sum + r_1$	
9				$MAC(sum + r_1 \vert i, s_{common})$	
10	SP	$\longrightarrow$	X_i	sum	
11	X_i	$\longrightarrow$	SP	$E_i^{median} \cdot E_{common}(0)$	
12				$E_i^{bic} \cdot E_{common}(0)$	
13				$E_i^{max} \cdot E_{common}(0)$	
14				$E_{common}((x_i - \frac{sum}{q_j})^2)$	
Round 3:					
15	SP	$\longrightarrow$	X_i	$E_{common}(sum' + r_7) = E_{common}(\sum_{i=1}^{q_j}(x_i - avg)^2) \cdot E_{common}(r_7)$	
16				$E_{common}(median + r_8) = (\prod_{i=1}^{q_j} E_i^{median} \cdot E_{common}(-r_{4_i})) \cdot$ $E_{common}(r_8)$	
17				$E_{common}(bic + r_9) = (\prod_{i=1}^{q_j} E_i^{bic} \cdot E_{common}(-r_{5_i})) \cdot E_{common}(r_9)$	
18				$E_{common}(max + r_{10}) = (\prod_{i=1}^{q_j} E_i^{max} \cdot E_{common}(-r_{6_i})) \cdot E_{common}(r_{10})$	
19				$H(MAC(sum + r_1 \vert 1, s_{common}), \ldots, MAC(sum + r_1 \vert q_j, s_{common}))$	
20	X_i	$\longrightarrow$	SP	$sum' + r_7$	
21				$MAC(sum' + r_7 \vert i, s_{common})$	
22				$median + r_8$	
23				$MAC(median + r_8 \vert i, s_{common})$	
24				$bic + r_9$	
25				$MAC(bic + r_9 \vert i, s_{common})$	
26				$max + r_{10}$	
27				$MAC(max + r_{10} \vert i, s_{common})$	
28	SP	$\longrightarrow$	X_i	sum'	
29				$median$	
30				bic	
31				max	
Round 4:					
32	SP	$\longrightarrow$	X_i	$H(MAC(sum' + r_7 \vert 1, s_{common}), \ldots, MAC(sum' + r_7 \vert q_j, s_{common}))$	
33				$H(MAC(median + r_8 \vert 1, s_{common}), \ldots, MAC(median + r_8 \vert q_j, s_{common}))$	
34				$H(MAC(bic + r_9 \vert 1, s_{common}), \ldots, MAC(bic + r_9 \vert q_j, s_{common}))$	
35				$H(MAC(max + r_{10} \vert 1, s_{common}), \ldots, MAC(max + r_{10} \vert q_j, s_{common}))$	

Figure 4.4: Benchmarking protocol

Summation

X_i	$\longrightarrow$	SP	$E_{common}(x_i)$
SP	$\longrightarrow$	X_i	$E_{common}(sum) = E_{common}(\sum_{l=1}^{q_j} x_l)$

Figure 4.5: Summation

Each subscriber X_i sends his KPI value x_i encrypted under the shared homomorphic encryption key, i.e. $E_{common}(x_i)$ for this protocol run to the service provider (SP). Recall that q_j is the number of subscribers in the j-th peer group. The service provider computes $\prod_{i=1}^{q_j} E_{common}(x_i) = E_{common}(\sum_{i=1}^{q_j} x_i)$. He now holds the encrypted sum $E_{common}(sum)$. Figure 4.5 shows the protocol steps for a summation.

The size q_j of the peer group is a result of peer group formation and distributed by the service provider SP. Summation is equivalent to average, since every participants then knows the number of subscribers q_j and can compute $avg = \frac{sum}{q_j}$ (and its inverse, such that there is no privacy violation).

In the protocol of Figure 4.4 this technique is used to compute the average (lines $1, 3$). The same technique is repeated for the variance sum' which is the average (sum) of $(x_{i,k} - avg)^2$ (lines $14, 15$). In the composed protocol the summation is already combined with the blinding for decryption.

Rank Computation

X_i	$\longrightarrow$	SP	$E^b_{common}(x_i)$
SP			$E^b_{common}(c_{\vec{\Phi(i)}}) = (\ldots,$
			$E^b_{common}(c_{\Phi(i)_{\Phi'(l)}}) = E^b_{common}(x_{\Phi(i)} < x_{\Phi'(l)}), \ldots)$

Figure 4.6: Rank computation

The rank $rank_i$ of a KPI x_i is its position in the descending-sorted list of all KPIs (from all subscribers). Recall, as in the last section, that q_j is the number of subscribers in the j-th peer group, i.e. the highest KPI has rank 1 and the lowest KPI has rank q_j. The rank of each element is computed as follows from the comparison of x_i and each other KPI $x_{i'}$ ($i' = 1, \ldots, q_j$): The service provider computes a matrix $\mathbf{C}$ with

$$E^b_{common}(c_{i,i'}) \quad = \quad E^b_{commom}(x_i < x_{i'})$$

where $c_{i,i'} = 1$, if $x_i \leq x_{i'}$ and $c_{i,i'} = 0$ otherwise.

The comparison bit $c_{i,i'}$ is computed following the protocol from [32]. The following description briefly reviews the protocol. For security proofs the reader is referred to [32].

Recall, that $E^b_{common}(x_i)$ denotes the bit-wise encryption of x_i in the homomorphic encryption system of [39]. All homomorphic operations are performed on bits, i.e. addition becomes exclusive-or and multiplication becomes logical and. Let $x_{i,h}$ denote the h-th bit of x_h. If $x_i > x_{i'}$, then there exists h, such that

$$c'_{i,i',h} = x_{i,h} \cdot (x_{i',h} + 1) \cdot \prod_{g=h+1}^{|x_i|} (x_{i,g} \cdot x_{i',g} \cdot 1) = 1$$

Note that there exists at most one such h, and therefore the service provider can send the (bit-wise encrypted) randomly permuted vector $\vec{c'_{i,i'}}$ to the subscriber X_i. The subscriber then computes $c_{i,i'} = \prod_{g=1}^{|x_i|}(c'_{i,i',g} + 1)$. Note that due to the random permutation the subscriber X_i will not learn the index h, even if it exists. Let $E^b_{common}(x_{\Phi(i)} < x_{\Phi'(l)})$ denote the (element-wise) encryption of the vector $\vec{c'_{i,i'}}$.

The computation of $c'_{i,i',h}$ involves one, multiple fan-in logical and operation on cipher texts. This can be achieved using the technique of [71]. It is directly linked to the homomorphic encryption scheme of [34]. Given two ciphertexts $E^b(a)$ and $E^b(b)$ one can compute their logical and as follows: Choose a parameter ϵ. For each g ($1 \leq g \leq \epsilon$) flip a random coin $r_g \in \{0, 1\}$. Compute $a'_g = E^b(a)^{r_g}$, i.e. $a'_g = E^b(a)$ if $r_g = 1$ and $a'_g = E^b(0)$ if $r_g = 0$. Note that $\forall g.a'_g = 0$ if $a = 0$ and that $\exists g.a'_g = 1$ if $a = 1$ (with high probability). Repeat the same procedure for $E^b(b)$ resulting in b'_g. Compute $ab_g = a'_g \cdot b'_g = E^b(a \cdot r_g + b \cdot r'_g)$. Note that $\forall g.ab_g = 0$ if $a \cdot b = 0$ and that $\exists g.ab_g = 1$ if $a \cdot b = 1$ (again with high probability). The decoding of $\vec{a}b$ corresponds to this observation, i.e. if there is at least one 1 decode to 1. Note that there is an error probability of incorrectly decoding a 1 as a 0. This probability is exactly $2^{-\epsilon}$. Cipher texts after applying the technique of [71] are still denoted as $E^b(ab)$, since the decryption procedure remains and only the decoding step has been added. The encoding can be inferred from the context.

Furthermore the service provider chooses two random permutations Φ and Φ' of the subscribers $[1, q_j]$. Note that Φ' is chosen fresh for each subscriber X_i in one run of the benchmarking protocol while Φ is chosen once for each run of the benchmarking protocol. Let $\Phi(i)$ and $\Phi'(i)$ denote the assigned element of i in the permutation Φ or $\Phi'(i)$, respectively.

The service provider SP sends a vector $\vec{c_{\Phi(i)}} = (c_{\Phi(i),\Phi'(1)}, \ldots, c_{\Phi(i),\Phi'(q_j)})$ to the subscriber X_i. Let $pos(\vec{c_{\Phi(i)}})$ denote the number of positive elements in $\vec{c_{\Phi(i)}}$. It holds that

$$rank_{\Phi(i)} = q_j - pos(\vec{c_{\Phi(i)}}) + 1$$

Each subscriber X_i now holds the rank of a KPI from a randomly chosen subscriber $X_{\Phi(i)}$.

The rank computation technique is used once within the benchmarking protocol (lines 2, 4). It builds the basis for median, maximum and best-in-class computation.

Median and maximum are elements with a specific rank of $\lceil \frac{q_j}{2} \rceil$ and 1, respectively. For best-in-class computation all elements with ranks lower than or equal to $\lceil \frac{q_j}{4} \rceil$ are summed up.

This rank computation protocol computes $O(q_j^2)$ comparisons while an efficient non-secure, sequential sorting algorithm requires only $O(q_j \log g_j)$ comparisons. A secure sorting algorithm could be based on sorting networks whose control flow is oblivious to the result of the comparisons and requires also $O(q_j \log g_j)$ comparisons ($O(q_j \log^2 g_j)$ in practice). The sorting performance can be significantly improved by exploiting the distributed nature of the computation. Each subscriber X_i interacts independently with the service provider SP in order to compute q_j comparisons. That way all comparisons can be computed in 1 round of the benchmarking protocol.

Unfortunately, not all multisets contain an element with the required rank, but all multisets do have a median, a maximum and best-in-class elements. The problem can arise when the multiset contains duplicate values, because then these values have the same rank and some ranks ($[1, q_j]$) remain unassigned. If each value is unique within the multiset, each rank will be assigned to an element and the median can be computed as element with rank $\lceil \frac{q_j}{2} \rceil$. Therefore the service provider appends an unique constant from a random permutation Φ_s to each bit-wise encrypted KPI value x_i. He can do so using the cipher texts $E^b_{common}(x_i)$ and the public key $E^b_{common}()$. As a result all ties between KPIs are broken. This does not influence the computation of the other statistics.

Selection

X_i	$\longrightarrow$	SP	$E_{common}(x_i)$
SP	$\xrightarrow{\text{OT}}$	X_i	$E_i = \begin{cases} E_{common}(x_{\Phi(i)} + r_i) & \text{if } rank_{\Phi(i)} \text{ is selected} \\ E_{common}(r_i) & \text{otherwise} \end{cases}$
X_i	$\longrightarrow$	SP	$E'_i = E_i \cdot E_{common}(0)$
SP			$E_{common}(x) = \prod_{i=1}^{q_j} E'_i \cdot E_{common}(-r_i)$

Figure 4.7: Selection

Selection is the process of computing the encrypted value (cipher text) of a KPI x_i with a selected rank (or the sum of KPIs with selected ranks) at the service provider SP's side. For each subscriber the service provider chooses

a random number r_i and prepares $\theta_0 = E_{common}(x_{\Phi(i)}) \cdot E_{common}(r_i) = E_{common}(x_{\Phi(i)} + r_i)$ and $\theta_1 = E_{common}(r_i)$. The subscriber X_i and the service provider engage in an 1-out-of-2 Oblivious Transfer protocol where X_i receives $E_i = \theta_0$, if her rank is the selected one, and $E_i = \theta_1$ otherwise, i.e. only the subscriber whose assigned KPI has been selected by its rank, receives the KPI's value (blinded by r_i). All other receive a blinded 0. Each subscriber rerandomizes the received value E_i, e.g. by multiplying it with a neutral element $E_{common}(0)$, and returns it to the service provider. Note that due to the rerandomization the service provider SP cannot decide which cipher text he received.

As in the last section, q_j is the number of subscribers in the j-th peer group. The service provider computes $\prod_{i=1}^{q_j}(E_i \cdot E_{common}(-r_i)) = E_{common}(x)$ where x is the element with the selected rank. Afterwards the service provider holds $E_{common}(x)$.

In the benchmarking protocol selection is repeated for median, maximum and all best-in-class elements and the service provider obtains encryptions of median ($median$), maximum (max) and best-in-class (bic). Note, that in the case of the best-in-class elements, multiple elements are selected at once, but due to the subsequent summation the computation is correct. The Oblivious Transfer protocols are executed first (lines $5, 6, 7$), then the selected values are returned (lines $11, 12, 13$) and finally the result is computed (lines $16, 17, 18$).

Decryption

SP	$\longrightarrow$	X_i	$E_{common}(v+r) = E_{common}(v) \cdot E_{common}(r)$
X_i	$\longrightarrow$	SP	$v + r$
			$MAC(v + r \mid i, s_{common})$
SP	$\longrightarrow$	X_i	v
			$H(MAC(v + r \mid 1, s_{common}), \ldots, MAC(v + r \mid q_j, s_{common}))$

Figure 4.8: Decryption

So far, the service provider only holds encrypted results and this technique helps him to have them decrypted by the subscribers. The service provider SP should be the first to learn the decrypted result in order to round it to insensitive values. He blinds the result by adding a random value r and sends the blinded cipher text $E_{common}(v + r)$ to each subscriber X_i. Let $|$ denote string concatenation, i.e. $a|b$ is the concatenation of a and b. X_i responds with its plain text $v + r$ and its message authentication $MAC(v + r|i, s_{common})$. The service provider delivers in the next round the clear result v and confirms with the (aggregated) hash of all concatenated message au-

thentication codes $H(MAC(v+r|1, s_{common}), \ldots, MAC(v+r|q_j, s_{common}))$. The subscriber X_i computes a validation bit $b_{i,j,k}$ as the comparison of the received aggregated hash and its locally computed value. This validation bit $b_{i,j,k}$ indicates whether the service provider SP submitted the same cipher text to all subscribers. In a later section it is proven that a match of the received and computed validation bit is only possible if all subscribers have received the same cipher text. A subscriber X_i can therefore verify using the validation bit, if the service provider adhered to this particular protocol step. This comparison prevents the service provider from mounting a specific attack by which he could gain additional knowledge about the KPIs.

In the benchmarking protocol the decryption is used five times. v is substituted with $sum, sum', median, max, bic$. The individual decryption operations are interleaved: the hidden and encrypted values are sent (lines 3, $15-18$), returned (lines 8, 9, $20-27$), plain-text delivered (lines 10, $28-31$) and their aggregates verified (lines 19, $32-35$).

Combined Protocol

The previous sections presented techniques used in the benchmarking protocol. The combination of all techniques in order to compute the statistics forms the benchmarking protocol. All protocols steps have been placed in the latest round possible to balance the communication cost between rounds and in the earliest round necessary in order to minimize the number of required rounds. The result is depicted in Figure 4.4.

The benchmarking protocol operates in rounds. This section gives a formal definition of round in the central communication pattern.

Definition 4.1 *A round is a step or sequence of steps in the protocol that needs to be completed by every subscriber before any subscriber can take the next step.*

Rounds are the main synchronization mode in the benchmarking protocol. A subscriber cannot initiate the steps of the next round before every other subscriber has completed this round. The order of subscribers in each round is arbitrary and can vary between any two rounds. Two subscribers can even engage in one round of the protocol with the service provider in parallel, i.e. the service provider can be multi-threaded.

The benchmarking protocol requires four rounds, i.e. a constant number of rounds independent of the number of participants or the input. Also the steps in each round are constant and do not depend on the number of participants or the input. Round complexity is one measure of communication complexity. The round complexity of the benchmarking protocol is constant ($O(1)$) and therefore asymptotically optimal.

Computing with Floating-Point Numbers

Homomorphic encryption operates on integers and all KPIs are submitted in encrypted form using homomorphic encryption. Fortunately all multiplications in the protocol on the encrypted values are with integers (or bits). Therefore floating point numbers can be represented by multiplying them with a large constant, e.g. 10^6. Then addition and multiplication with integers carry along this scaling factor. The results are divided by the constant before being presented to the subscriber. The solution offers less accuracy than regular floating-point numbers and operations, but due to the restricted domain of the input KPIs this is acceptable. Our protocols therefore allow operations on floating point numbers.

When computing using homomorphic encryption it is also important to prevent wrap-around (the modulus) of the computed results. The according modulus needs to be chosen with a large enough size. In the implementation using Paillier's encryption choosing a large enough domain is not a problem, since the domain of the plain text is almost equal to the key length. Key lengths in Paillier's encryption system are equal to key lengths in the popular RSA system [67], i.e. 512 to 2048 bits.

4.3 Security Proof

This section describes the security proof of the benchmarking protocol from Figure 4.4. It contains two proofs: first in the semi-honest security model as defined in [34] and second in the constrained malicious security model extended from Goldreich's malicious security also in [34].

First this section lists the security assumption in Section 4.3.1. Then Section 4.3.2 starts with the semi-honest model. It first reviews Goldreich's definitions in slightly simplified form adapted for the benchmarking protocol and its proof. It repeats the composition theorem from [34] used in the construction. In Section "Proof Outline" Goldreich's proof technique is described which is then filled with the details of the benchmarking protocol in the next sections. The security is proven by a comparison of the views (Section "Views") and the simulators (Section "Simulator") which are given in the next sections. The comparison is given in Section "Comparison".

Next Section 4.3.3 starts the definitions for security in the constrained malicious model. Section "Proof Outline" reviews the proof technique again. The protocol is decomposed into two sub-protocols which are analyzed separately (Sections "Message Computation" and "Decryption"). Section "Combination of the Sub-Protocols" completes the proof for composed protocol by combining the individual results.

4.3.1 Security Assumptions

Semantic Security

Semantic security means it must be unfeasible for a probabilistic polynomial-time adversary to derive information about a plain text when given its cipher text and the public key. A more formal definition can be found in [36]. The property of semantic security has been proven equivalent to cipher text indistinguishability. If an encryption scheme possesses the property of indistinguishability, then an adversary will be unable to distinguish pairs of cipher texts based on the message they encrypt. Paillier's encryption system has been proven semantically secure against chosen plain text attacks under the Decisional Composite Residuosity Assumption [63]. Goldwasser, Micali encryption has been proven secure under the intractability assumption of deciding Quadratic Residuosity modulo composite numbers whose factorization is unknown [39].

Secret Sharing

A (t, t) secret sharing scheme divides a secret s into a number t ($t \geq 2$) of shares. Given $t-1$ shares one cannot reconstruct s or even infer any information about s. Such a secret sharing scheme is called a perfect secret sharing scheme. An example is sharing with "addition modulo a constant c" where $t-1$ shares are random numbers $r_1, \ldots, r_{t-1} (0 \leq r_i < c)$ and the last share is $r_t = s - \sum_{i=1}^{t-1} r_i \bmod c$ [47, 75].

Cryptographic Hash Functions

A hash function is a function that is easy to compute and compresses an input x of finite arbitrary length to an output $H(x)$ of fixed finite length [65]. Cryptographic hash functions provide pre-image resistance, i.e. given a cryptographic hash $H(x)$ (with unknown x) it is computationally infeasible to find a value for x. In collision-resistant hash function it is computationally infeasible to find any two distinct inputs x and x', such that $H(x) = H(x')$.

A message authentication code (MAC) is a cryptographic hash function parameterized by a secret key k. More importantly, MACs provide computation-resistance, i.e. given any number of authenticated texts $\langle x_i, MAC(x_i, k)\rangle$ it is computationally infeasible to compute another authenticated text $\langle x, MAC(x, k)\rangle$ ($x \neq x_i$) without knowing the key k. A successful attempt of producing an authenticated text is called MAC forgery.

4.3.2 The Semi-Honest Model

The semi-honest model is based on comparing the real protocol execution or real model with an ideal model. In the ideal model all participants send their input to a trusted third party which responds with the correct answer.

For a protocol to be secure any attack in the real model must be feasible in the ideal model. For semi-honest security this means that what can be computed in the real model must not be more than what can be computed in the ideal model where only input and output are available.

An attacker in the semi-honest model follows the protocol as it is described, but keeps a record of all messages received and tries to infer as much information as possible from it. Such an attacker would in a real computer system monitor the communication interfaces and APIs or even the network wire to his computer. He would not modify the data content or use a self-written simulation program. Preventing attackers from modifying program variables and the program itself has been described in the literature [11, 42, 79]. Semi-honest security and these security mechanisms are therefore complementary.

Loosely speaking, the definition of semi-honest security states that what an attacker can learn from the protocol he can learn from his input and output. The view of an attacker defines what he can learn from a protocol.

Definition 4.2 *The view of the i-th party during an execution of the benchmarking protocol on* $(x_1, \ldots, x_n)$ *is denoted*

$$VIEW_i(x_1, \ldots, x_n) = \{x_i, r_i, m_1, \ldots, m_\phi\}$$

where r_i *represents the outcome of the i-th party's internal coin tosses, and* m_i *represents the i-th message it has received.*

The result and everything the attacker can compute after a protocol run are implicit in the view. For the definition of security it therefore suffices to show that the view itself can be computed (simulated) from the input and output. Security against a semi-honest attacker (for deterministic functions) is defined according to Goldreich as follows:

Definition 4.3 *Let*

$$f(x_1, \ldots, x_n) : (\{0,1\}^*)^n \mapsto (\{0,1\}^*)$$

be the benchmarking functionality. For

$$I = \{i_1, \ldots, i_t\} \subset \{1, \ldots, n\}$$

let

$$VIEW_I(x_1, \ldots, x_n) = \ (I, VIEW_{i_1}(\vec{x}), \ldots, VIEW_{i_t}(\vec{x}))$$

The protocol t-privately computes f *if there exists a polynomial-time simulator, denoted* S*, such that for every* I *of size* t*,* $S(I, (x_{i_1}, \ldots, x_{i_t}), f(\vec{x}))$ *is computationally indistinguishable from* $VIEW_I(x_1, \ldots, x_n)$*:*

$$S(I, (x_{i_1}, \ldots, x_{i_t}), f(\vec{x})) \stackrel{c}{=} VIEW_I(x_1, \ldots, x_n)$$

Composition Theorem

Often protocols are composed of sub-protocols whose security has been proven independently. The composition theorem from [34] described in this section helps proving the security of the composed protocol. Loosely speaking, the composition theorem states that given a proven sub-protocol, it is only necessary to prove the security of the composed protocol using the sub-protocol as a black box (oracle).

More formally, an oracle-aided protocol is one where all participants submit their input to an oracle which replies with the answer (or answers) according to a function f. An oracle-aided protocol (with oracle functionality f) is said to privately compute g, if there exists a polynomial-time algorithm (simulator) S satisfying definition 4.3. An oracle-aided protocol is said to privately reduce g to f, if it privately computes g while using oracle functionality f: g privately reduces to f.

Theorem 4.4 *If g privately reduces to f and there exists a protocol for privately computing f, then there exists a protocol for privately computing g.*

For a proof of this theorem, see [34]. In the benchmarking protocol the OT [29, 66] functionality will be replaced with an oracle for the purpose of the proof. Proofs for the specific OT implementation used in the implementation can be found in [59]. Our proof also abstracts from the cipher text details of the comparison in [32], although no messages are omitted in this case.

Proof Outline

This section states the security theorem to be proven and relates it to definition 4.3. Then it outlines the construction of the proof and its parts before they are described precisely for the benchmarking function in the next sections.

The following theorem will be proven

Theorem 4.5 *The benchmarking protocol 1-privately computes the functions average, maximum, variance, median, best-in-class and a validation bit in the semi-honest model.*

The validation bit ensures that the service provider has submitted the same cipher text to each subscriber and thereby prevents that the service provider has gained knowledge of KPIs by deviating from the protocol in this step. Its details are part of the next section. This section will prove that inclusion of the validation bit does not leak additional information in the semi-honest model.

The proof starts by constructing a simulator that is given the attacker's input (x_{i_1}) and output ($f(\vec{x})$). The simulator computes a sequence of messages $m_1, \ldots, m_\phi$. Then for each message m_i an argument based on a security assumption is given that this message m_i is computationally indistinguishable from the message received in the real model, i.e. the real execution of the protocol.

Before arguing for computational indistinguishability, consider the following example of an insecure protocol. If two probability distributions are computationally indistinguishable, then there exists no (probabilistic) polynomial-time distinguisher.

A distinguisher runs the protocol in the real model (i.e. it determines the input $\vec{x}$) and is handed either the view $VIEW_I(\vec{x})$ in the real model or the output of the simulator. The distinguisher is successful if it can determine with probability p non-negligibly larger than $p > \frac{1}{2} + \epsilon$, which one it received. The following example constructs a polynomial-time distinguisher for the two ensembles.

Assume the following insecure protocol where all parties send their inputs unencrypted, i.e. in plain text, to the service provider. Obviously the service provider can compute the correct result, but the following polynomial-time distinguisher for any simulator of the service provider exists. The distinguisher receives a view either from a real protocol run or from the simulator and outputs a decision which of these the view is from. The distinguisher proceeds as follows:

1. Choose and record random numbers r_i and use them as input $x_{i,k}$ for the real protocol run.

2. Receive message $m_1, \ldots, m_n$ (either from the protocol run or the simulator).

3. If all $m_i = r_i$, decide for real protocol execution, otherwise decide for the simulator.

The distinguisher succeeds, since the service provider has no input and the simulator has only the results as input. Therefore the simulator cannot do better than randomly guess the inputs and the distinguisher is successful in most cases. The central party can compute more about the subscribers' input in the real model than it can in the ideal model which is obvious, since it is sent the inputs of each party.

Security in the semi-honest model proves that no such distinguisher can exist.

Both types of participants are potential attackers: subscribers and the service provider. The benchmarking protocol cannot be privately computed in presence of a set of two or more attacking participants, since the set can be composed of a subscriber and the service provider. If they collude

and use the private key known to the subscriber to decrypt all incoming messages at the service provider, security has been violated. Theorem 4.5 therefore refers only to 1-private computation. This also implies that secure and authenticated channels are required for the execution of the protocol.

The next sections will show a simulator for each of the two possible views (again, subscriber and service provider). Two parts of the view – input and internal coin tosses – are not handled explicitly, because they can be simulated by copying the input and by using the same random number generator the implementation of the protocol uses in executing the real protocol. It remains to show that the received messages can be simulated computationally indistinguishably.

Therefore a simplified version of the view is presented which only shows the messages received. Afterwards a simulator for each view is given that has a corresponding entry for each message received. Then for each entry an argument based on the security assumptions is given as to why the output of the simulator is computationally indistinguishable from the message entry.

Statistical closeness implies computational indistinguishability [35]. Identically distributed functions are also statistically close. Therefore, if the probability distribution of a message is known, a function generating this probability distribution (i.e. an identically distributed function) can be used as a simulator. An example are a random share in the t, t perfectly secure "modular addition" secret sharing scheme and a uniformly chosen random value. Both are uniformly distributed in $[0, c[$.

Computational indistinguishability can also be based on already existing proven computational indistinguishability results, e.g. cipher text indistinguishability implies that two cipher texts are computationally indistinguishable, even if the distribution of the plain texts is different and known to the distinguisher.

Views

Recall from definition 4.2 that the view of a party is its input, its internal coin tosses and the messages it receives. Both input and coin tosses are chosen by the party itself and can be simulated as described in Section "Proof Outline". Therefore the focus is on the messages a party receives. This section restates from Figure 4.4 the messages a subscriber and the service provider receive during a run of the benchmarking protocol. This helps to illustrate the construction of the simulator which produces a computationally indistinguishable message for each message in the view from just the input and output of that party.

Subscribers' Messages Subscriber X_i receives the following messages during the benchmarking protocol. For simplicity the message have been numbered according to Figure 4.4. Recall that the subscriber has possession

of the common decryption keys D_{common} and D^b_{common} and can consequently decrypt the corresponding messages as indicated by $\rightarrow$ below. Also due to the composition theorem, OT has been replaced with its ideal functionality where the subscriber receives one of two items according to this choice bit. The messages exchanged during the implementation of the OT protocol ([59]) are left out.

1. Round 1:
2. Round 2:

   (3) $E_{common}(sum + r_1) \rightarrow sum + r_1$

   (4) $E^b_{common}(c_{\vec{\Phi(i)}}) \rightarrow (\ldots, c_{\Phi(i)_{\Phi'(l)}}, \ldots)$

   (5) $E_i^{median} \rightarrow median' + r_{4_i}$

   (6) $E_i^{bic} \rightarrow bic' + r_{5_i}$

   (7) $E_i^{max} \rightarrow max' + r_{6_i}$

   (10) sum

3. Round 3:

   (15) $E_{common}(sum' + r_7) \rightarrow sum' + r_7$

   (16) $E_{common}(median + r_8) \rightarrow median + r_8$

   (17) $E_{common}(bic + r_9) \rightarrow bic + r_9$

   (18) $E_{common}(max + r_{10}) \rightarrow max + r_{10}$

   (19) $H(MAC(sum + r_1|1, s_{common}), \ldots, MAC(sum + r_1|q_j, s_{common}))$

   (28) sum'

   (29) $median$

   (30) bic

   (31) max

4. Round 4:

   (32) $H(MAC(sum'+r_7|1, s_{common}), \ldots, MAC(sum'+r_7|q_j, s_{common}))$

   (33) $H(MAC(median+r_8|1, s_{common}), \ldots, MAC(median+r_8|q_j, s_{common}))$

   (34) $H(MAC(bic + r_9|1, s_{common}), \ldots, MAC(bic + r_9|q_j, s_{common}))$

   (35) $H(MAC(max+r_{10}|1, s_{common}), \ldots, MAC(max+r_{10}|q_j, s_{common}))$

Service Provider's Messages The service provider does not have access to the common decryption key and therefore cannot decrypt messages encrypted with $E_{common}(\cdot)$ or $E^b_{common}(\cdot)$. He also acts as the sender in OT sub-protocols, so all messages regarding OT are left out. The messages the service provider receives during the benchmarking protocol are then as follows:

1. Round 1:

   (1) $E_{common}(x_i)$
   (2) $E^b_{common}(x_i)$

2. Round 2:

   (8) $sum + r_1$
   (9) $MAC(sum + r_1|i, s_{common})$
   (11) $E_i^{median} \cdot E_{common}(0)$
   (12) $E_i^{bic} \cdot E_{bic}(0)$
   (13) $E_i^{max} \cdot E_{common}(0)$
   (14) $E_{common}((x_i - \frac{sum}{q_j})^2)$

3. Round 3:

   (20) $sum' + r_7$
   (21) $MAC(sum' + r_7|i, s_{common})$
   (22) $median + r_8$
   (23) $MAC(median + r_8|i, s_{common})$
   (24) $bic + r_9$
   (25) $MAC(bic + r_9|i, s_{common})$
   (26) $max + r_{10}$
   (27) $MAC(max + r_{10}|i, s_{common})$

4. Round 4:

Simulators

Given the messages as the main part of the view, the simulators can be constructed starting with the simulator for the subscriber's view.

Subscribers' Simulator

Lemma 4.6 *The view* $VIEW_{X_i}(x_1, \ldots, x_n)$ *of subscriber* X_i *during the benchmarking protocol can be simulated computationally indistinguishably by a simulator* $S_{X_i}(x_i, sum, sum', max, median, bic)$.

Lemma 4.6 is proven in the following by specifying simulator $S_{X_i}(x_i, sum, sum', max, median, bic)$.

The input of a subscriber is his input KPI value x_i. The outputs of a subscriber are the result values sum, sum', max, $median$, bic and the validation bit $b_{i,j,k}$ which indicates that the service provider has not gained knowledge of KPIs by deviating from the protocol. The simulation of input and internal coin tosses is left out and performed as described in Section "Proof Outline". The decrypted messages are simulated instead of the encrypted received ones. The subscriber (and the distinguisher) can decrypt with the common keys $D_{common}(\cdot)$ and $D^b_{common}(\cdot)$ and furthermore, if the decrypted messages can be simulated computationally indistinguishably, then the encrypted ones can be as well, since encryption is a deterministic mapping of probability distributions. The domain of the homomorphic operation in the homomorphic encryption scheme $E_{common}(\cdot)$ is denoted by the domain operator on $D_{common}(\cdot)$: $\mathrm{dom}(D_{common}(\cdot))$. Random variables that are reused during the simulator are denoted as $r', r'', \ldots$, while random variables that are used only during one step are denoted as r. The simulator $S_{X_i}(x_i, sum, sum', max, median, bic)$ for the subscribers' view is:

1. Round 1:
2. Round 2:
   - (3) a uniformly chosen random value r' $(0 \leq r' < \mathrm{dom}(D_{common}(\cdot)))$
   - (4) q_j uniformly chosen random bits r $(0 \leq r \leq 1)$
   - (5) a uniformly chosen random value r $(0 \leq r < \mathrm{dom}(D_{common}(\cdot)))$
   - (6) a uniformly chosen random value r $(0 \leq r < \mathrm{dom}(D_{common}(\cdot)))$
   - (7) a uniformly chosen random value r $(0 \leq r < \mathrm{dom}(D_{common}(\cdot)))$
   - (10) sum
3. Round 3:
   - (15) a uniformly chosen random value r'' $(0 \leq r'' < \mathrm{dom}(D_{common}(\cdot)))$
   - (16) a uniformly chosen random value r''' $(0 \leq r''' < \mathrm{dom}(D_{common}(\cdot)))$
   - (17) a uniformly chosen random value r'''' $(0 \leq r'''' < \mathrm{dom}(D_{common}(\cdot)))$
   - (18) a uniformly chosen random value r''''' $(0 \leq r''''' < \mathrm{dom}(D_{common}(\cdot)))$

(19) if $b_{i,j,k}$ is true, then the validation hash $H(MAC(r'|1, s_{common}), \ldots, MAC(r'|q_j, s_{common}))$, else a random number r $(0 \leq r < \mathrm{dom}(H(\cdot)))$

(28) sum'

(29) $median$

(30) bic

(31) max

4. Round 4:

(32) if $b_{i,j,k}$ is true, then the validation hash $H(MAC(r''|1, s_{common}), \ldots, MAC(r''|q_j, s_{common}))$, else a random number r $(0 \leq r < \mathrm{dom}(H(\cdot)))$

(33) if $b_{i,j,k}$ is true, then the validation hash $H(MAC(r'''|1, s_{common}), \ldots, MAC(r'''|q_j, s_{common}))$, else a random number r $(0 \leq r < \mathrm{dom}(H(\cdot)))$

(34) if $b_{i,j,k}$ is true, then the validation hash $H(MAC(r''''|1, s_{common}), \ldots, MAC(r''''|q_j, s_{common}))$, else a random number r $(0 \leq r < \mathrm{dom}(H(\cdot)))$

(35) if $b_{i,j,k}$ is true, then the validation hash $H(MAC(r'''''|1, s_{common}), \ldots, MAC(r'''''|q_j, s_{common}))$, else a random number r $(0 \leq r < \mathrm{dom}(H(\cdot)))$

Service Provider's Simulator

Lemma 4.7 *The view $VIEW_{SP}(x_1, \ldots, x_n)$ of the service provider SP during the benchmarking protocol can be simulated computationally indistinguishably by a simulator $S_{SP}(sum, sum', max, median, bic)$.*

Lemma 4.7 is proven in the following by specifying simulator $S_{SP}(sum, sum', max, median, bic)$.

The service provider has no input and his outputs are the result values $sum, sum', max, median$ and bic (no validation bit). The simulation of input and internal coin tosses is left out again. The service provider does not have access to the common decryption keys $D_{common}(\cdot)$ and $D^b_{common}(\cdot)$ and all encrypted message he receives cannot be decrypted by him. The domains of the homomorphic encryption schemes are denoted by the domain operator on $E_{common}(\cdot)$ and $E^b_{common}(\cdot)$: $\mathrm{dom}(E_{common}(\cdot))$ and $\mathrm{dom}(E^b_{common}(\cdot))$, respectively. The domain of the message authentication code is denoted by $\mathrm{dom}(MAC(\cdot))$. The simulator $S_{SP}(sum, sum', max, median, bic)$ for the service provider's view is:

1. Round 1:

(1) a uniformly chosen random value r $(0 \leq r < \mathrm{dom}(E_{common}(\cdot)))$

(1) a uniformly chosen random value r $(0 \leq r < \mathrm{dom}(E^b_{common}(\cdot)))$

2. Round 2:

(8) $sum + r_1$ (where r_1 is an internal coin toss)

(9) a uniformly chosen random value r $(0 \leq r < \mathrm{dom}(MAC(\cdot)))$

(11) a uniformly chosen random value r $(0 \leq r < \mathrm{dom}(E_{common}(\cdot)))$

(12) a uniformly chosen random value r $(0 \leq r < \mathrm{dom}(E_{common}(\cdot)))$

(13) a uniformly chosen random value r $(0 \leq r < \mathrm{dom}(E_{common}(\cdot)))$

(14) a uniformly chosen random value r $(0 \leq r < \mathrm{dom}(E_{common}(\cdot)))$

3. Round 3:

(20) $sum' + r_7$ (where r_7 is an internal coin toss)

(21) a uniformly chosen random value r $(0 \leq r < \mathrm{dom}(MAC(\cdot)))$

(22) $median + r_8$ (where r_8 is an internal coin toss)

(23) a uniformly chosen random value r $(0 \leq r < \mathrm{dom}(MAC(\cdot)))$

(24) $bic + r_9$ (where r_9 is an internal coin toss)

(25) a uniformly chosen random value r $(0 \leq r < \mathrm{dom}(MAC(\cdot)))$

(26) $max + r_{10}$ (where r_{10} is an internal coin toss)

(27) a uniformly chosen random value r $(0 \leq r < \mathrm{dom}(MAC(\cdot)))$

4. Round 4:

Comparison

This section shows why the simulators produce a computationally indistinguishable output. In the case of the subscribers' simulator, output values (steps 10, 28 to 31) can be simulated by simply copying the actual outputs of the protocol and are therefore identically distributed. The argument is shown at the example of step 10 of the protocol.

1. (10) $SP \longrightarrow X_i$ sum

2. (10) sum

3. (10) sum

The first line is taken from Figure 4.4 and corresponds to the message sent to subscriber X_i. The second line is taken from the view of X_i from Section "Subscribers' Messages" and the third is the output of the simulator from Section "Subscribers' Simulator". Clearly the second and third line are identical and the second line is derived from the first line. The arguments for steps 28 to 31 of the benchmarking protocol are identical.

In step 4 of the subscriber's view the comparison bits are simulated using q_j random bits.

1. (4) $SP \longrightarrow X_i\ E^b_{common}(c_{\vec{\Phi(i)}}) = (\ldots, E^b_{common}(c_{\Phi(i)_{\Phi'(l)}}) = E^b_{common}(x_{\Phi(i)} < x_{\Phi'(l)}), \ldots)$
2. (4) $E^b_{common}(c_{\vec{\Phi(i)}}) \rightarrow (\ldots, c_{\Phi(i)_{\Phi'(l)}}, \ldots)$
3. (4) q_j uniformly chosen random bits r $(0 \leq r \leq 1)$

The three lines are formed from Figure 4.4 and Sections "Subscribers' Messages" and "Subscribers' Simulator" again. The second line includes the decryption operation in the view of the subscriber X_i who is in possession of the key $D^b_{common}(\cdot)$ denoted by $\rightarrow$ and the first line shows the message sent. The number of positive bits is uniformly distributed between 1 and q_j (inclusive) due to random permutation Φ of the KPIs (subscribers), i.e. an input with a random rank is assigned to the participant X_i. Furthermore the order of the comparison bits is randomly chosen, such that as a result each bit $c_{\Phi(i)_{\Phi'(l)}}$ is uniformly randomly chosen.

Values hidden by adding a random number (modulo a constant) are secret shares and therefore identically distributed to and computationally indistinguishable from uniform random numbers (steps 3, 5 to 7 and 15 to 18). The compressed overview for example step 3 is given in the following:

1. (3) $SP \longrightarrow X_i\ E_{common}(sum + r_1) = E_{common}(\sum_{i=1}^{q_j} x_i) \cdot E_{common}(r_1)$
2. (3) $E_{common}(sum + r_1) \rightarrow sum + r_1$
3. (3) a uniformly chosen random value r' $(0 \leq r' < \mathrm{dom}(D_{common}(\cdot)))$

The assumption for indistinguishability is given in Section "Secret Sharing" and the decryption operation in the view $(\rightarrow)$ is equal to the other steps, e.g. step 4.

Finally the validation hashes in steps 19, and 32 to 35 appear as random numbers if they do not match the correct validation hash, since the subscriber cannot compute a pre-image and are therefore identically distributed to (and computationally indistinguishable from) a cryptographic hash of a random number. If they match the correct validation hash, the verification bit is set to true in the protocol and in the simulator they are

set to the correct validation hash keeping the identical distribution. The following gives a compressed overview of the three aspects message, view and simulator output for the example step 19.

1. (19) $SP \longrightarrow X_i\ H(MAC(sum + r_1|1, s_{common}), \ldots, MAC(sum + r_1|q_j, s_{common}))$
2. (19) $H(MAC(sum + r_1|1, s_{common}), \ldots, MAC(sum + r_1|q_j, s_{common}))$
3. (19) if $b_{i,j,k}$ is true, then the validation hash $H(MAC(r'|1, s_{common}), \ldots, MAC(r'|q_j, s_{common}))$, else a random number r $(0 \leq r < \mathrm{dom}(H(\cdot)))$

In the case of the service provider's simulator the output values are identically distributed and computationally indistinguishable, since they are copied from the output of the protocol in the ideal model (steps 8, 20, 22, 24 and 26). They are blinded by internal coin tosses by the service provider which are obviously known to him and chosen by the simulator. The compressed overview for step 8 is therefore very similar to the compressed view of step 10 of the subscribers' view from above.

1. (8) $X_i \longrightarrow SP\ sum + r_1$
2. (8) $sum + r_1$
3. (8) $sum + r_1$ (where r_1 is an internal coin toss)

The encrypted values are computationally indistinguishable from random numbers in steps 1, 2, and 11 to 14 which is a result of the proof of semantic security of the homomorphic encryption schemes.

1. (1) $X_i \longrightarrow SP\ E_{common}(x_i)$
2. (1) $E_{common}(x_i)$
3. (1) a uniformly chosen random value r $(0 \leq r < \mathrm{dom}(E_{common}(\cdot)))$

The security assumption for indistinguishability of the second and third line has been stated in Section "Semantic Security". The service provider cannot decrypt received values in his view, since he is not in possession of the common keys $D_{common}(\cdot)$ or $D^b_{common}(\cdot)$.

The following shows a compressed overview for step 9 of the benchmarking protocol.

1. (9) $X_i \longrightarrow SP\ MAC(sum + r_1|i, s_{common})$
2. (9) $MAC(sum + r_1|i, s_{common})$

3. (9) a uniformly chosen random value r $(0 \leq r < \mathrm{dom}(MAC(\cdot)))$

Since the service provider does not know the secret key for the MAC function, MACs are computationally indistinguishable from random numbers or otherwise the service provider would have a non-negligible advantage in MAC forgery. This argument applies to steps 9, 21, 23, 25, and 27.

This completes the proof of semi-honest security. Simulators for both views – subscriber and service provider – have been given and it has been shown that these simulators produce computationally indistinguishable output. Therefore it has been shown that the protocols (1-)privately compute the benchmarking function in the semi-honest security model according to definition 4.3.

4.3.3 Constrained Malicious Model

The constrained malicious model is stricter than the semi-honest model, but less strict than the malicious model as defined by Goldreich [34]. It allows a party to behave in arbitrary ways just as the malicious model does, but only under the constraint that the correct result is delivered to the other party. Its motivation is economical, since the service provider is assumed to sell his service, he is assumed to deliver the correct result to build a sustainable business. The service provider (and the subscribers) therefore have a vested interest in obtaining (and delivering) the correct result. The motivation to participate in the protocol is required even in the malicious model, since a party can abort the protocol at its discretion or submit incorrect input in order to mount a denial-of-service attack. In particular, a malicious subscriber can submit the maximum possible KPI value and thereby falsify the result of the maximum computation. Differently from an auction, where the maximum value or at least its submitter (Vickrey auctions [78]) are revealed, this is not case for benchmarking. Such attacks are not excluded by security in the malicious model, but the effect of correctly and willingly participating in the protocol is not taken into account in the malicious model. In the constrained malicious model this effect materializes as the delivery of the correct result.

The malicious model construction [34] reduces secrecy to secrecy in the semi-honest model, which in consequence also prevents someone from gaining additional knowledge by deviating from the protocol. This is also prevented in the constrained malicious model where one can deviate arbitrarily, but is not allowed to gain additional knowledge.

Security in the constrained malicious model is also defined by comparing the real protocol execution to an ideal model. The difference is that the attackers allowed in the constrained malicious model are more powerful than those allowed in the semi-honest model. The ideal model again consists of a third party that receives the inputs from all parties and delivers the output

to all parties. Since, we assume that the correct result is delivered to all subscribers, any attacker (as a service provider) in the constrained malicious model is admissible. Early aborting of the protocol by a service provider is ruled out. In case of the benchmarking protocol the service provider provides no input, but even if he would (e.g. in case of other centralized protocols), he could not substitute it, since it is assumed that the correct result is delivered including his potential input. Definition 4.8 defines an attacker in the ideal (constrained malicious) model. It is identical to the definition of an ideal attacker in the semi-honest model [34].

Definition 4.8 *Let*

$$f(x_1, \ldots, x_n) : (\{0,1\}^*)^n \mapsto (\{0,1\}^*)$$

be the benchmarking functionality and let

$$I = \{i_1, \ldots, i_t\} \subset \{1, \ldots, n\}, \neg I = \{1, \ldots, n\} \setminus I$$

Let

$$\vec{x}_I = (x_1, \ldots, x_n)_I = (x_{i_1}, \ldots, x_{i_t})$$

denote the subsequence of $\vec{x}$ under I. A pair (I, C), where C is a polynomial-size circuit family, represents an adversary A in the ideal (constrained malicious) model. The joint execution of the benchmarking protocols under C (and the other protocol participants $\neg C$ adhering to the protocol specification), denoted as $IDEAL_{C(\vec{x}_I),\neg C(\vec{x}_{\neg I})}$, is defined as:

$$IDEAL_{C(\vec{x}_I),\neg C(\vec{x}_{\neg I})} \stackrel{def}{=} (C(\vec{x}_I, f(\vec{x})), f(\vec{x}))$$

The means to protect against a constrained malicious service provider is the verification bit $b_{i,j,k}$. This section will clarify its purpose and prove that it protects against a constrained malicious service provider.

Before describing the countermeasure this paragraph describes a feasible attack by a malicious service provider that gains additional knowledge by deviating from the protocol without being detected. This attacker is allowed in the constrained malicious model, but not the semi-honest model. Imagine the following service provider: In the second round he submits to each subscriber X_i an encryption of his KPI value $x_{i,k}$ ($E_{common}(x_{i,k})$) instead of the encryption of the sum $E_{common}(sum + r_1)$ to all subscribers. According to the protocol description subscriber X_i will respond with $x_{i,k}$. If the service provider gathers all KPIs $\vec{x}_k$ he can compute the benchmarking function locally and deliver the correct results to all subscribers in the next steps when sending the clear plain texts *sum*, *sum′*, *median*, *max* and *bic*. Nevertheless, he has learnt all KPI values and thus gained additional knowledge he should not be able to. Preventing this form of attack is the goal of the constrained malicious model.

Formally, an attacker in the constrained malicious model is defined as:

Definition 4.9 *Let*

$$f(x_1, \ldots, x_n) : (\{0,1\}^*)^n \mapsto (\{0,1\}^*)$$

be the benchmarking functionality and let

$$I = \{i_1, \ldots, i_t\} \subset \{1, \ldots, n\}, \neg I = \{1, \ldots, n\} \setminus I$$

Let

$$\vec{x}_I = (x_1, \ldots, x_n)_I = (x_{i_1}, \ldots, x_{i_t})$$

denote the subsequence of $\vec{x}$ under I. A pair (I, C), where C is a polynomial-size circuit family, represents an adversary A in the real model. The joint execution of our protocols under C and $\neg C$ where $\neg C$ coincides with the strategies defined by our protocols, denoted as $REAL_{C(\vec{x}_I), \neg C(\vec{x}_{\neg I})}$, is defined as the output resulting from the interaction between $C(\vec{x}_I)$ and $\neg C(\vec{x}_{\neg I})$. Let $REAL_C$ be the output of I. An adversary A is admissible (for the constrained malicious model) if each party $\neg I$ outputs $f(\vec{x})$.

$$REAL_{C(\vec{x}_I), \neg C(\vec{x}_{\neg I})} = (REAL_C, f(\vec{x}))$$

Security in the constrained malicious model means that an attacker can only do what he can do in the ideal model. If that is the case, then there is a (polynomial-time) transformation of an attacker in the real (constrained malicious) model into an attacker in the ideal model. The formal definition of security in the constrained malicious model is

Definition 4.10 *Let*

$$f(x_1, \ldots, x_n) : (\{0,1\}^*)^n \mapsto (\{0,1\}^*)$$

be the benchmarking functionality and let

$$I = \{i_1, \ldots, i_t\} \subset \{1, \ldots, n\}, \neg I = \{1, \ldots, n\} \setminus I$$

Let

$$\vec{x}_I = (x_1, \ldots, x_n)_I = (x_{i_1}, \ldots, x_{i_t})$$

denote the subsequence of $\vec{x}$ under I. A protocol is said to t-securely and privately compute f in the constrained malicious model, if there exists a polynomial-time computable transformation of admissible attackers A for the real model into admissible attackers B for the ideal model, such that for every I of size t the output in the ideal model $IDEAL_{B(\vec{x}_I), \neg B(\vec{x}_{\neg I})}$ is computationally indistinguishable from the output in the real model $REAL_{A(\vec{x}_I), \neg A(\vec{x}_{\neg I})}$:

$$IDEAL_{B(\vec{x}_I), \neg B(\vec{x}_{\neg I})} \stackrel{c}{=} REAL_{A(\vec{x}_I), \neg A(\vec{x}_{\neg I})}$$

Proof Outline

This section states the theorem to be proven for security in the constrained malicious model and relates it to definition 4.10. Then it outlines the construction of the proof and its parts before they are stated in the next sections.

The claim of the proof is the following theorem

Theorem 4.11 *The benchmarking protocol 1-securely and privately computes the functions average, maximum, variance, median and best-in-class in the presence of a constrained malicious service provider as in definition 4.10.*

Recall that | denotes string concatenation. The validation bit

$$b_{i,j,k} : msg_{SP} \stackrel{?}{=} H(MAC(result|1, s_{common})| \ldots |MAC(result|q_j, s_{common}))$$

provides this guarantee, if it is $true$, i.e. the service provider has sent msg_{SP} equal to the (computed) hash for each result value $result \in \{sum, sum', median, max, bic\}$. A true validation bit $b_{i,j,k}$ guarantees that the service provider has submitted the same message to all subscribers and that is sufficient for the proof of theorem 4.11.

For the proof the benchmarking protocol is divided into three sections or sub-protocols. First, all steps of the benchmarking protocol are performed except the cipher texts sent to the subscribers for decryption. The main insight is that all of these messages received are computationally indistinguishable from independently chosen random numbers. This implies that the view of an attacker can be simulated no matter how he behaves (if abortion is excluded). Second, the cipher texts are decrypted by the subscribers and returned to the service provider. The ideal functionality for this decryption sub-protocol corresponds to a decryption oracle, that decrypts exactly one value each for the service provider. It is proven that the above validation bit $b_{i,j,k}$ ensures that the same value is decrypted (or it will be $false$) by all subscribers, such that the real protocol corresponds to the ideal functionality. Third, the correct result is delivered to the subscribers. The proof argues that if the correct benchmarking results are delivered, then they must have been decrypted in the decryption sub-protocols, since the other messages can be simulated by random numbers.

The validation bit approach still allows the service provider to gain additional knowledge by deviating from the protocol, but not without being detected. Once more this corresponds to the economic motivation, where the service provider wants to deliver the benchmarking service and only cheats to gain additional knowledge. If he cheats, the validation bit $b_{i,j,k}$ will show this and details can be investigated from the logs of the execution.

Message Computation

Lemma 4.12 *Let $VIEW^{\star}_{SP}(x_1, \ldots, x_n)$ be the view of the service provider SP in the benchmarking protocol without decryption (protocol steps 1, 2, 9,*

11-14, 21, 23, 25 and 27 in Figure 4.4). No polynomial-time attacker A (without abortion) can implement party SP in benchmarking protocol without decryption, such that the resulting view $VIEW^{\star}_{A}(x_1, \ldots, x_n)$ *is computationally distinguishable from* $VIEW^{\star}_{SP}(x_1, \ldots, x_n)$.

Corollary 4.13 *In the benchmarking protocol without decryption there exists a polynomial-time computable transformation of admissible attackers A for the real model into admissible attackers B for the ideal model.*

Lemma 4.12 follows from the construction of the view of party SP (see Section "Service Provider's Simulator"). The benchmarking protocol without decryption encompasses all steps of the benchmarking protocol in Figure 4.4 except decryption, as detailed in the next section. Excluding the plain texts each message can be simulated using an independently chosen random number. Therefore the view $VIEW^{\star}_{A}(x_1, \ldots, x_n)$ is independent of A and computationally indistinguishable from $VIEW^{\star}_{SP}(x_1, \ldots, x_n)$.

A (polynomial-time) transformation from an attacker A in the real model as the service provider to an attacker B in the ideal model emulates the execution of A on the output available in the ideal model (which is computationally indistinguishable from the messages in the real model) and uses that as the output of the attacker B. Let $S_{VIEW_{SP}}$ be simulator for the view $\overline{VIEW}_{SP}$ of SP in the ideal model. The transformed attacker B in the ideal model executes the protocol in the ideal model and runs the circuit A on the output of the simulator $S_{VIEW_{SP}}$ and outputs that as the output of the transformation. This output must be computationally indistinguishable, since the circuit A is a deterministic mapping of probability distributions and the inputs are computationally indistinguishable. If the outputs were computationally distinguishable, A would be the distinguisher for the inputs which contradicts the computationally indistinguishability assumption of the inputs.

Furthermore since the attacker in the ideal model has no input (as the service provider SP) it holds that the output $REAL_A(\vec{x})$ of the attacker A alone can be simulated by an attacker B without input in the ideal model.

$$REAL_A(\vec{x}) \stackrel{c}{=} IDEAL_B()$$

Decryption

Lemma 4.14 *The decryption sub-protocol implements the decryption oracle functionality. For every attacker A that can implement SP in the decryption sub-protocol (without abortion) there is an attacker B in the ideal model, such that the output* $IDEAL_{B,\neg B}$ *in the ideal model is computationally indistinguishable from the output* $REAL_{A,\neg A}$ *in the real model.*

The decryption sub-protocol is used to decrypt the result held as cipher text by the service provider. It is invoked five times by the main protocol;

once for each result value. In the real protocol the service provider SP sends the encrypted and blinded result $E_{common}(result + r)$ to each subscriber X_i who replies with the decrypted result $result + r$ and an authentication code $MAC(result + r|i, s_{common})$. Finally, the service provider has to produce

$$msg_{SP} = H(MAC(result+r|1, s_{common})| \ldots |MAC(result+r|q_j, s_{common}))$$

Each subscriber X_i compares msg_{SP} to his computed hash and outputs the result as the verification bit $b_{i,j,k}$. In the ideal model the service provider has access to an oracle that outputs one decryption per invocation. Note that in the ideal protocol it does not matter whether the oracle learns the plain text or not, i.e. whether the result is sent blinded or in clear, while in the real protocol it ensures that the service provider learns the result first.

Therefore the proof needs to show that in the real protocol the service provider can obtain at most one decryption, if the validation bit $b_{i,j,k}$ is $true$. First note that the service provider can only produce the correct message msg_{SP}, if he has all authentication codes from all subscribers. If he can produce the correct authentication code with another pre-image, he has produced a hash collision which contradicts the security of the cryptographic hash function.

Secondly note that the service provider must have submitted the same encrypted result to all subscribers, if he obtained all authentication codes. If he has an authentication code, but submitted a different encrypted result, he has successfully forged an authentication code, which contradicts the security of the message authentication code.

In conclusion, the service provider has submitted the same encrypted and blinded result to all subscribers, if the validation bit $b_{i,j,k}$ is $true$. If he can submit only one encrypted result, he can obtain only one decryption per invocation of the real protocol. Therefore the real protocol produces the same output as the ideal protocol (if the validation bit $b_{i,j,k}$ is $true$).

The transformation of an attacker A as a service provider in the real protocol to an attacker B in the ideal model constructs an attacker A' in the ideal model based on the first messages m_1 sent by A to X_1. B takes the message m_1 of A' and submits it to the oracle. The output of the transformation to B is the output of the oracle.

Combination of the Sub-Protocols

Lemma 4.15 *If the number of decryption sub-protocols equals the number of benchmarking results and attacker A is admissible in the constrained malicious model, i.e. $REAL_{A(),\neg A(\vec{x}_{\neg SP})} = (REAL_A, f(\vec{x}))$, then for a composition of benchmarking protocol without decryption and decryption sub-protocols there is an attacker B in the ideal model for any attacker A in the real model that produces computationally distinguishable output $IDEAL_{B,\neg B}$ from $REAL_{A,\neg A}$.*

Since the constrained malicious model postulates that the correct benchmarking results are delivered to the subscribers, the proof of Lemma 4.15 can focus on the indistinguishability of the outputs of the attackers. Let P be the benchmarking protocol, $P^\star$ the benchmarking protocol without decryption and D_n n invocations of the decryption sub-protocol. Let $REAL_A^{Protocol}(\vec{x})$ be the output of an attacker A in the real model posing as service provider SP in protocol $Protocol \in \{P, P^\star, D_5\}$.

If the attacker is admissible in the constrained malicious model, then there must exist a circuit C that extracts the correct benchmarking results from its output and delivers them to the subscribers.

$$C(REAL_A^P(\vec{x})) = f(\vec{x})$$

The benchmarking protocol is composed of the benchmarking protocol without decryption and five decryption sub-protocols. Let A' be the part of the attacker in the benchmarking protocol without decryption and A'' be the attacker in the five decryption sub-protocols. A' is also used for the notation of a polynomial-time circuit that extracts the five cipher texts for the decryption sub-protocols from the output of the attacker A' in the benchmarking protocol without decryption.

$$C(REAL_{A'}^{P^\star}(\vec{x}), REAL_{A''}^{D_5}(A'(REAL_{A'}^{P^\star}(\vec{x})))) = f(\vec{x})$$

From Lemma 4.12 it follows that the output of the attacker in the benchmarking protocol without decryption can be simulated.

$$C(IDEAL_B^{P^\star}(), REAL_{A''}^{D_5}(A'(REAL_{A'}^{P^\star}(\vec{x})))) = f(\vec{x})$$

Since the ideal attacker in the benchmarking protocol without decryption does not need any input, there exists a circuit C' emulating its output.

$$C'(REAL_{A''}^{D_5}(A'(REAL_{A'}^{P^\star}(\vec{x})))) = f(\vec{x})$$

From Lemma 4.14 it follows that the real attacker in the decryption sub-protocols can be simulated.

$$C'(IDEAL_{B'}^{D_5}(A'(REAL_{A'}^{P^\star}(\vec{x})))) = f(\vec{x})$$

Since the domain of the benchmarking function ($\mathrm{dom}(f)$) equals the domain of the plain texts ($\mathrm{dom}(D_{common}(\cdot))$) and the number of decryption sub-protocols equals the number of benchmarking results, the circuit C' must be invertible.

$$IDEAL_{B'}^{D_5}(A'(REAL_{A'}^{P^\star}(\vec{x}))) = C'^{-1}(f(\vec{x}))$$

The ideal attacker in the benchmarking protocol has no input and only the output $f(\vec{x})$. Therefore there exists an attacker B'', such that its output (at

the attackers' site) equals the output of the circuit C'^{-1} on the benchmarking results.

$$IDEAL^{D_5}_{B'}(A'(REAL^{P^\star}_{A'}(\vec{x}))) = IDEAL^{P}_{B''}(\vec{x})$$

From Lemma 4.14 it follows that there is an attacker in the real model whose output is computationally indistinguishable from the attacker in the ideal model.

$$REAL^{D_5}_{A'''}(A'(REAL^{P^\star}_{A'}(\vec{x}))) \stackrel{c}{=} IDEAL^{P}_{B''}(\vec{x})$$

From the composition theorem it follows that

$$REAL^{P^\star}_{A'}(\vec{x}), REAL^{D_5}_{A'''}(A'(REAL^{P^\star}_{A'}(\vec{x}))) \stackrel{c}{=} IDEAL^{P^\star}_{B'''}(), IDEAL^{P}_{B''}(\vec{x})$$

Since the output of the ideal attacker in the benchmarking protocol without decryption can be simulated, it follows that there exist an attacker B'''', such that

$$REAL^{P}_{A''''}(\vec{x}) \stackrel{c}{=} IDEAL^{P}_{B''''}(\vec{x})$$

This completes the proof of Lemma 4.15, i.e. for any polynomial time circuit A'''' representing an attacker in the real model, there is an attacker B'''' in the ideal model, such that its output is computationally indistinguishable. Theorem 4.11 is a corollary of Lemma 4.15, since the constrained malicious model requires an admissible attacker and the number of benchmarking results equals the number of decryption sub-protocols.

4.4 Summary

This section described the benchmarking protocol in detail. The required centralized communication pattern is augmented by the key distribution where all subscribers have access to three common keys for decryption and signing (message authentication codes). The key distribution is performed via a trusted dealer in the form of an extended certificate authority.

The benchmarking protocol is described formally in Figure 4.4 and informally in the text. The protocol requires linear communication and computation cost per participant $O(q_j)$, i.e. the overall cost is $O(q_j^2)$. It furthermore requires only a constant number of rounds (4) and computes the statistics average, variance, maximum, median and best-in-class. The chapter continues with a proof of security following the security definitions from [34].

It begins with a proof of security in the semi-honest model. In the semi-honest model, the participants are expected to follow the protocol, but keep a record of all messages and try to infer as much information as possible from them. Protocols secure in the semi-honest model protect the confidentiality of the messages.

The section continues to proof the security in the constrained malicious model. In the constrained malicious model, the participants may behave arbitrarily, but under the constraint that they deliver the correct result

to the other parties, i.e. the other parties learn the correct result. The motivation is that if the service provider behaves economically rational, i.e. he delivers the correct result for a long lasting business relationship, he cannot obtain additional information even if he deviates maliciously from the protocol. The proof that the protocol is secure against a constrained malicious service provider is given in Section 4.3.3.

Security against a constrained malicious party is a stronger notion of security than semi-honest security. It is shown that the benchmarking protocol secure in the semi-honest setting is attackable in the constrained malicious model. The protocols have been augmented with the message authentication codes and their aggregation in order to thwart attacks in the constrained malicious model.

5 Comparison

Performance of secure computation remains critical. This dissertation shows how to build a practically feasible, privacy-preserving benchmarking platform. Practical performance is therefore a key component.

This chapter presents a very efficient, yet randomization-based method for privacy-preserving comparison. Our experiments show that its performance is significantly better than the most efficient cryptographic method. Comparison is important for privacy-preserving benchmarking, since it dominates its $O(|q_j^2|)$ communication and computation complexity.

In the next section the comparison method is introduced in the two-party setting solving Yao's millionaire's problem. This includes a performance evaluation in comparison to the method used for comparison in the benchmarking protocol in Chapter 4. The algorithms for choosing the randomization parameters follow next. Then the modifications to the benchmarking protocol are presented. The security of the randomization approach is assessed in a number of experiments and it is shown that the leakage is acceptable in practice.

5.1 Yao's Millionaires' Protocol

A	$\longrightarrow$	B	$E_A(a)$
		B	$r, r'(0 \leq r' < r), r'' \in \{0,1\}$
A	$\longleftarrow$	B	$E_A(c) = ((E_A(a) \cdot E_A(-b))^r \cdot E_A(r'))^{-1^{r''}} =$
			$E_A(-1^{r''} \cdot (r \cdot (a-b) + r'))$
A		B	$(c \geq 0) \oplus r'' \Longleftrightarrow a \geq b$

Figure 5.1: Yao's millionaires' protocol

Yao's millionaires' problem has been introduced in [80]. Imagine two millionaires wanting to compare their riches, but not wanting to exchange the exact amount. They can use a protocol for privately computing the "greater-than" (or a similar) function.

Alice has an integer value a and Bob has an integer value b and they

want to compute $a \geq b$. $E_A(\cdot)$ denotes the encryption in a homomorphic, public key encryption system, where the private key is known only to Alice. Bob chooses three random values r, r' $(0 \leq r' < r)$ and r'' $(r'' \in \{0,1\})$. These three values multiplicatively blind Alice from obtaining the difference or any information about a. Bob computes

$$\begin{aligned} E_A(c) &= ((E_A(a) \cdot E_A(-b))^r \cdot E_A(r'))^{-1^{r''}} \\ &= E_A(-1^{r''} \cdot (r \cdot (a-b) + r')) \end{aligned}$$

It holds that

$$a < b \Longleftrightarrow (c < 0 \wedge r'' = 0) \vee (c \geq 0 \wedge r'' = 1)$$

Using the exclusive-or operator $(\oplus)$ one can also write

$$a < b \Longleftrightarrow c < 0 \oplus r''$$

Since Bob sends $E_A(c)$ to Alice, she obtains her share of the result by decryption. In order to have Alice learn the result first, Bob can always choose $r'' = 0$.

The protocol presented in Figure 5.1 is faster than the fastest known privacy-preserving protocol from [32]. The implementation results presented in [49] shown in Figure 5.2 compare the performance of the randomized protocol to different parameters of [32]. The protocol from [32] has a slight error probability of obtaining the incorrect result resulting from the use of the technique in [71]. The parameter ϵ that control the error probability of $2^{-\epsilon}$ is set to $24, 40$ and 56. The results show that even for low error probabilities and input bit length, the randomized Yao's millionaires' protocol outperforms the protocol from [32]. Already for 32 bit the randomized protocol shown above is always faster. For the proposed error parameter of $\epsilon = 56$ [32], the randomized protocol is 1.46 times faster for 16 bits and 2.68 times faster for 32 bits.

Note that this comparison was performed on a single computer, such that communication cost is neglected. The randomized protocol from Figure 5.1 transmits only two cipher texts while the protocol from [32] transmits $(\epsilon+1)l$ cipher texts where l is the bit length. Therefore a further advantage of the randomized protocol can be expected when considering communication cost.

Contrary to the protocol of [32] the randomized protocol shown in Figure 5.1 has another feature: it allows split Yao's millionaires' computation. In split Yao's millionaires' protocol the result is not obtained by a single party, but shared between the two parties. In this case, Alice determines a bit by comparing the received value c to 0 and Bob has r''. The result is the exclusive-or of the two bits, i.e. a perfect secret sharing. Split Yao's millionaires' protocols have applications as building blocks in many other protocols, e.g. [2, 33, 49].

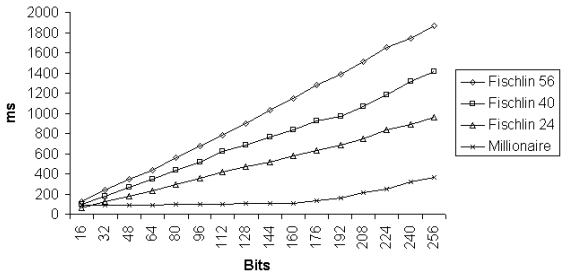

Figure 5.2: Yao's protocol performance

5.2 Choosing The Parameters

In order to hide the difference $a - b$ Bob chooses two random numbers r and r'. This section describes an algorithm to choose these numbers, such that they hide the difference effectively. First, one can abstract from the difference $a - b$ to a single random variable x. We assume that $x \neq 0$ which is true in the benchmarking protocol due to the KPIs made unique by the service provider.

Let $|x|$ denote the bit length of variable x. Note that, the bit length of the product rx is bounded by the bit lengths of the factors:

$$|r| + |x| - 1 \leq |rx| \leq |r| + |x|$$

Therefore choosing r uniformly in a range $[1, 2^k - 1]$ is counter-productive, since its expected bit length has very low variance (bit length k occurs with probability $\frac{1}{2}$). As a consequence the bit length of x is leaked with very high probability. This corresponds to a leakage of $\log |x|$ bits.

The idea is to choose the bit length of r independently first and then choose the remaining bits. This reduces the entropy of r, such that one needs to choose larger r, but it hides the bit length of x. If one chooses $|r|$ uniformly in $[1, k]$, then the bit length of x is almost optimally hidden, but very small values of r are very likely compared to large values of r. Very small or very large values of r reveal significantly more information about x than medium values, such that there is an increased probability of relatively large leakage.

In order to reduce the probability of small values of r occurring one can choose the bit length of r using a binomial distribution which can be approximated by a normal distribution with mean $\mu = \frac{k}{2}$ and standard variance $\sigma = \frac{k}{c}$ (where c is a constant). The probability of small values occurring can be significantly reduced choosing the appropriate parameter c. Of course, choosing c too small will increase the leakage due to the bit length again. These parameters are investigated in Section 5.4.

The remaining bits of r can be chosen uniformly. The second randomization parameter r' needs to be chosen, such that $r' < r$. In the experiments r' is chosen uniformly in $[0, 2^{|r|-1} - 1]$, i.e. such that it has at least one bit less than r, but this is only a necessary simplification in order to make the experiments computationally tractable. Choosing r' uniformly in $[0, r-1]$ will only increase entropy and reduce leakage.

5.3 Modified Benchmarking Protocol

In order to take advantage of the improved randomized comparison method the benchmarking protocol needs to be modified. In particular the protocol step of rank computation needs to be adapted. There is another, in practice quite important, advantage that the second homomorphic encryption system $E^b(\cdot)$ is no longer necessary. Instead the subscribers can send one cipher text each of their KPIs which can be used in the comparison and summation protocols. The next section presents the updated rank computation.

5.3.1 Rank Computation

X_i	$\longrightarrow$	SP	$E_{common}(x_i)$
SP			$E_{common}(c_{\vec{\Phi(i)}}) = (\ldots,$
			$E_{common}(c_{\Phi(i)_l}) = E_{common}(r_{1_l} \cdot (x_{\Phi(i)} - x_{\Phi(l)}) + r_{2_l}), \ldots)$

Figure 5.3: Rank computation

The rank $rank_i$ of a KPI x_i is its position in the descending-sorted list of all KPIs (from all subscribers). Recall that q_j is the number of subscribers in the j-th peer group, i.e. the highest KPI has rank 1 and the lowest KPI has rank q_j. The rank of each element is computed as follows from the difference of x_i and each other KPI $x_{i'}$ $(i' = 1, \ldots, q_j)$: The service provider computes a matrix $\mathbf{C}$ with

$$\begin{aligned} E_{common}(c_{i,i'}) &= (E_{common}(x_i) \cdot E_{common}(x_{i'})^{-1})^{r_{1_{i,i'}}} \cdot E_{common}(r_{2_{i,i'}}) \\ &= E_{common}(r_{1_{i,i'}} \cdot (x_i - x_{i'}) + r_{2_{i,i'}}) \end{aligned}$$

where $1 \leq r_{1_{i,i'}}$ and $0 \leq r_{2_{i,i'}} < r_{1_{i,i'}}$ are random numbers $(r_{1_{i,i'}}, r_{2_{i,i'}} \in \mathbb{N})$.

Furthermore he chooses two random permutation Φ and Φ' of the subscribers $[1, q_j]$. Let $\Phi(i)$ and $\Phi'(i)$ denote the assigned element of i in the permutation Φ and Φ', respectively.

The service provider sends a vector $c_{\vec{\Phi(i)}} = (c_{\Phi(i),\Phi'(1)}, \ldots, c_{\Phi(i),\Phi'(q_j)})$ to the subscriber X_i. Let $pos(c_{\vec{\Phi(i)}})$ denote the number of non-negative elements in $c_{\vec{\Phi(i)}}$. Since $c_{i,i'} \geq 0 \iff x_i \geq x_{i'}$, it holds that

$$rank_{\Phi(i)} = q_j - pos(c_{\vec{\Phi(i)}}) + 1$$

Round 1:				
1	X_i	$\longrightarrow$	SP	$E_{common}(x_i)$
Round 2:				
2	SP	$\longrightarrow$	X_i	$E_{common}(sum + r_1) = E_{common}(\sum_{i=1}^{q_j} x_i) \cdot E_{common}(r_1)$
3				$E_{common}(c_{\vec{\Phi(i)}}) = (\ldots,$
				$E_{common}(c_{\Phi(i)_{\Phi'(l)}}) = E_{common}(r_{2_l} \cdot (x_{\Phi(i)} - x_{\Phi'(l)}) + r_{3_l}), \ldots)$
4	SP	$\xrightarrow{\text{OT}}$	X_i	$E_i^{median} = \begin{cases} E_{common}(x_{\Phi(i)} + r_{4_i}) & \text{if } pos(\vec{c}_{\Phi(i)}) = \lceil \frac{q_j}{2} \rceil \\ E_{common}(r_{4_i}) & \text{otherwise} \end{cases}$
5				$E_i^{bic} = \begin{cases} E_{common}(x_{\Phi(i)} + r_{5_i}) & \text{if } pos(\vec{c}_{\Phi(i)}) \geq \lceil \frac{3}{4} q_j \rceil \\ E_{common}(r_{5_i}) & \text{otherwise} \end{cases}$
6				$E_i^{max} = \begin{cases} E_{common}(x_{\Phi(i)} + r_{6_i}) & \text{if } pos(\vec{c}_{\Phi(i)}) = q_j \\ E_{common}(r_{6_i}) & \text{otherwise} \end{cases}$
7	X_i	$\longrightarrow$	SP	$sum + r_1$
8				$MAC(sum + r_1 \vert i, s_{common})$
9	SP	$\longrightarrow$	X_i	sum
10				$E_i^{median} \cdot E_{common}(0)$
11				$E_i^{bic} \cdot E_{common}(0)$
12				$E_i^{max} \cdot E_{common}(0)$
13				$E_{common}((x_i - \frac{sum}{q_j})^2)$
Round 3:				
14	SP	$\longrightarrow$	X_i	$E_{common}(sum' + r_7) = E_{common}(\sum_{i=1}^{q_j}(x_i - avg)^2) \cdot E_{common}(r_7)$
15				$E_{common}(median + r_8) = (\prod_{i=1}^{q_j} E_i^{median} \cdot E_{common}(-r_{4_i})) \cdot E_{common}(r_8)$
16				$E_{common}(bic + r_9) = (\prod_{i=1}^{q_j} E_i^{bic} \cdot E_{common}(-r_{5_i})) \cdot E_{common}(r_9)$
17				$E_{common}(max + r_{10}) = (\prod_{i=1}^{q_j} E_i^{max} \cdot E_{common}(-r_{6_i})) \cdot E_{common}(r_{10})$
18				$H(MAC(sum + r_1 \vert 1, s_{common}), \ldots, MAC(sum + r_1 \vert q_j, s_{common}))$
19	X_i	$\longrightarrow$	SP	$sum' + r_7$
20				$MAC(sum' + r_7 \vert i, s_{common})$
21				$median + r_8$
22				$MAC(median + r_8 \vert i, s_{common})$
23				$bic + r_9$
24				$MAC(bic + r_9 \vert i, s_{common})$
25				$max + r_{10}$
26				$MAC(max + r_{10} \vert i, s_{common})$
27	SP	$\longrightarrow$	X_i	sum'
28				$median$
29				bic
30				max
Round 4:				
31	SP	$\longrightarrow$	X_i	$H(MAC(sum' + r_7 \vert 1, s_{common}), \ldots, MAC(sum' + r_7 \vert q_j, s_{common}))$
32				$H(MAC(median + r_8 \vert 1, s_{common}), \ldots, MAC(median + r_8 \vert q_j, s_{common}))$
33				$H(MAC(bic + r_9 \vert 1, s_{common}), \ldots, MAC(bic + r_9 \vert q_j, s_{common}))$
34				$H(MAC(max + r_{10} \vert 1, s_{common}), \ldots, MAC(max + r_{10} \vert q_j, s_{common}))$

Figure 5.4: Modified Benchmarking protocol

Each subscriber X_i now holds the rank of a KPI from a randomly chosen subscriber $X_{\Phi(i)}$.

The rank computation technique is used once within the modified benchmarking protocol (line 3). It builds the basis for median, maximum and best-in-class computation.

Median and maximum are elements with a specific rank of $\lceil \frac{q_j}{2} \rceil$ and 1, respectively. For best-in-class the elements with ranks greater or equal to $\lceil \frac{q_j}{4} \rceil$ are included in the computation.

In order to make the KPIs unique and break eventual ties, such that all ranks between 1 and q_j occur once, the service provider SP can still modify the cipher texts. This is only necessary for the comparison and does not affect other arithmetic computations. The service provider SP is given an encrypted KPI $E_{common}(x_i)$ and computes

$$E_{common}(y_i) = E_{common}(x_i)^{q_j} \cdot E_{common}(i) = E_{common}(x_i \cdot q_j + i)$$

The order of KPIs is preserved, i.e. $x_i < x_{i'} \implies y_i < y_{i'}$ and $x_i > x_{i'} \implies y_i > y_{i'}$. The values of y_i can be computed blindly by the service provider and used in the comparison just as the KPIs x_i.

5.3.2 Combined Protocol

The composed protocol is formally depicted in Figure 5.4.

5.4 Assessment

In this section the leakage of the randomized protocol is determined. Let R, R' and X denote the random variables with instances r, r' and x. The hiding is achieved as $Y = RX + R'$.

In the experiments the bit length of R is 512 bits and the bit length of X is 16 bits. The limitation to 16 is due to the computational feasibility and can be extended in practice. Nevertheless even 16 bits offer ample width for the comparison of real-world sized KPIs. Many KPIs fit into 16 bit or can be rounded to 16 significant bits and in the worst case one can compare the logarithm of the KPIs. Very precise results are unnecessary for benchmarking, since there always is a certain error in the measurements of the KPIs. Given these bit lengths one randomized hiding can be encoded in one cipher text with 768 bits plain text space.

The measure for leakage is the lost entropy. The entropy $H(X)$ of a (discrete) random variable X is an information theoretical concept measuring its uncertainty. It is defined as

$$H(X) = -\sum_i p(X = x_i) \log p(X = x_i)$$

Assuming a uniform distribution of X the entropy of the plain text is $H(X) = 16$. The leakage of the randomized hiding is $H(X) - H(X|Y)$ where $H(X|Y)$ is the conditional entropy of the plain text X given a randomized cipher text Y.

The lost entropy is a probabilistic quantity measuring the expected loss over a number of samples. It does not bound the maximum loss for a single sample, but even in cryptographic approaches the maximum loss is a full disclosure. In order to estimate the skew of the loss distribution when possible the probability of occurrence of a loss is given.

The conditional entropy is defined as

$$H(X|Y) = -\sum_j p(Y = y_j) \sum_i p(X = x_i|Y = y_j) \log p(X = x_i|Y = y_j)$$

The critical factor of the conditional entropy is $p(X = x|Y = y)$. Let $\alpha \sim \beta$ denote that α and β are proportional to each other, i.e. $c = \frac{\alpha}{\beta}$. Assuming a fixed y and uniform X it holds that

$$\begin{aligned} p(X = x|Y = y) &= \frac{p(Y = y|X = x)p(X = x)}{p(Y = y)} \\ &\sim p(Y = y|X = x) \end{aligned}$$

The conditional probability $p(Y = y|X = x)$ can be computed as

$$p(Y = y|X = x) = \sum_{\substack{y = rx + r' \\ 0 \leq r' \leq 2^{|r|-1} - 1 \\ 1 \leq r \leq 2^k - 1}} p(R = r)p(R' = r'|R = r)$$

Recall that r is chosen by first choosing its bit length from a normal distribution and then the remaining bits uniformly. Given a cipher text y and a plain text x one can enumerate all possible r and r' in order to compute $p(Y = y|X = x)$. With the bit lengths used in the experiments this enumeration quickly becomes intractable. The solution is to enumerate all bit lengths of r. The bit length is chosen from a normal distribution and therefore one can easily compute the probability that a certain bit length has been chosen. Let $N(k, \mu, \sigma)$ denote the probability that k is chosen from a normal distribution with mean μ and standard deviation σ.

First, the upper and lower bounds of r and r' are computed. Let $\overline{r}$ and $\overline{r}'$ denote the values corresponding to the upper bound of r and $\underline{r}$ and $\underline{r}'$ the values corresponding to the lower bound of r. They are computed as

$$\begin{aligned} \overline{r} &= \lfloor \frac{y}{x} \rfloor \\ \overline{r}' &= y - \overline{r}x \\ \underline{r} &= \lfloor \frac{y}{x+1} \rfloor + 1 \\ \underline{r}' &= y - \underline{r}x \end{aligned}$$

An adjustment of $\underline{r}$ and $\underline{r}'$ is necessary if $|\underline{r}| = |\underline{r}'|$, since it violates the domain $[0, 2^{|r|-1} - 1]$ of r'. Note that this only reduces the conditional entropy and increases the leakage as already noted above. Recall that this adjustment is only necessary in order to make the computation tractable.

Then the bit lengths k of numbers between $\underline{r}$ and $\overline{r}$ are enumerated. Let $\underline{r}_k = \max(2^{k-1}, \underline{r})$ be the minimum element of bit length k. Let $\overline{r}_k = \min(2^k - 1, \overline{r})$ denote the maximum element of bit length k. The conditional probability is then computed as the sum for all bit lengths k for the product of the number of elements $\phi_k = \overline{r}_k - \underline{r}_k + 1$ for bit length k, the probability $\chi_k = N(k, \mu, \sigma)$ of choosing bit length k, the probability $\psi_k = \frac{1}{2^{k-1}}$ of choosing an element r given bit length k and the probability $\omega_k = \frac{1}{2^{k-1}}$ of choosing an element r' given bit length k:

$$\begin{aligned} p(Y = y|X = x) &= \sum_{k=|\underline{r}|}^{|\overline{r}|} \phi_k \chi_k \psi_k \omega_k \\ &= \sum_{k=|\underline{r}|}^{|\overline{r}|} (\overline{r}_k - \underline{r}_k + 1) N(k, \mu, \sigma) \frac{1}{2^{k-1}} \frac{1}{2^{k-1}} \end{aligned}$$

In the experiments $\mu = 256$ and $\sigma = 32$ were chosen, but the possible bit lengths were cut off at 1 and 512. The next sections continue to describe the estimations of the leakage in the benchmarking protocol.

5.4.1 Single Sample

Computing the conditional entropy $H(X|Y)$ for all possible x and all possible y is intractable for the used bit lengths. Therefore $H(X|Y)$ is estimated using the following experiment:

1. Choose random y as follows:
   (a) Choose a random x uniform in $[1, 2^l]$ (where $l = 16$)
   (b) Choose a random r by
      - choosing a bit length k from a normal distribution with $\mu = 256$ and $\sigma = 32$ with cutoffs 1 and 512.
      - choosing a random s uniform in $[0, 2^{k-1} - 1]$
      - computing $r = 2^{k-1} + s$
   (c) Choose a random r' uniform in $[0, 2^{k-1} - 1]$
   (d) Compute $y = rx + r'$
2. For all $X = x$ compute $p(Y = y|X = x)$
3. Compute $p(X = x|Y = y) = \frac{p(Y=y|X=x)}{\sum_x p(Y=y|X=x)}$.

4. Compute $H(X|Y = y)$[1].

To compute $H(X|Y)$ this experiment is repeated $n = 100000$ times and the average over all $H(X|Y = y)$ is the estimation of $H(X|Y)$. Since the samples of Y are chosen according to the distribution when X is uniformly chosen, this approximation's expected value equals the unknown $p(Y = y)$. Another interesting parameter is the probability that a given leakage occurs. For example, if the average leakage is low and acceptable, but a high or very high leakage occurs with even a low probability, this might be unacceptable in practice.

Estimating the probability that a certain leakage occurs is very difficult. The assumption is that random blinding factors r leak more information the closer they are to the boundaries of R. The underlying intuition is that very small r combined with small x leak almost all information about x, since the bit length will be revealed again and many possible values of x are excluded. As well, very large r combined with large x leak much information about x due to the same reasons. The estimation for the occurrence of a leak is therefore estimated as the probability that r is not smaller than y which is the maximum r for a given y and $x = 1$ and that r is not larger than $2^k - 1 - y$ (where $k = 512$). This probability can be computed similarly to $p(Y = y|X = x)$ by enumeration of the bit lengths. The probability of occurrence of a leakage h is then computed as the maximum of all samples with leakage greater or equal to h.

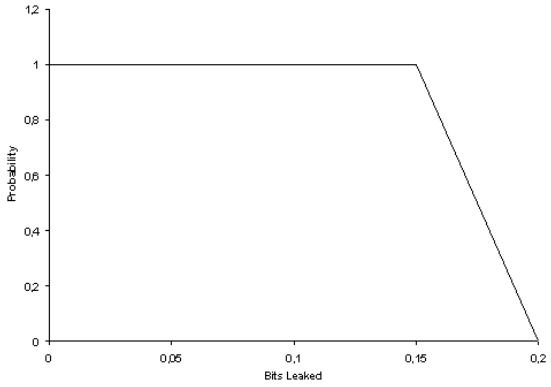

Figure 5.5: Probability of leakage of a single sample

The results of this experiment are depicted in Figure 5.5. The average leakage is 0.11 bits from a single sample. This leakage occurs almost always, as the estimated probability of occurrence is 1, while a leakage of 0.2 bits or

[1]For $Y = y$ this does not require $p(Y = y)$

above occurs with probability 0 in the experiment. This probability distribution of leakages would be acceptable in practice, but in the benchmarking protocol a subscriber is given q_j samples albeit from different plain texts. The resulting leakage is estimated in the next sections.

5.4.2 Multiple Samples

In the benchmarking protocol from Figure 5.4 a subscriber is sent q_j samples for comparison. A collusion of $q_j - 1$ subscribers can submit each KPI $x_i = 0$, such that the difference with the honest subscriber $X_{i'}$ is (almost) $x_{i'}$. Then it is likely that one attacker will receive q_j samples of one $X = x_{i'}$. This section estimates the leakage of n samples of Y for one $X = x$.

Clearly the expected leakage is upper bound by n times the leakage of one sample, i.e. $0.11n$ bits. Nevertheless it is not clear whether this leakage is also the expected average. In order to compute the probability of $X = x$ given multiple samples $y_1, \ldots, y_n$ the formula can be extended as follows

$$\begin{aligned} p(X = x | Y = y_1, Y = y_2) &\sim p(Y = y_1, Y = y_2 | X = x) \\ &= p(Y = y_1 | X = x) p(Y = y_2 | X = x) \end{aligned}$$

The probabilities $p(Y = y_1 | X = x)$ and $p(Y = y_2 | X = x)$ can be computed as before. The computation of the entropy follows.

The experiment can be modified as follows:

1. Choose a random x uniform in $[1, 2^l]$ (where $l = 16$)
2. Choose two random y_1 and y_2 as follows:
   (a) Choose a random r by
      - choosing a bit length k from a normal distribution with $\mu = 256$ and $\sigma = 32$ with cutoffs 1 and 512.
      - choosing a random s uniform in $[0, 2^{k-1} - 1]$
      - computing $r = 2^{k-1} + s$

   (b) Choose a random r' uniform in $[0, 2^{k-1} - 1]$
   (c) Compute $y_{\{1,2\}} = rx + r'$
3. For all $X = x$ compute $p(Y = y_1 | X = x)$ and $p(Y = y_2 | X = x)$
4. Compute $H(X | Y = y_1, Y = y_2)$

The experiment is performed for number of samples $n = 1, \ldots, 50$ and repeated 2000 times each. The results are depicted in Figure 5.6. The curve is flattening and appears to be logarithmic, such that with 45 samples (the maximum in the performance evaluation in Chapter 7) the leakage is only 2.39 bits; less than one half of the upper bound.

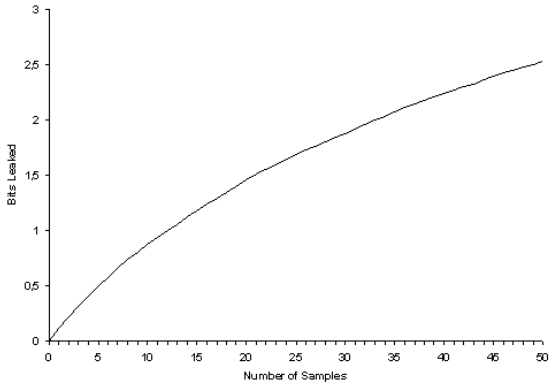

Figure 5.6: Leakage of multiple samples

Although the leakage still is low, the estimation is likely to be an overestimation, since it is unlikely that an attacker controls all other subscribers. This is particularly unlikely, since the subscribers remain anonymous and the benchmarking platform chooses the peer group participants. In the next section estimations are given for limited attackers – an assumption commonly made in secure computation.

5.4.3 Multiple Samples With Noise

As noted above in the benchmarking protocol from Figure 5.4 a subscriber is sent q_j samples for comparison, but in this section the assumption is that the attacker controls only a fraction of the subscribers. Let there be t samples of Y out of which s have a common $X = x$ which corresponds to the number of compromised subscribers. The other $t - s$ samples introduce noise in the attacker's correct estimation of x.

The attacker is given t samples and needs to pick s samples in order to infer x. Due to the random permutation by the service provider the attacker has no advantage in picking the subset where $X = x$, besides possible exclusions where no possible x remain. The resulting leakage is then estimated as the average of the leakages due multiple samples as in Section 5.4.2 of all possible subsets of size s. The new experiment is as follows

1. Choose a random x uniform in $[1, 2^l]$ (where $l = 16$)
2. Choose s random $y_{\{1,\ldots,s\}}$ as follows:
   (a) Choose a random r by
      - choosing a bit length k from a normal distribution with $\mu = 256$ and $\sigma = 32$ with cutoffs 1 and 512.

- choosing a random s uniform in $[0, 2^{k-1} - 1]$
- computing $r = 2^{k-1} + s$

(b) Choose a random r' uniform in $[0, 2^{k-1} - 1]$

(c) Compute $y_{\{1,\ldots,s\}} = rx + r'$

3. Choose $t - s$ random $y_{\{1,\ldots,t-s\}}$ as follows:

(a) Choose a random x' uniform in $[1, 2^l]$ (where $l = 16$)

(b) Choose a random r by

- choosing a bit length k from a normal distribution with $\mu = 256$ and $\sigma = 32$ with cutoffs 1 and 512.
- choosing a random s uniform in $[0, 2^{k-1} - 1]$
- computing $r = 2^{k-1} + s$

(c) Choose a random r' uniform in $[0, 2^{k-1} - 1]$

(d) Compute $y_{\{1,\ldots,t-s\}} = rx' + r'$

4. For all possible subsets $S_g = h_1, \ldots, h_s$ of $1, \ldots, t$

(a) For all $X = x$ compute $p(Y = y_{h_1}|X = x), \ldots, p(Y = y_{h_s}|X = x)$

(b) Compute $H_{S_g} = H(X|Y = y_{h_1}, \ldots, Y = y_{h_s})$

5. Let m be the number of subsets with a real-valued entropy H_{S_g}. Compute $H^\star(X|Y) = \frac{1}{m} \sum_{g=1}^{m} H_{S_g}$.

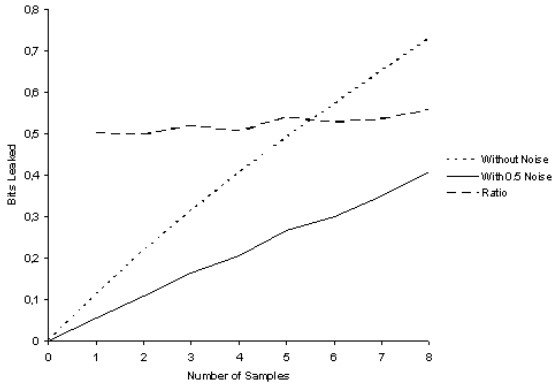

Figure 5.7: Leakage of multiple samples with noise

In the experiment s was chosen as a fixed ratio of t ($s = \frac{t}{2}$) and iterated over $s = 1, \ldots, 8$. The experiment was repeated $n = 200$ times and

averaged. Unfortunately the subset selection is exponential in the sample size, such that the number of repetitions needs to be lower than in previous experiments. The result is depicted in Figure 5.7. The line with noise has a significantly smaller slope. The ratio of samples with noise and without noise is also depicted and almost flat at 0.5. One can conclude that the leakage is restricted by the ratio of compromised subscribers.

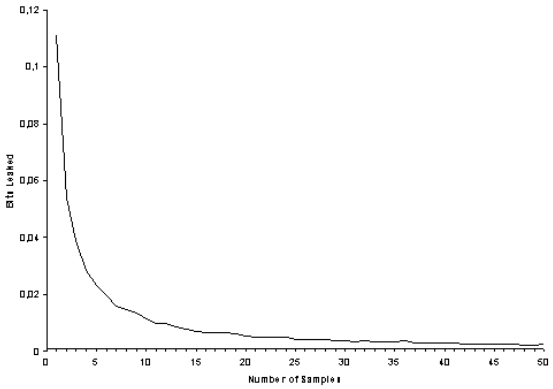

Figure 5.8: Leakage of single sample with noise

In most cases the attacker does not even control a fraction of the subscribers, but has compromised only a single subscriber. This corresponds to the assumption made in the security proof of the benchmarking protocol. The additional samples then only act as noise in identifying the value of a single KPI. Given the setup above the leakage with $s = 1$ and $t = 1, \ldots, 50$ is estimated. The result is depicted in Figure 5.8. The leakage with 6 subscribers (the minimum peer group size) is on average 0.02 bits. This leakage is very acceptable in practice.

5.5 Summary

In this chapter a randomized protocol to privately compare values was introduced. Its performance is a factor of 2.7 better than the best cryptographic method. It has been successfully included in the benchmarking protocol and resulted in another simplification by removing the need for the second homomorphic encryption scheme. Its leakage has been assessed in a number of well documented experiments. Given a single sample its leakage is approximately 0.11 bits and in the application in the benchmarking protocol its leakage depends on the power of the attacker. If the attacker has compromised all subscribers except one the leakage is on average 2.4 bits. If the attacker has compromised half of the subscribers the leakage is half of the expected average. In the most realistic case of an attacker acting on its own,

the expected leakage is limited to 0.02 bits. Note that an attacker does not know the identity of honest subscribers due to the anonymity property of the protocols and that peer groups are formed by the benchmarking platform. In summary, this section presented a practically secure and extraordinarily efficient protocol for privacy-preserving comparison. It enables the practical performance of the benchmarking protocol.

6 Software Architecture

In building a privacy-preserving benchmarking platform secure computation protocols are only one aspect. This chapters covers the architecture of the entire benchmarking system. It starts by describing an analysis of the use cases of the benchmarking platform in Section 6.1. Three use cases are identified and described in detail.

An important prerequisite for the benchmarking protocol is peer group formation. In peer group formation subscribers are assigned a peer group. An automatic, i.e. service provider performed, peer group formation is a novel feature for the benchmarking platform. A special modification to the k-means data clustering algorithm is presented and evaluated.

The chapter concludes with the comparison of the notification and the polling model for executing the benchmarking protocol. A simulation of the more complex polling model is done to evaluate its running time.

6.1 Use Case Analysis

Use case analysis is a commonly used technique to capture features in system development. Additionally it can be used as a guide for design and development. The use cases for the benchmarking platform have been developed from an end user's perspective covering all phases of a customer acquiring a benchmark report. The following three use cases comprise the benchmarking platform:

- Registration
- Statistics Retrieval
- Statistics Computation

6.1.1 Registration

Figure 6.1 shows the use case diagram for registration. Registration is the process of a customer becoming a subscriber. As described in Section 4.1.2 registration is a protocol between the customer, the service provider and a trusted third party. A certificate authority (CA) is envisioned as the trusted

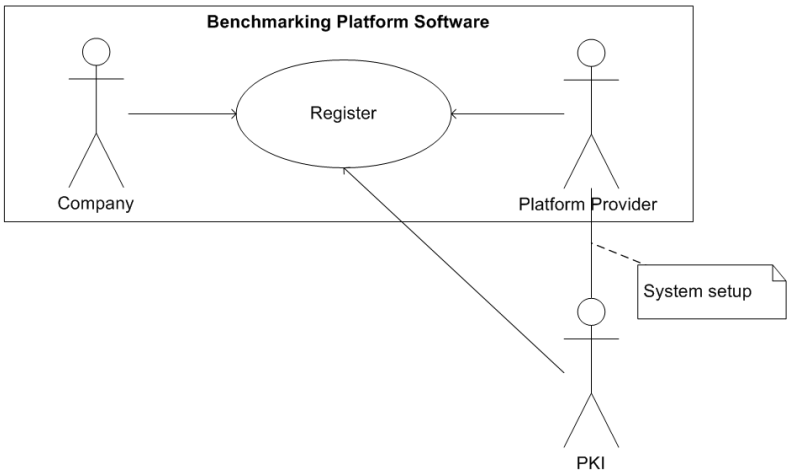

Figure 6.1: Use case: Registration

third party, since a certificate is issued to the subscriber. Registration occurs only once for each subscriber when she signs up for the benchmarking service. The pre-conditions for registration is a setup of the system parameters by the service provider and the CA. The details of the setup are described in Section 4.1.2. After registration the subscriber is able to engage in the other two use cases (statistics retrieval and computation).

6.1.2 Statistics Retrieval

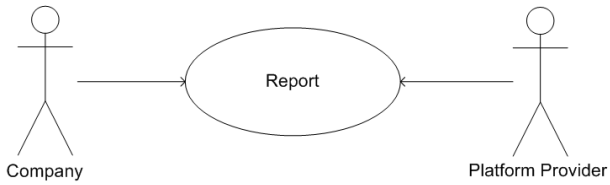

Figure 6.2: Use case: Statistics retrieval

Figure 6.2 shows the use case diagram for statistics retrieval. In statistics retrieval the subscriber obtains the statistics for her peer group. The service

provider has obtained the results in a benchmarking protocol and can store these for retrieval by the subscriber.

It is an important design decision to decouple the statistics retrieval from the statistics computation, i.e. the benchmarking protocol. Compared to other privacy-preserving systems [2, 30] this separation is a major deviation. However it provides clear advantages:

- A subscriber can retrieve her relevant statistics right after registration. There is no delay between service payment and delivery. This increases the benefit to the subscriber.
- Statistics do not need to be computed any time a demand arises. This allows subscribers to interact loosely with the service provider. Statistics computation can be done in the background.

In order to retrieve the statistics the subscriber needs to be assigned to a peer group. The process of identifying a peer group is not listed as a separate use case, but is a prerequisite for statistics retrieval (and computation). It is nevertheless important and in Section 6.2 an automatic algorithm is described.

Pre-condition for statistics retrieval is successful registration and peer group assignment. The peer group must have completed at least one statistics computation.

6.1.3 Statistics Computation

Figure 6.3 shows the use case diagram for statistics computation. Statistics computation runs the benchmarking protocol. It involves all members of the peer group and the service provider. Privacy preservation is the main constraint around which the protocol has been designed.

The statistics computation can be initiated by the service provider or a dedicated peer group member. The benchmarking protocol dictates that the service provider starts the protocol run and in case a peer group member initiates the protocol the service provider acts as an intermediary.

The peer group must have a minimum size for the statistics computation to be secure and each subscriber must have successfully registered.

6.1.4 Service Summary

The benchmarking platform must support the three use cases. Therefore the service provider has at least three services: a registration service offering a "subscribe" operation, a retrieval service offering an "assign" (peer group) and a "retrieve" (statistics) operation, and the services for implementing the benchmarking protocol. The subscriber does not need to offer services derived from the use cases. In Section 6.3 the orchestration of the benchmarking protocol is discussed in more detail. Depending on the model of

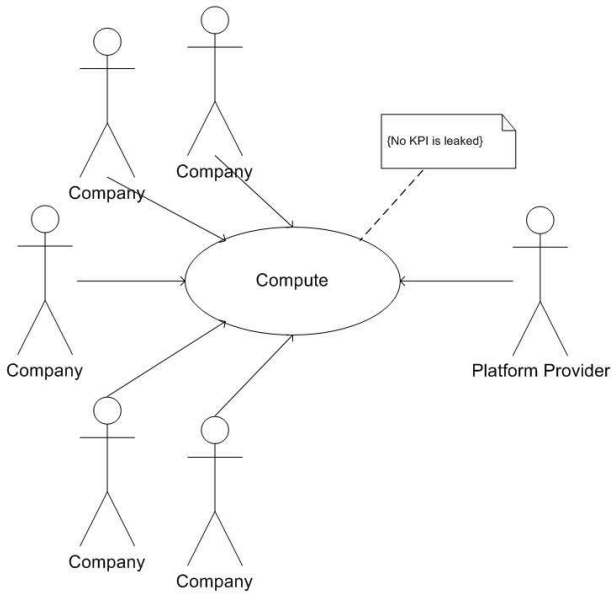

Figure 6.3: Use case: Statistics computation

protocol orchestration the subscriber may need to offer a service for protocol synchronization.

6.2 Peer Group Formation

Peer group formation is the process of computing the peer groups from the set of subscribers at a given point in time. Peer group formation creates a mapping between subscribers and peer groups. A peer group has a minimum size for its statistics to be meaningful in the benchmarking process. This minimum size is larger than one, since a subscriber wants to compare to its competition and not just itself. Therefore a peer group always maps to multiple subscribers.

6.2.1 Automatic Peer Group Formation

Automatic peer group formation is peer group formation performed by the service provider. It can be performed much like data clustering, i.e. grouping of related subscribers. For this every subscriber needs to map into a multi-dimensional data point. These data points are formed along several axis (criteria). The next section describes the criteria useful for subscriber classification.

Classification Criteria

Each company is classified by a number of criteria. These criteria refer to characteristics of the company, that are rather stable for the company but vary widely between companies. They should be relevant to the business model of the company, such that they group companies that have an interest in benchmarking against each other. Their combination should be almost identifying, i.e. unique. Examples include

- revenue
- profit
- net worth
- number of employees
- main company location (e.g. headquarters)
- main market region
- industry type
- industry sector
- legal form
- age

These characteristics are supposed to be public or at least the subscriber is willing to share them with the service provider. Each criterion is divided into a fixed number of discrete subclasses. These subclasses can be concrete instances, e.g. for industry type: resources, manufacturing, services, and administration, or intervals on a linear or logarithmic scale. Let m be the number of criteria, then each subscriber forms a data point in m-dimensional space.

K,L-Means Clustering

Automatic peer group formation on classified subscribers can be seen as a data clustering problem. A data clustering algorithm groups similar data points, such that the average distance to the cluster center is minimized.

A commonly used data clustering algorithm is the k-means algorithm [54]. It starts by randomly selecting k cluster centers. Each data point is assigned to the nearest cluster center and then the cluster center is recomputed as the center of its assigned data points. The algorithm iterates this process until the cluster centers stabilize, i.e. the distance between two iterations is below a threshold.

```
1  means[] := random datapoint[k]
2  do
3     size[] := 0[k]
4     flag[] := false[n]
5     for i := 1 to n
6        cluster[i] := index of closest means[]
7        size[cluster[i]] := size[cluster[i]] + 1
8     do
9        reassign := false
10       for i := 1 to k
11          if size[i] < l
12             min := index of closest datapoints[] with
13                cluster[min] != i
14                flag[min] == false)
15             size[i] := size[i] + 1
16             size[cluster[min]] := size[cluster[min]] - 1
17             flag[min] := true
18             cluster[min] := i
19             reassign := true
20    while reassign
21    for i := 1 to k
22       recompute means[]
23 until means[] stabilize
```

Table 6.1: K,l-means clustering algorithm

The k-means clustering algorithm [54] needs to be adapted to support a minimum cluster size. The minimum cluster size is necessary to protect the privacy of the individual subscribers and to create useful peer groups. Too-small peer groups (e.g. just two subscribers) reveal the parties' KPI values and are not particularly useful for benchmarking.

A solution called constrained k-means clustering using linear programming has been proposed in [6]. This solution does not scale to our requirements. The test case of 10^4 companies and 10^3 clusters leads to a linear programming model with 10^7 variables (and even more constraints). Such a large problem can only be solved using special hardware and the problem size increases by a factor of 10^2 using real-world requirements. Therefore a different algorithm is required.

A small extension to the cluster center assignment in the form of a greedy algorithm for cluster reassignment can fix the minimum size. Let l be the minimum cluster size. After the cluster center assignment, each cluster is processed again to ensure a minimum size l. If a cluster does not have l data points assigned to it, it is assigned the closest data points that

1. are not yet assigned to it
2. have not been reassigned in this iteration

This reassignment is continued until no more reassignments are necessary, i.e. all cluster have at least size l.

The condition that data points that have been reassigned in this iteration are not reassigned again, prevents two cluster centers competing for a certain data point, thus reassigning it alternatively and resulting in an infinite loop. Once the reassignment is complete, the algorithm proceeds as a regular k-means algorithm and recomputes the cluster centers. The final algorithm, called k,l-means clustering, is depicted in table 6.1 as pseudocode. The addition to the regular k-means algorithm is confined to the lines 8-20 and described in greater detail than the remaining algorithm.

6.2.2 Incremental Peer Group Formation

K,l-means clustering operates on all data points potentially assigning all subscribers new peer groups. This worst case would put a huge recomputation burden on the platform, since after a clustering algorithm run, all peer groups that had a change in membership need to be recomputed. Instead, it is advisable to limit the recomputation to a small set of affected peer groups. As well a customer wishes to retrieve a benchmarking result right after she becomes a subscriber and not wait for a clustering operation and recomputation to complete.

The solution is to compute the peer groups incrementally. When a new subscriber arrives, she is assigned to an existing peer group (and can therefore immediately retrieve statistics) by finding the closest cluster center of an existing peer group. When an upper limit on the peer group size has been reached, the peer group is split into two.

Subsequently the k,l-means clustering algorithm is used with $k = 2, l = \frac{\text{upper limit}}{2}$ on the previous peer group (including the new subscriber). At most these two resulting peer groups need to be recomputed when a new customer subscribes. The previous cluster mean is removed and the two new ones are added to the set of cluster means, incrementing the number of clusters by one.

Splitting a peer group and assigning each previous subscriber to one of the new ones contradicts the assignment invariant of the single peer group model. The subscriber can now be seen as belonging to two (different) peer groups with different membership. In order not to affect the privacy considerations of the platform, a time limit between recomputations of peer groups can be set. The assumption is that after that time period, KPIs have sufficiently changed to be statistically independent of previous values. The reverse calculation of KPIs from the statistics (see Section 3.5.2) does not hold in case of independent variables.

Evaluation

Without loss of generality assume 10 criteria each divided into 5 subclasses, i.e. n data points in a 10-dimensional space. Data points (customer) are

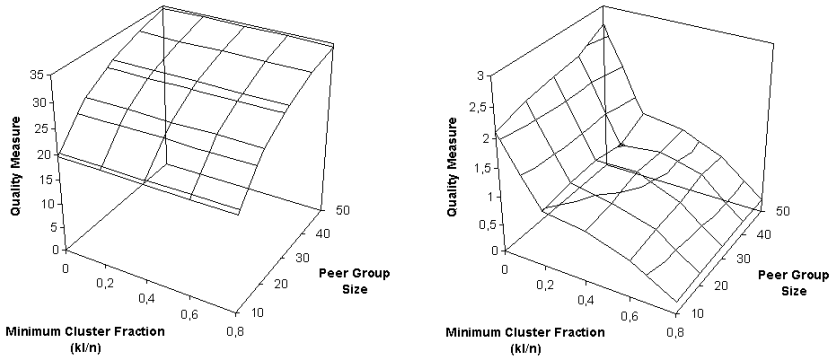

Figure 6.4: Quality for L_1 distance

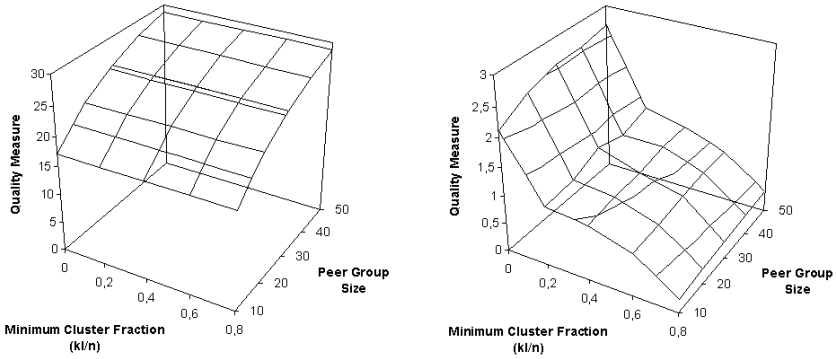

Figure 6.5: Quality for L_2 distance

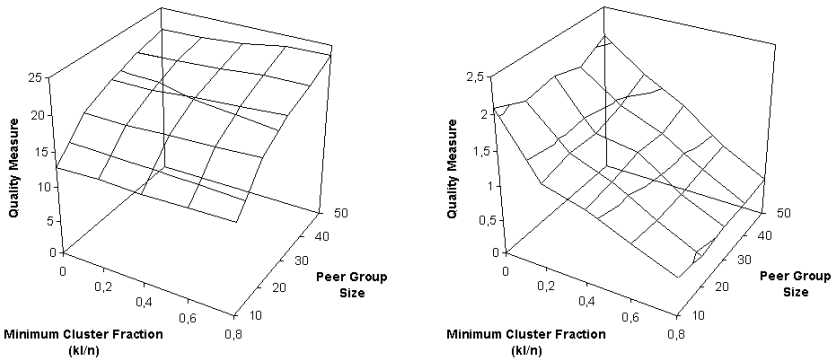

Figure 6.6: Quality for L_∞ distance

chosen according to a probability distribution. K,l-means clustering algorithm is evaluated for benchmarking on two distributions. First, each data point is chosen independently and uniformly. Second, in the pre-clustered distribution k cluster centers are chosen and then $\frac{n}{k}$ data points with a fixed distance of 1 are chosen for each cluster. The second distribution implies that there exists a clustering with an average distance of 1 from the cluster center and a variance of 1.

Before evaluating the performance of the peer group formation, a measure for the quality of a peer group formation must be defined. The purpose of a peer group is benchmarking, i.e. the more competition among the peer group the better. One can expect that the more coherent the peer group, i.e. the more similar the peer group members, the more competition exists among them. Therefore we define the quality of a peer group as the sum of the spreads of the classification criteria. Let $\alpha_{i,l}$ be the value of the l-th criteria ($l \in [1, m]$) of peer group member $X_i \in \mathbb{Q}_j$. Recall that $\mathbb{Q}_j$ is the set of members of the j-th peer group and that p is the number of peer groups. Then the quality measure qm of a successful peer group formation is:

$$qm = \frac{1}{p} \sum_{j=1}^{p} \sum_{l=1}^{n} \max_{i \in \mathbb{Q}_j}(\alpha_{i,l}) - \min_{i \in \mathbb{Q}_j}(\alpha_{i,l})$$

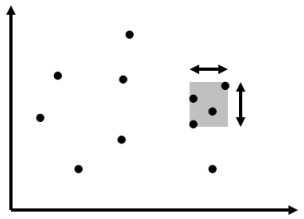

Figure 6.7: Quality metric in the 2-dimensional case

Figure 6.7 depicts an example for the two dimensional case. One can imagine a bounding box around a peer group. The sum of its edge lengths is the quality metric for that peer group. The average for all peer groups is the quality metric of the peer group formation.

The quality guides the selection of the best distance metric. Three common distance metric where compared: the L_1 or Taxicab distance, the L_2 or Euclidean distance, the L_∞ or maximum distance. Let $\vec{x}$ and $\vec{y}$ be two n-dimensional vectors and let x_i and y_i denote the i-th coordinate of $\vec{x}$ and $\vec{y}$, respectively. Then the distance $\delta(\vec{x}, \vec{y})$ from $\vec{x}$ to $\vec{y}$ in each metric are defined as follows:

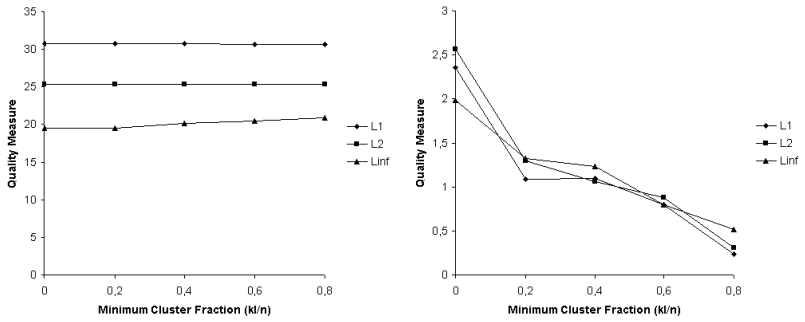

Figure 6.8: Quality for peer group size of 30

L_1:

$$\delta(\vec{x}, \vec{y}) = \sum_{i=1}^{n} |x_i - y_i|$$

L_2:

$$\delta(\vec{x}, \vec{y}) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}$$

L_∞:

$$\delta(\vec{x}, \vec{y}) = \max_{i \in [1,n]} |x_i - y_i|$$

Figures 6.4, 6.5, and 6.6 depict the average quality for peer group formations using k,l-means clustering. On the left side the uniform distribution for data points is depicted and on the right side the pre-clustered distribution. The quality measure is shown in two dimensions for the variables of l as a fraction of $\mu = \frac{n}{k}$ on the x-axis and the average cluster size μ (i.e. peer group size) on the y-axis. The figures are drawn from 3 experiments for 10^4 data points.

The conclusion is that for the uniform distribution the maximum distance performs best and while the Taxicab distance performs worst, while for the pre-clustered distribution no clear distinction can be made. Figure 6.8 shows this comparison more clearly in two diagrams for the average cluster size of 30. On the left side in the uniform distribution case a clear distinction can be made, while on the right in the pre-clustered distribution case no clear distinction is possible. Therefore for the remainder of the experiments the maximum distance has been chosen as the best for peer group formation.

The results of the peer group formation experiments for 10^4 data points are depicted in Figures 6.9, 6.10 and 6.11. The result of the uniform distribution are shown on the left side and the result of the pre-clustered distribution

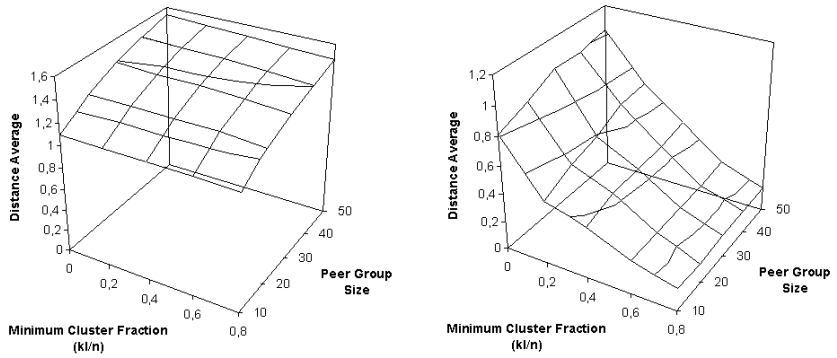

Figure 6.9: Average distance from cluster center

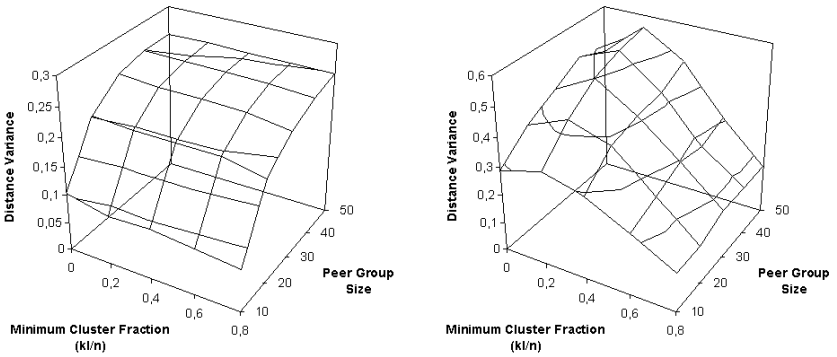

Figure 6.10: Variance of distance from cluster center

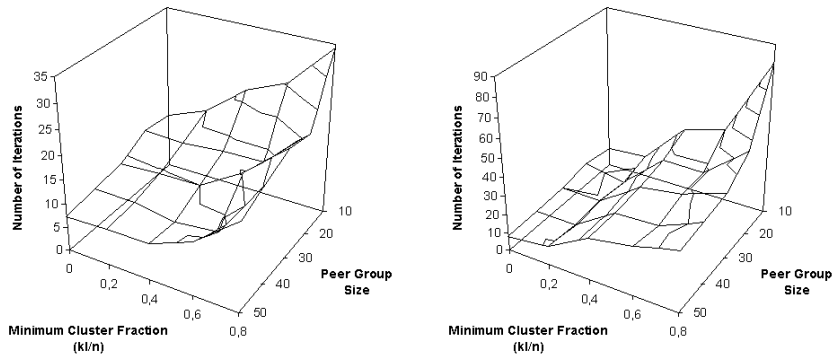

Figure 6.11: Number of iterations of algorithm

on the right. For each graph we show l as a fraction of $\mu = \frac{n}{k}$ on the x-axis from 0 to 0.8 in 0.2 steps: $x = \frac{l}{y} = \frac{kl}{n}$. On the y-axis the average cluster size μ is arranged from 10 to 50 in steps of 10, e.g. then a graph point at 0.6, 20 means a value of $k = 500$ and $l = 12$.

In Figure 6.9 the average distance from the cluster center is depicted on the z-axis. By comparing the graph to the quality measure graph of Figure 6.6 one can see that the average of the distance is also a reasonable quality measure. Furthermore, one should note that the found clustering improves on the pre-clustering, as the average decreases below 1.

Figure 6.10 shows the variance of the distance from the cluster center and Figure 6.11 shows the number of iterations until the cluster centers stabilized on the z-axis. An increasing threshold for the stabilizing criterion is used in the experiment when the number of iterations increases, since in some situations the algorithm does not converge otherwise, e.g. when l is chosen to be equal to μ then the algorithm does not converge for large data point sets.

From the graphs one can conclude that given an uniform distribution, the l parameter has almost no impact on the quality of the result. On the other hand, given a clustered distribution it provides a significant advantage in quality even at low factors at the cost of slightly destabilizing the algorithm (i.e. increasing the number of iterations). Summarizing, the modified k,l-means algorithm performs at least as well as k-means algorithm and may even perform better for benchmarking applications.

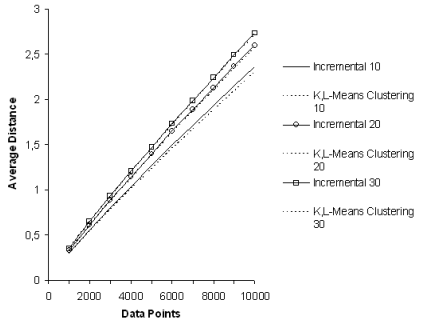

Figure 6.12: Incremental vs. regular k,l-means clustering

The next experiment compared the performance of our incremental algorithm with the regular k,l-means algorithm. The size of the data sets (data points) was increased from 0 to 10^4 and the average distance of both algorithms was measured in intervals of 10^3. The lower limits were 10, 20, and 30 which corresponds to average cluster sizes of 13, 27, and 40, and high limits of 20, 40, and 60 respectively. The results are shown in Figure 6.12 and one can conclude that the performance difference is negligible. Incre-

mental peer group formation is favorable to regular k,l-means clustering due to the limited recomputation effort when a new customer subscribes.

6.3 Protocol Orchestration

The benchmarking protocol operates in a central communication model, but it also operates as every other secure computation protocol in several (4) rounds. Each subscriber needs to execute a sub-protocol with the service provider in each round. The order of the subscribers in each round is arbitrary, but each subscriber's turn needs to happen before the next round can be started. The open question is how do subscribers know that it is their turn, since the protocol is started by the platform provider (or some other party). There are two options: notification and polling.

6.3.1 Notification Model

In the notification model, each client runs a little server that when notified (contacted) triggers the execution of the sub-protocol for that step. The platform can fully coordinate the notifications, and the execution time of the entire protocol is limited only by communication and computation cost (i.e. there is no idle time).

6.3.2 Poll Model

In the polling model, the subscribers poll the platform at intervals and if the subscribers can execute a sub-protocol, they do so. This model carries with it some idle time before the protocol can be completed, since the subscribers are only loosely synchronized. The great advantage of this model is that it does not require a server on the subscriber's side which eases deployment. The subscriber software can be a client deployed behind a corporate firewall that does not allow inbound connections. Therefore this model is preferred.

This paragraph describes a randomized algorithm to determine the interval between polls by the subscriber to the platform. The arrival of new subscribers is modeled with an exponential distribution with a fixed arrival rate λ. The average interval between two arrivals $t_\lambda = \frac{1}{\lambda}$. The incremental peer group formation algorithm is used, such that a new arrival is mapped to an existing peer group and if its size reaches a high limit, the peer group is split into two equal-sized groups that are then recomputed. Every time the subscriber polls the platform it receives two values: the current round $s \in \{0, 1, 2, 3, 4\}$ and the number of subscribers that have completed that round q (including the subscriber itself). She receives the pair $\langle s = 0, q = 1 \rangle$ if the protocol has not been started or is about to finish (i.e. round 4 has been completed by this subscriber). Let q_j be the number of members of the peer group (i.e. l in the k,l-means algorithm) and f be a scaling factor.

The time t_s for the subscriber to sleep until the next poll is set to a random number in the interval between 1 and $f \cdot \frac{t_\lambda}{2^s} \cdot \frac{q_j - q}{q_j - 1}$. The rationale behind this formula is that if the subscriber is the first to poll in a round, her expected waiting time until all subscribers have polled in that round is half their interval. If she is the last in the round, she can execute the next round right away. Between those two events the interval is linearly decreasing.

6.3.3 Comparison

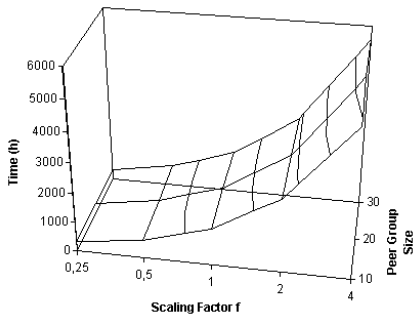

Figure 6.13: Average completion time in polling mode

The polling algorithm has been simulated in an experiment and its expected waiting time has been evaluated. The time to execute the protocol was neglected. Assume one new customer per day, such that fixing a number for λ determines the granularity of our simulation. In the experiment $1440 = 24 \cdot 60$ was used, fixing the granularity at one minute. The average time to completion for the protocol (which is the waiting time in our simulation) is depicted in Figure 6.13. The scaling factor f is depicted on the x-axis and the minimum peer group size l on the y-axis. The average

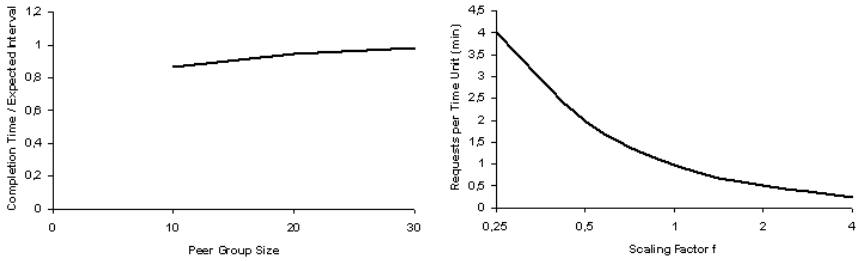

Figure 6.14: Completion time as a fraction of average arrival time and average number of requests per participant per arrival time

time to completion is linear in f (note the logarithmic scale) and the effect of the minimum peer group size is shown in greater detail on the left side of Figure 6.14. The graph is the average of the sections parallel to the y-axis where each have been divided by the scaling factor f. The completion time as a fraction of t_λ slowly increases with the minimum peer group size l, but generally stays below 1.0, i.e. the protocol can be completed within one polling interval. On the other hand the number of polling requests per time unit (one hour with our assumptions) linearly decreases with the scaling factor f as shown on the right side of figure 6.14. One has to pick an appropriate scaling factor f given the requirements and capabilities of a real platform balancing the load by the requests against the time to complete the protocol.

6.4 Summary

This chapter has shown the basis of the architecture of the privacy-preserving benchmarking platform. The first section presented a use case analysis. Three use cases have been identified: registration, statistics retrieval and statistics computation. Separating statistics retrieval from statistics computation allows the subscriber to retrieve statistics immediately following successful registration. This clearly increases the perceived benefit of the subscriber and allows for a more flexible use of the service. The platform offers services for registration and statistics retrieval. Depending on the protocol orchestration model, either both the platform and the subscribers offer services for statistics computation, or only the platform provider does and the subscribers are only clients.

Before one can retrieve (or compute) statistics, subscribers need to be assigned to peer groups. The assignment is called peer group formation. The second section presented a data clustering algorithm that can be used for automatic peer group formation. This data clustering algorithm called k,l-means data clustering is based on k-means data clustering. Its novel extensions is that it allows to specify a minimum cluster size for large problem sizes. Specifying a minimum cluster size is necessary for peer group formation, since peer groups need to remain above a minimum threshold to guarantee privacy and every subscriber should be served. The evaluation results show the applicability of the data clustering algorithm. Furthermore, the data clustering algorithm can be run incrementally, i.e. every time a subscriber registers. Evaluation showed that this can be done without a significant performance decrease.

The third and last section compares the two protocol orchestration models. Protocol orchestration is concerned with the client and server of all necessary communications. In the notification model the subscriber runs a server that can be contacted to initiate the protocol's rounds. In the polling

model the subscriber runs only clients and polls the server when to start the rounds. Evaluation using a specific polling algorithm has shown that with polling, the protocol can be completed in (slightly less than) one polling interval, i.e. practical protocols can be developed in both models.

7 Evaluation

This chapter describes the results of the performance evaluation of the implementation of the benchmarking protocols. In addition to the core implementation of the protocols, an implementation of an entire benchmarking system including a GUI was completed. The system (including the GUI) was demonstrated to several product groups within SAP to create awareness how secure computation protocols can be realized and improve business applications. This chapter focuses on a series of measurements performed to evaluate the protocols as the most critical component under conditions close to real-world deployment. In the first section the test setup and some implementations details are discussed. The second section describes the result for network performance in terms of time. The third section describes the result for network performance in terms of traffic volume. The fourth section shows the result for computational performance depending on the cryptographic parameters. The last section summarizes the results of the evaluation.

7.1 Test Setup

The benchmarking platform software is implemented as web services as described in Section 2.5.1. The secure and authenticated channels are realized with WS-Security as described in Section 2.5.2. Each subscriber client software is installed as a separate web application with its own web service container. To be able to scale the number of subscribers to real-world sizes for peer groups, all such web applications (including the web service containers) are installed in one web application server on one machine. The machine is a "state-of-the-art" (year 2007) server machine with 2 processors and 8 GB of RAM.

As a web application server Tomcat 5.5[1] is used and as a web service container Axis2 1.2[2] is used. For WS-Security Rampart 1.2[3] is used. All implementations are done in Java and are run on the virtual machine of

[1] http://tomcat.apache.org/
[2] http://ws.apache.org/axis2/
[3] http://ws.apache.org/axis2/modules/rampart/1_2/security-module.html

Sun's Java SE 6[4]. Since Tomcat, Axis2 and our protocols are written in Java they are all operating system neutral.

The benchmarking platform itself is run on a separate machine than the clients. This machine has 2 GB of RAM. Separating only the service provider from all clients still requires all communication to cross physical networks. Therefore the network connectivity in the test setup matches real setups very well.

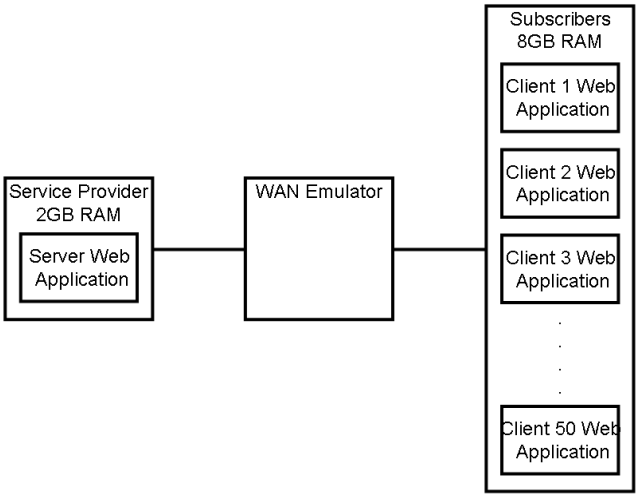

Figure 7.1: Test setup

In order to be able to simulate wide area network (WAN) conditions, a WAN emulator is placed between the two machines. It acts as an IP router and can delay packets. It has the dummynet [68] software installed that can be configured to introduce arbitrary delays. Other than the WAN emulator the machines are connected via a non-dedicated Gigabit Ethernet. Figure 7.1 depicts the test setup in schematic form.

7.2 Network Performance

The goal of this test is to measure the practical performance of the protocol as it would be in a real deployment. The benchmarking platform was implemented and tests were automated, such that it was possible to run the benchmarking protocol without user interference. This implementation was then run with different settings and the actual absolute running time in seconds from starting the protocol until the output of the statistics at the server was measured. Two parameters were modified independently: peer

[4]http://java.sun.com/javase/

group size and network delay.

The peer group size is modified in increments of 5 from 5 to 45. This tests the scalability of the benchmarking protocol. We expect a quadratically increasing running time with increasing peer group size, since communication and computation complexity are $O(q_j^2)$ where q_j is the number of members in the peer group.

The network delay is a parameter of the WAN emulator and simulates real WAN conditions. The delay is increased in increments of 50 ms from 0 ms to 200 ms. The setting of 0 ms corresponds to a network setup where the WAN emulator is not present, i.e. local area network (LAN) conditions, and 200 ms approximately corresponds to the round trip time (RTT) between Germany and Japan over the Internet. Therefore the conditions range from local to distant wide area networks and capture many realistic scenarios. This test evaluates the impact of the network on the overall performance and the proportion of time spent due to network communication. Theory predicts a linear increase of the time with increasing delay, since we use a linear number of connections.

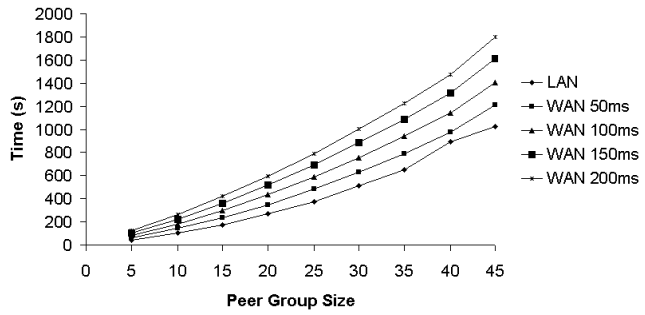

Figure 7.2: Network performance for different peer group sizes

Figure 7.2 depicts the absolute running time in seconds for different group sizes. The graph shows the predicted quadratic increase, although constants are low. The different network conditions are depicted as lines and increasing delay leads to increasing overall running time.

For 5 clients the benchmarking protocol can always be completed in a few minutes: 43 seconds for LAN conditions and 125 seconds for WAN conditions with 200 ms delay. The proportional network overhead for 5 clients is therefore $\frac{125s-43s}{43s} = 191\%$.

With 45 clients a benchmarking protocol run requires several minutes: $1024s = 17$ minutes for LAN conditions and $1800s = 30$ minutes for WAN conditions with 200 ms delay. The proportional network overhead for 45 clients is therefore $\frac{1800s-1024s}{1024s} = 76\%$. The conclusion is that the computational effort increases faster than the network communication overhead.

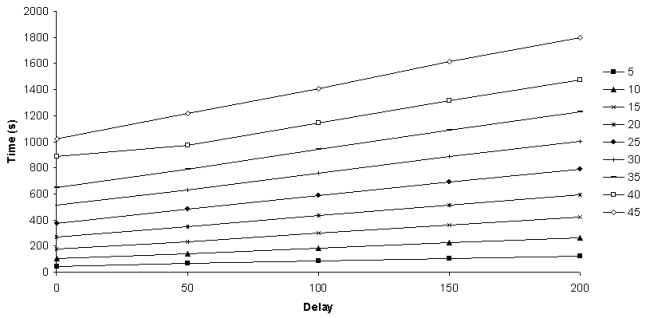

Figure 7.3: Network performance for different delays

Figure 7.3 shows the running time in seconds for different network conditions, i.e. delays. Each peer group size is depicted as a line and, as in Figure 7.2, increasing peer group size leads to an increase in overall performance. The lines show a linear increase as predicted from the theoretic model.

For 5 clients the benchmarking protocol's performance decreases from 43 seconds to 125 seconds: an 82 seconds decrease. For 45 clients the benchmarking protocol's performance decreases from 1024 seconds to 1800 seconds: a 776 seconds decrease. The proportional increase per peer group member is 16.4 seconds for 5 clients and 17.2 seconds for 45 clients. These increases remain fairly constant for all peer group sizes. A summary of the average increases per participant for the different delays is given in Table 7.1.

50 **ms**	100 **ms**	150 **ms**	200 **ms**
3.9 s	8.0 s	12.2 s	16.3 s

Table 7.1: Running time increase per participant

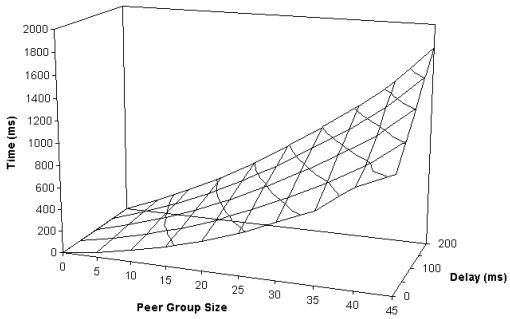

Figure 7.4: Network performance

Figure 7.4 combines Figure 7.2 and Figure 7.3 into one two-dimensional graph. The graph shows the linear increase along its y-axis for the network delay and the super-linear increase along its x-axis for the peer group size.

In conclusion the practical performance of the benchmarking protocol is satisfactory for peer group sizes up to 45. Even over WAN network conditions, the protocol can be completed in minutes, although an off-line computation seems warranted for a 30 minute computation. Peer group sizes of around 45 (or smaller) are acceptable for real-world benchmarking although larger peer groups might sometimes be desirable.

The obstacle to further increases in peer group size is memory. Although in the test setup the benchmarking platform had 2 GB RAM (and 1.5 GB Java heap space) available, it could not complete the test with larger peer group sizes. This is mostly due to the memory consumption of the WS-Security module and the web service stack which are 3rd party tools used in the implementation. A local computation of the benchmarking protocol could easily cope with 200 clients on the same machine, a number beyond practical expectations for peer group size. Therefore it is advisable to improve the resource consumption of the web service stack and in particular the WS-Security module first.

7.3 Network Traffic

The goal of this test is to measure the absolute amount of network traffic in bytes sent over the network between a subscriber and the service provider. The actual amount of traffic gives an indication of the network performance. In the second round of the benchmarking protocol q_j (the number of subscribers in the peer group) data items are sent for rank computation. Therefore the communication complexity between one subscriber and the service provider is expected to scale linearly with the number of subscribers in the protocol.

Figure 7.5, Figure 7.6, and Figure 7.7 depict the network traffic per round for one client in case of 2, 4, and 8 subscribers, respectively. Clearly the communication cost for round 2 dominates the overall computation cost, since more than half of the network traffic is accumulated in round 2. Round 2 not only includes the rank computation, but also the selection via Oblivious Transfer.

The graphs depict the traffic in both directions (from subscriber to service provider and vice-versa) as additive bars for each round with the direction from the service provider to the subscriber on top. All the measurements are in bytes. The setup round informs all clients that the protocol is about to start via a web service method call and contains no other information. The other rounds correspond to the rounds in the benchmarking protocol.

The network traffic measured has been obtained by a self-written TCP

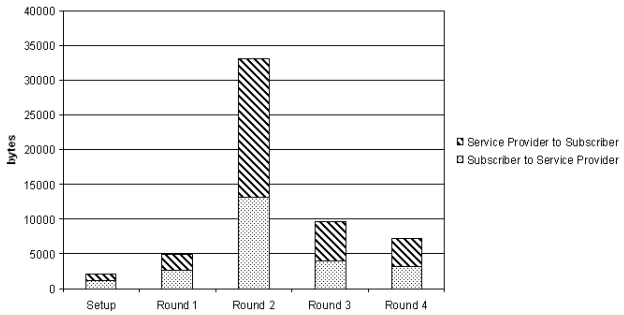

Figure 7.5: Network traffic for 2 subscribers

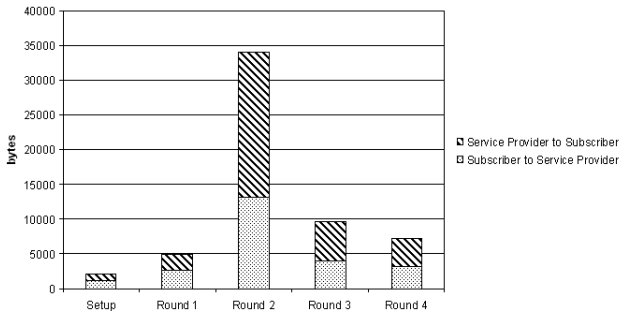

Figure 7.6: Network traffic for 4 subscribers

monitor that intercepts the calls. Therefore the measurement includes all traffic at TCP level including SOAP (see Section 2.5.1) XML markups. Only a fraction of the traffic actually contains data values. Furthermore, all encrypted data values have been encoded as decimal strings and therefore have another expansion in the encoding.

Nevertheless we can conclude that similar amounts of traffic are transported from subscriber to service provider (approx. 25.5 kBytes) as from service provider to subscriber (between 34 kBytes and 37 kBytes). This balance holds throughout all rounds. The traffic from service provider to subscriber increases in round 2 from 19 kBytes to 22 kBytes. We can conclude that one encoded encrypted value therefore requires about 500 bytes of traffic compared to 192 bytes in memory. The traffic for all other rounds remains constant.

Figure 7.8 compares the traffic of the sum of both directions for the different number of clients. The linear increase in round 2 clearly stands out, while all other rounds stay constant.

In summary, the traffic for the benchmarking protocol is reasonable with

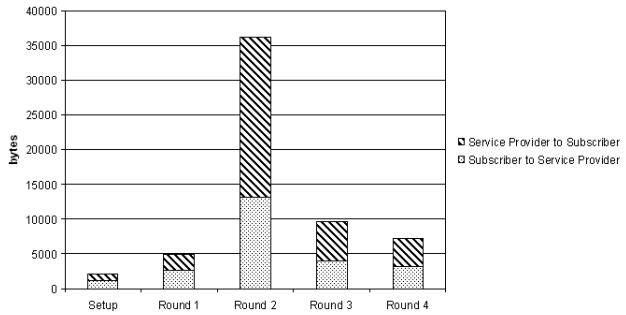

Figure 7.7: Network traffic for 8 subscribers

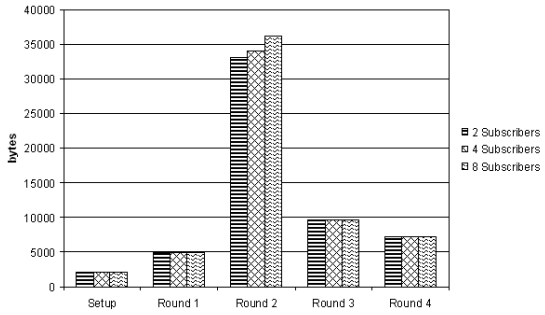

Figure 7.8: Network traffic per round

low amounts of kBytes. A significant overhead for the web service protocols must be included (192 bytes to 500 for an encrypted value). Nevertheless compared to the simple plain-text submission of KPI values, network traffic is significantly larger.

7.4 Key Length

This experiment evaluates the impact of the key length of the homomorphic encryption system on the overall performance. All experiments are run locally, i.e. there are no web service invocations and WS-Security encryptions. The performance was measured for multiple runs (10) and then averaged.

The encryption system used is Paillier's encryption system [63]. There is only one key pair $D_{common}(\cdot)/E_{common}(\cdot)$ in the modified benchmarking protocol and the key length of this key pair is varied throughout the experiments: starting with a key size of 512 bit up to a key size of 2048 bit in 256 bit increments.

Figure 7.9 shows the local performance over the peer group size for dif-

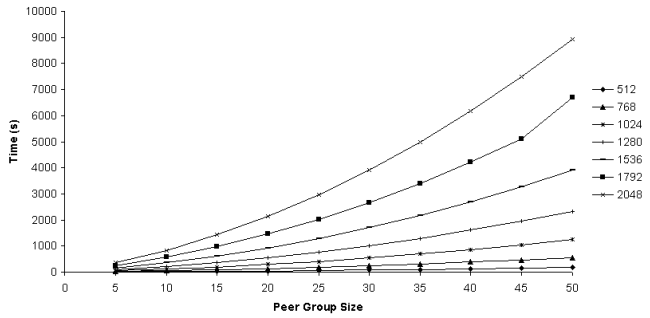

Figure 7.9: Local performance over peer group size

ferent key length. Each line depicts a different key length. The graph shows a super linear increase as can be expected from the $O(n^2)$ computation complexity of the benchmarking protocol. A strong increase between the different key lengths is already visible.

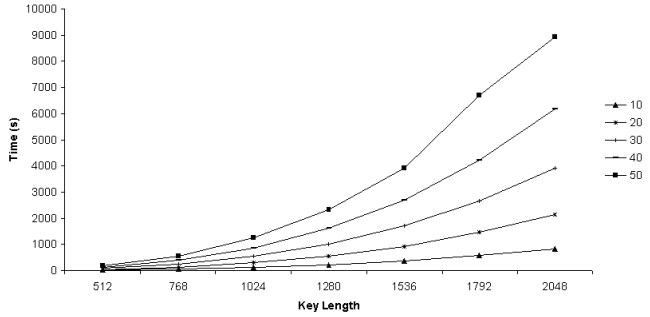

Figure 7.10: Local performance over key length

Figure 7.10 shows the performance over the key length for different peer group sizes. To simplify the graph only peer group sizes of 10, 20, 30, 40 and 50 are depicted as separate lines. There is clearly a super linear increase for each peer group size. This can be expected due to the $O(k^3)$ complexity of modular exponentiation used in the encryption system throughout the protocol.

The impact is nevertheless quite significant. The running time increases from 189 seconds to 8932 seconds for a peer group of size 50. Running times spanning several hours (2.5 in this case) are at the borderline of practical feasibility.

Figure 7.11 depicts the two-dimensional combination of the graphs. A super linear increase in the direction of both axes is visible. Given an upper threshold one can determine the acceptable key length using this graph.

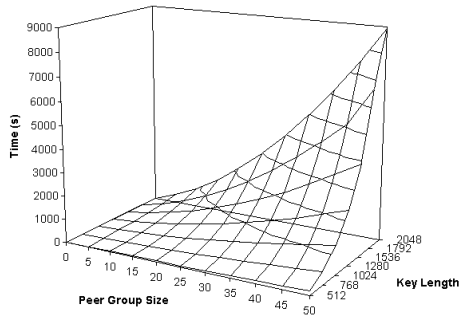

Figure 7.11: Local performance over key length and peer group size

In summary, key length has a significant impact on the performance. A careful choice between the security of the system and the performance needs to be made. The key of Paillier's encryption system is equal to a key of the popular RSA encryption system [67]. In the implementation, a key length of 768 bits has been chosen which provides still-reasonable performance. This key length has been used throughout all other experiments. It will probably be necessary to adapt this key length over time.

7.5 Summary

This section described the quantitative experiments with the benchmarking platform. Three main experiments were run to evaluate the impact of network characteristics and security parameters. The measured quantities were:

- absolute and local running time (in seconds)
- network traffic volume (in bytes)

From the experiments the following conclusions can be drawn:

- The benchmarking platform is practically feasible over WAN conditions for reasonable peer group sizes (up to 45).
- Memory consumption of the web service stack on the benchmarking platform system is a key bottleneck.
- Key size has a major impact on performance and needs to be chosen carefully and adapted over time.

The experiments therefore proved the system's practical feasibility although certain bottlenecks and trade-offs have been identified.

8 Related Problems

Several variations of the techniques used for the benchmarking protocols exist which solve related, independently important problems. The composed protocols may be used as general purpose protocols, such as the Yao's millionaires' protocol as seen in Chapter 5 or also for benchmarking, but with improved asymptotic performance. The protocols are only described briefly and without proofs, since the explanation and proof arguments were previously provided for the more elaborate benchmarking protocol in Chapter 4.

8.1 Constant Cost Benchmarking

The constant cost benchmarking protocol serves the same purpose as the benchmarking protocol, except that it only implements:

- average
- variance
- maximum

i.e. it does not implement median and best-in-class. It follows the same centralized communication pattern and has the same security guarantees. Its advantage is that compared to benchmarking protocol it has a better asymptotic performance. While the benchmarking protocol has communication cost $O(q_j^2)$ square in the number of participants, the constant cost benchmarking protocol is linear $O(q_j)$, i.e. constant per participant. The constant-cost benchmarking protocol trades functionality against performance and offers an alternative for very large peer groups or high-latency networks.

Figure 8.1 describes the constant cost benchmarking protocol in the same formal notation as the benchmarking protocol from Chapter 4. The constant cost benchmarking protocol uses the same techniques and follows the same pattern as the benchmarking protocol. A detailed description of the steps of the constant cost benchmarking protocol follows.

First the subscriber X_i sends the encrypted KPI $E_{common}(x_i)$ to the service provider. Compared to the benchmarking protocol of Figure 5.4 the

Round 1:

$$
\begin{array}{lcll}
X_i & \longrightarrow & SP & E_{common}(x_i) \\
SP & \longrightarrow & X_i & E_{common}(c) = E_{common}(-1^{r_3} \cdot (r_1 \cdot (x_i - max) + r_2)) \\
SP & \xrightarrow{\text{OT}} & X_i & E^c = \begin{cases} E_{common}(x_i + r_4) & \text{if } c \geq 0 \oplus (r_3 = 0) \\ E_{common}(max + r_4) & \text{if } c < 0 \oplus (r_3 = 0) \end{cases} \\
X_i & \longrightarrow & SP & E_{common}(max') = E^c \cdot E_{common}(0) \\
SP & & & E_{common}(max) = E_{common}(max' - r_4)
\end{array}
$$

Round 2:

$$
\begin{array}{lcll}
SP & \longrightarrow & X_i & E_{common}(sum) = E_{common}(\sum_{i=1}^{n} x_i) \\
 & & & E_{common}(max) \\
X_i & \longrightarrow & SP & sum \\
 & & & MAC(sum|i, s_{common}) \\
 & & & max \\
 & & & MAC(max|i, s_{common}) \\
 & & & E_{common'}((x_i - \frac{sum}{n})^2)
\end{array}
$$

Round 3:

$$
\begin{array}{lcll}
SP & \longrightarrow & X_i & E_{common'}(sum') = E_{common'}(\sum_{i=1}^{n}(x_i - \frac{sum}{n})^2) \\
 & & & H(MAC(sum|1, s_{common}), \ldots, MAC(sum|n, s_{common})) \\
 & & & H(MAC(max|1, s_{common}), \ldots, MAC(max|n, s_{common})) \\
X_i & \longrightarrow & SP & sum' \\
 & & & MAC(sum'|i, s_{common})
\end{array}
$$

Round 4:

$$
SP \longrightarrow X_i \quad H(MAC(sum'|1, s_{common}), \ldots, MAC(sum'|n, s_{common}))
$$

Figure 8.1: Constant cost benchmarking protocol

rank computation which requires $O(n)$ communication cost per participant has been replaced by a maximum computation based on the Yao's millionaires' protocol from Section 5.1. The service provider chooses three random values r_1, r_2 $(0 \leq r_2 < r_1)$ and r_3 $(0 \leq r_3 \leq 1)$ and computes a multiplicatively hidden comparison value $c = -1^{r_3} \cdot (r_1 \cdot (x_i - max) + r_2)$ for the KPI x_i and a stored maximum value max. This step is still performed in

the first round of the protocol. The computation can entirely be performed on cipher texts as $((E_{common}(x_i) \cdot E_common(max)^{-1})^{r_1} \cdot E_{common}(r_2))^{-1^{r_3}}$ and even the maximum value max is stored encrypted as $E_{common}(max)$. The random bit r_3 selected by SP flips the sign of c and he correspondingly flips the positions of the inputs to the subsequent OT protocol if $r_3 = 1$. After X_i has received c the service provider SP and X_i engage in an OT protocol where X_i selects the maximum of max and x_i (with secret share r_4 and encrypted under $E_{common}(\cdot)$). X_i re-randomizes the chosen cipher text and returns it to SP. The service provider can now set the new maximum value by adding the secret share r_4.

The second round of the protocol is a combination of the second step of a summation protocol (see Section 4.2.3) for sum, the first three steps of simplified decryption protocols for sum and max and the first step for the summation protocol for sum'. The simplified decryption protocol does not add a secret share before sending the cipher text to the subscribers. This can be done, since the result does not need to be rounded by SP before publishing, because there is no requirement for unique KPIs without rank computation and no unique summands are required. In Figure 8.1 above, the service provider SP sends $E_{common}(sum)$ to subscriber X_i (equivalent steps are done for max in round 2 and later for sum'). X_i responds with sum and a message authentication code $MAC(sum|i, s_{common})$ in the same round. In round 3 after receiving all MACs the aggregated hash value $H(MAC(sum|1, s_{common}), \ldots, MAC(sum|n, s_{common}))$ is returned to X_i.

Round 3 combines the second step of the summation protocol for sum', the last steps for decryption of sum and max and the first three steps for decryption of sum'. The last round finishes the (simplified) decryption protocol for sum'.

The protocol steps for median and best in class (based on the rank computation) have been omitted, such that the overall communication cost is constant per participant.

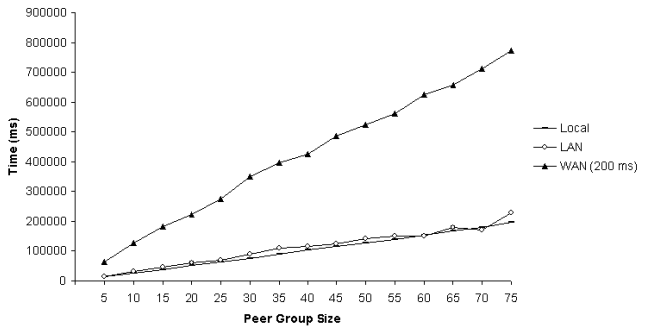

Figure 8.2: Constant cost benchmarking protocol performance

The performance results of Figure 8.2 show that over local area networks (LAN) the performance is computation-bound, but linearly increasing. Over wide area network (WAN) conditions the performance is network-bound, i.e. communication cost becomes the dominating cost factor. The WAN was simulated [68] with a 200 millisecond round trip time (RTT), which approximately corresponds to an Internet connection from Germany to Japan. Implementation was done in Java [40] using web services in the Tomcat/Axis framework and a 768 bit Paillier key length. More details on the implementation are given in Chapter 7. Nevertheless, performance is practical even for large peer groups (75 members) over WAN conditions with less than 13 minutes for one KPI computation.

8.2 Short-lived Common Key

One drawback of the benchmarking protocol is that it uses a common key among all participants X_i that must remain unknown to the service provider SP. This key must be stored in a safe place at each subscriber, since its public revelation would break the security of the benchmarking protocol. A break-in at any subscriber's site is therefore a risk to the security of the benchmarking platform. This section presents a modification of the key setup procedure that chooses a fresh, unique common key for each invocation of the benchmarking protocol. This key is only sensitive during the run of the benchmarking protocol and its revelation afterwards, e.g. due to a break-in, does not affect other protocol runs.

The subscribers run a key agreement protocol using common coin-flipping before each run of the benchmarking protocol. The common coin-flipping protocol is safe against malicious attackers controlling the network, and therefore safe against a malicious service provider. There is a trade-off necessary for the common coin-flipping benchmarking protocol. The benchmarking protocol as described in Chapter 4 and the benchmarking protocol with a short-lived common key differ in two properties.

- Key lifetime
- Anonymity

While the benchmarking protocol with a short-lived common key clearly achieves that each common key is only live for the duration of one protocol run, the common key in the benchmarking protocol lives forever, i.e. until it has been compromised. The other property in which they differ is anonymity. The subscribers in the benchmarking protocol remain strictly anonymous amongst each, i.e. no subscriber is ever referred to with a static (live beyond the run-time of the protocol) identifier. In the benchmarking protocol with a short-lived key every subscriber X_i has a public-, private key-pair and every other subscriber can perform $E_{X_i}()$, i.e. the encryption

with X_i's public key. The benchmarking protocol with a short-lived common key matches the regular benchmarking protocol in all other properties, e.g. security and complexity. The benchmarking protocol with a short-lived common key is secure against a constrained malicious service provider, but not against collusion with the service provider. It has a communication complexity of $O(q_j^2)$ that is already met by the common coin-flipping protocol itself.

Let $E_{X_i}()$ denote the semantically secure encryption with the public key of participant X_i. El-Gamal encryption [28] would be a suitable widely used encryption system. Assume that all subscribers X_i know the public keys of all other subscribers $X_{i'}$ at the beginning of the protocol. This can be achieved if each subscribers is given a global pseudonym σ_i. The certificate authority can then issue a certificate on this pseudonym σ_i during registration. The service provider reveals the composition, i.e. $\mathbb{Q}_j$, to all subscribers before the beginning of the protocol. The service provider can then safely distribute the certificates to the subscribers as well, such that they do not need to permanently store them. The certificate authority is necessary to still achieve security against a constrained malicious service provider. A semi-honest service provider could simply distribute the keys as in [21, 22].

The subscribers X_i can now channel a common coin-flipping protocol of [72] through the service provider. Let $commit(r)$ denote the commitment to a value r. A commitment is a cryptographic function that binds the committer to the value r, but reveals nothing about r. If the committer later reveals r to the verifier, the verifier can verify that r matches the commitment, i.e. the committer has not changed its value r. This is used in the coin-flipping protocol to ensure that all parties independently chose random values before they reveal them to each other.

Round 1:			
X_i	$\longrightarrow$	SP	$commit(r_i)$
Round 2:			
SP	$\longrightarrow$	X_i	$commit(r_1), \ldots, commit(r_{q_j})$
X_i	$\longrightarrow$	SP	$E_{X_1}(r_i), \ldots, E_{X_{i-1}}(r_i), E_{X_{i+1}}(r_i), \ldots, E_{X_{q_j}}(r_i)$
Round 3:			
SP	$\longrightarrow$	X_i	$E_{X_i}(r_1), \ldots, E_{X_i}(r_{i-1}), E_{X_i}(r_{i+1}), \ldots, E_{X_i}(r_{q_j})$
X_i			$r = \sum_{i=1}^{q_j} r_i$

Figure 8.3: Common Coin-Flipping Protocol through Service Provider

The commonly chosen random number r can now be used as a seed to choose a fresh common key for homomorphic encryption $E_{common}()$ for

this run of the protocol. If a subscriber gets compromised, it is no longer necessary to re-issue a new common key to all subscribers, but only the key of the compromised subscriber needs to be updated.

8.3 Coalition-Safe Benchmarking

As mentioned in Section 8.2, one drawback of the benchmarking protocol is that it uses a common key among all participants X_i that must remain unknown to the service provider SP. This section presents a protocol version that uses a threshold variant of this key and is secure against coalitions of up to $t-1$ parties including the service provider SP. Note that the benchmarking protocol is not secure against collusion with the service provider. Such security against collusion with the service provider provides the guarantee that if $t-1$ parties (including the service provider) get (passively) compromised, the benchmarking platform can continue to operate, although with a reduced security level. It is not necessary to restart the entire system including key distribution, if one subscriber gets compromised. The benchmarking platform can continue to run, although with a reduced security level.

The following security goal for coalition-safe benchmarking can be stated:

Definition 8.1 *A coalition-safe benchmarking protocol can tolerate collusion of up* $t-1$ *parties including the service provider without sacrificing its security.*

The protocol is secure if at most $t-1$ parties (including the service provider) are compromised, where t is the threshold for security against coalitions. The threshold t should be chosen, such that it is $O(q_j)$, but in our case $t < q_j$. The guidelines for picking t are described in Section 8.3.3.

8.3.1 Trade-Offs

The coalition-safe benchmarking protocol, the regular benchmarking protocol and the state-of-the-art from the literature differ in several properties.

- Anonymity
- Security
- Function Definition (Leakage)
- Complexity

The coalition-safe benchmarking protocol requires global identifiers for each participant in order to reconstruct the plain text during distributed decryption. The protocol is therefore at best *pseudonymous* and no longer *anonymous*, i.e. each party is statically identified by a pseudonym (to all

subscribers) and no longer anonymous (among all subscribers). No protocol for secure statistics computation besides the benchmarking protocol or general secure multi-party computation currently considers anonymity. They all either require secure, authenticated channels between all parties or have unique keys for each subscriber in a centralized communication model [21, 22].

The loss of anonymity implies that subscribers are now able to track the composition of a peer group. The impossibility of multi-group benchmarking now does not only apply to the privacy against the service provider, but also to the privacy against other subscribers. This additional information for the subscribers compared to the anonymous benchmarking protocol might reduce the willingness of a customer to join the benchmarking platform.

The coalition-safe benchmarking protocol is only secure in the *semi-honest* setting and no longer in the *constrained malicious* setting, but the coalition-safe benchmarking protocol is secure against coalitions with the service provider, while the regular benchmarking protocol only provides security against coalitions without the service provider. Security in the semi-honest model can be motivated by a systems perspective (attackers cannot modify the software), whereas the constrained malicious model has an economic motivation. The main obstacle to constrained malicious security is that each subscriber decrypts q_j intermediate cipher texts during the benchmarking protocol. Without guarantees that these values have been computed according to the protocol specification, a coalition of a subscriber and the service provider can cheat by decrypting all input values. It is important to note that each protocol secure in the semi-honest model can be transferred into a protocol secure in the malicious model by following the compiler of [34]. Security in the malicious model implies security in the constrained malicious model.

Centralized statistics computations [21, 22] so far only consider collusion of subscribers and require a semi-honest service provider. While security against collusion of malicious subscribers is provided, no such security guarantee exists for collusion with the service provider. The protocols [21, 22] are not secure against coalitions with the service provider for any number of subscribers even in the semi-honest model. A semi-honest service provider does not match the economic requirements of the application, since distrust in the service provider can be assumed and security is used a differentiating sales argument.

The coalition-safe benchmarking protocol sightly *leaks* information in addition to the benchmarking functionality. It leaks information about identical input values. This can be tolerated in practice, but must be taken into account in the function definition of the security proof.

The computation and communication complexity of the coalition-safe benchmarking protocol is *cubic* ($O(q_j^3)$) as opposed to *square* ($O(q_j^2)$) as in the regular benchmarking protocol. As the regular benchmarking pro-

tocol can already be considered borderline in terms of practical complexity, coalition-safe benchmarking cannot be considered practical anymore. It therefore does not support the main thesis of this dissertation that a practical benchmarking platform is practically feasible. In particular this would apply for protocols secure in the malicious model, where general (and complex) guarantees for adherence to protocol specification must be given. The communication complexity of the coalition-safe benchmarking protocol is $O(q_j^3)$.

The complexity of the benchmarking protocol stems from the median computation, as can be seen from the constant cost benchmarking protocol of Section 8.1. The only other secure multi-party computation protocol for computing the median [1] has a communication complexity of $O(q_j^2 \log dom(x))$. The logarithm of the domain of the input values $\log dom(x)$ is roughly equal to the number of subscriber q_j in our practical cases. Therefore there is no more efficient protocol than ours available to compute the median. No additional complexity is required for the central communication pattern, since [1] requires point-to-point communication. No complexity figures for security in the malicious model are given in [1].

In summary, the coalition-safe benchmarking protocol and the benchmarking protocol combine different trade-offs in the described properties for anonymity, security and complexity in the centralized communication model. The coalition-safe benchmarking protocol offers safety against the collusion with the service provider at state-of-the-art complexity, while the benchmarking protocol improves complexity at the loss of security against collusion.

8.3.2 Threshold Encryption

In order to improve the security of the benchmarking protocol and transform it into a coalition-safe protocol threshold encryption is used. In threshold encryption the decryption function is distributed among n parties. No single party can decrypt a cipher text, but only if t or more parties cooperate they can decrypt the cipher text. It may be that $t < n$, such that any subset of size t of the parties can decrypt a cipher text.

In [20] a threshold variant of Paillier's encryption scheme [63] is described. It inherits all other properties of Paillier's scheme, i.e. it is public-key, semantically secure and homomorphic in the addition operation. The private key is shared using Shamir's threshold secret sharing scheme [73] and any subset of t parties can decrypt a cipher text. Each share s_i of the private key given to X_i reveals nothing about the entire private key or the share of any other party.

The decryption function is distributed as follows: Let $d_i(c, s_i)$ denote the local decryption step at party X_i using as input the cipher text c and X_i's share s_i of the private key. The computed share $d_i(c, s_i)$ of the plain-

text reveals nothing about the plaintext. Even $t-1$ computed shares of the plaintext reveal nothing about the plaintext. Let $d(i_1, \ldots, i_t, d_{i_1}(c, s_{i_1}), \ldots, d_{i_1}(c, s_{i_1}))$ denote the combination function for all shares. After exchanging the shares $d_i(c, s_i)$ of the plaintext (and the identifier i) this function can be performed locally to compute the plaintext m, i.e. $m = d(i_1, \ldots, i_t, d_{i_1}(c, s_{i_1}), \ldots, d_{i_1}(c, s_{i_1}))$. Both $d_i(\cdot)$ and $d(\cdot)$ can be performed locally, only the result of $d_i(\cdot)$ needs to be transferred.

Let $E^t_{common}(\cdot)$ denote encryption in t-threshold homomorphic, public-key, semantically secure encryption scheme using a common (shared) key among all subscribers and unknown to the service provider. Let $D^i_{common}(\cdot)$ denote X_i's share of the plaintext computed using its share of the key, i.e. $D^i_{common}(\cdot) = d_i(\cdot)$ for $E^t_{common}(\cdot)$.

8.3.3 Key Distribution

The goal of the coalition-safe protocol is to protect against collusion of up $t-1$ parties including the service provider. Each subscriber X_i is given a share of the common private key, such that she can perform $D^i_{common}(\cdot)$. There are several options for distributing these keys (key shares) to the subscribers. According to [20] key distribution requires a dealer that can only theoretically be replaced by a secure multi-party computation protocol.

The first option is similar to the key distribution in the benchmarking protocol. The certificate authority distributes long-living key shares during registration to all subscribers. Each subscriber is given a pseudonym σ_i as in the intermediate solution and the service provider reveals the composition of the peer group to the subscribers at the beginning of the protocol. The parameter t in this solution must be chosen smaller than the minimum peer group size, which should in practice still be $O(q_j)$. For larger peer group sizes t is significantly smaller than q_j, since t is a system-wide constant.

The second option is similar to the intermediate solution and fresh shares are distributed for each run of the protocol. Then $t = n-1$ and the security of the coalition-safe protocol matches the state-of-the-art in the semi-honest model. The drawback is that in practice a trusted dealer is necessary to distribute the shares. Using the certificate authority to play this role involves it in every protocol run degenerating the protocol to a two-server protocol for which more elegant solutions exist (see [60]). Therefore this solution is practically infeasible, since the economic motivation of the service provider is no longer met and he must involve a second, mutually distrustful service provider.

In summary the shares of the common key can be distributed similarly to the key distribution of the benchmarking protocol while not deviating from the originally set goals except in the anonymity property, i.e. the single service provider can remain while anonymity must be given up. This solution makes the threshold parameter t a system-wide constant.

Round 1:				
1	X_i	$\longrightarrow$	SP	$E^t_{common}(x_i)$
Round 2:				
2	SP	$\longrightarrow$	X_i	$E^t_{common}(\mathbf{C}) = (\ldots,$
				$E^t_{common}(c_{l,l'}) = E^t_{common}(r1_{l,l'} \cdot (x_l - x_{l'}) + r2_{l,l'}), \ldots)$
3				$E^t_{common}(\vec{x})$
4	X_i	$\longrightarrow$	SP	$E^t_{common}(\mathbf{C}_{\Phi_i}) = (\ldots, E^t_{common}(r3_{l,l'} \cdot c_{\Phi_i(l),\Phi_i(l')} + r4_{l,l'}), \ldots)$
5				$E^t_{common}(\vec{x}_{\Phi_i}) = (\ldots, E^t_{common}(x_{\Phi_i(l)}) \cdot E^t_{common}(0), \ldots)$
6	SP			$E^t_{common}(\mathbf{C}) = E^t_{common}(\mathbf{C}_{\Phi_i})$
7				$E^t_{common}(\vec{x}) = E^t_{common}(\vec{x}_{\Phi_i})$
Round 3:				
8	SP	$\longrightarrow$	X_i	$E^t_{common}(sum) = E^t_{common}(\sum_{i=1}^{q_j} x_i)$
9				$E^t_{common}(\mathbf{C})$
10	X_i	$\longrightarrow$	SP	$D^i_{common}(E^t_{common}(sum))$
11				$D^i_{common}(E^t_{common}(\mathbf{C}))$
Round 4:				
12	SP	$\longrightarrow$	X_i	$E^t_{common}(median) = E^t_{common}(x_l)$ where $pos(\vec{c}_l) = \lceil \frac{q_j}{2} \rceil$
13				$E^t_{common}(bic) = \prod E^t_{common}(x_l)$ where $pos(\vec{c}_l) \leq \lceil \frac{3}{4} q_j \rceil$
14				$E^t_{common}(max) = E^t_{common}(x_l)$ where $pos(\vec{c}_l) = q_j$
15				sum
16	X_i	$\longrightarrow$	SP	$D^i_{common}(E^t_{common}(median))$
17				$D^i_{common}(E^t_{common}(bic))$
18				$D^i_{common}(E^t_{common}(max))$
19				$E^t_{common}((x_i - \frac{sum}{q_j})^2)$
Round 5:				
20	SP	$\longrightarrow$	X_i	$E^t_{common}(sum') = E^t_{common}(\sum_{i=1}^{q_j}(x_i - avg)^2)$
21				$median$
22				bic
23				max
24	X_i	$\longrightarrow$	SP	$D^i_{common}(E^t_{common}(sum'))$
Round 6:				
25		sum'		

Figure 8.4: Coalition-Safe Benchmarking protocol

8.3.4 Protocol

This section describes the coalition-safe variation of the benchmarking protocol. The design principle of the coalition-safe protocol is that no party including the service provider shall obtain any additional share of a plaintext that is not decrypted later. This ensures that any coalition of up to $t-1$ parties has at most $t-1$ shares which still ensures the confidentiality of the plaintext.

During the benchmarking protocol many intermediate values get decrypted in order to compute the rankings of the values. To prevent parties from obtaining shares for these plain texts, the intermediate values must be decrypted commonly. Although a single value does not reveal any infor-

mation, their entirety reveals the ranks of the input values. In the benchmarking protocol the service provider chooses a permutation to separate the ranks from the input values and blinds the comparison values, such that they reveal no information, but since coalitions with the service provider are allowed, this permutation and blinding must be chosen commonly. The service provider sends all comparison values of the $q_j \times q_j$ matrix $\mathbf{C}$ to all subscribers. Each subscriber (in turn) chooses a permutation $\Phi_i(\cdot)$ of the rows. Each comparison value is multiplicatively hidden again, i.e. each subscriber X_i chooses two random numbers r and r' ($r' < r$) for each cipher text. He computes $E_{common}(r \cdot c + r')$ which does not change the sign of the comparison value. The final permutation and hiding, i.e. the combination of all individual permutations and hidings, is unknown to each participant or even a coalition of $q_j - 1$ subscribers (plus the service provider). Then these values are decrypted and from an identically (to the row permutation) permuted vector of all inputs the corresponding cipher text for each rank is chosen. These are decrypted similarly to the intermediate values. The structure of the remainder of the protocol remains. The complete coalition-safe protocol is depicted in Figure 8.4.

The steps of the coalition-safe benchmarking protocol are described in detail in the following paragraphs. Recall, that no party can decrypt any cipher text of $E^t_{common}(\cdot)$ by himself. The coalition-safe benchmarking protocol is composed of the same building blocks, summation, rank computation, and decryption, as the benchmarking protocol. Selection is no longer necessary, since it can be done locally at the service provider who has now access to the comparison values. Most building block protocols are modified substantially to fit threshold encryption.

Summation

Summation is not modified to fit threshold encryption. The subscriber X_i still sends the encrypted KPI $E^t_{common}(x_i)$ to the service provider SP. SP sums by multiplying the cipher texts, i.e.

$$E^t_{common}(\sum_{i=1}^{q_j} x_i) = \prod_{i=1}^{q_j} E^t_{common}(x_i)$$

There are two summation sub-protocols in the coalition-safe benchmarking protocol (steps 1, 8, 19, and 20).

Rank Computation

After having received all encrypted KPIs $E^t_{common}(\vec{x})$ from all subscribers X_i, the service provider SP computes a matrix $E^t_{common}(\mathbf{C})$. This $E^t_{common}(\mathbf{C})$ contains at position l, l' an entry for the KPI pair $x_l, x_{l'}$ where

$$\begin{aligned} E^t_{common}(c_{l,l'}) &= E^t_{common}(r_{1_{l,l'}} \cdot (x_l - x_{l'}) + r_{2_{l,l'}}) \\ &= (E^t_{common}(x_l) - E^t_{common}(x_{l'})^{-1})^{r_{1_{l,l'}}} \cdot E^t_{common}(r_{2_{l,l'}}) \end{aligned}$$

is computed as the comparison value as before. This can be completed before any communication in round 2 is taking place.

This matrix $E^t_{common}(\mathbf{C})$ is then sent along with $E^t_{common}(\vec{x})$ to subscriber X_i (steps 2 and 3). Each subscriber X_i chooses uniformly a random permutation $\Phi_i(\cdot)$ of $[1, q_j]$ and random numbers $r_{3_{l,l'}}$ and $r_{4_{l,l'}}$ $(0 \leq r_{4_{l,l'}} \leq r_{3_{l,l'}})$ for each pair l, l'. He permutes $E^t_{common}(\mathbf{C})$ according to $\Phi_i(\cdot)$ and multiplicatively hides each comparison value $c_{l,l'}$ (again). The plain text size of the homomorphic, threshold encryption scheme must be chosen accordingly and the leakage is limited to q_j samples as in Section 5.4.3, but only in coalitions with the service provider. He also permutes the vector of KPIs $E^t_{common}(\vec{x})$ by the same permutation $\Phi_i(\cdot)$ and re-randomizes each cipher text by homomorphically adding 0. The result is returned to the service provider SP (steps 4 and 5).

The service provider now updates his local storage of $E^t_{common}(\mathbf{C})$ and $E^t_{common}(\vec{x})$ (steps 6 and 7). He then forwards the already permuted and additionally multiplicatively hidden values in matrix $E^t_{common}(\mathbf{C})$ to the next subscriber X_i, such that the multiplicative hidings add up and the permutations form a joint permutation. This sub-protocol step requires the subscribers to sequentially communicate with the service provider SP. This is repeated until all subscribers have permuted and multiplicative hidden the comparison values. The matrix $E^t_{common}(\mathbf{C})$ needs to be decrypted for selection, but this is done using a regular decryption protocol described later.

Decryption

The most common sub-protocol in the coalition-safe benchmarking protocol is threshold decryption. For threshold decryption all subscribers must collaborate. Recall that each subscriber X_i holds a share of the private decryption key with which he can perform $D^i_{common}(\cdot)$.

The service provider holds a cipher text, e.g. $E^t_{common}(x)$ of plain text x. He distributes the cipher text to all subscribers X_i and collects their plain text shares as the decryption $D^i_{common}(E^t_{common}(x))$. He then combines the shares and sends the result x to each subscriber X_i in the next round. This completes the decryption sub-protocol. As in the decryption sub-protocol of the benchmarking protocol the service provider obtains the plain text first and then publishes it. An assurance that he submitted the same cipher text to all subscribers is not necessary in the semi-honest model only in

the constrained malicious model. The coalition-safe benchmarking protocol provides guarantees against coalitions, but only in the semi-honest model.

The decryption protocol is performed for all outputs *sum* (steps 8, 10 and 15), *median* (steps 12, 16, and 21), *bic* (steps 13, 17, and 22), *max* (steps 14, 18, and 23), sum' (steps 20, 24, and 25) and as well as intermediate values such as $E^t_{common}(\mathbf{C})$ (steps 9 and 11). The difference is that for intermediate values the last step of distributing the plain text is omitted.

Combined Protocol

This completes the sub-protocols of the coalition-safe benchmarking protocol. Each step of the protocol has been explained and is carefully crafted within the composed protocol in order to minimize the overall rounds and balance the computational load between rounds if possible. The final protocol is depicted in Figure 8.4.

8.3.5 Proof Sketch

The main security theorem for the coalition-safe protocol can be stated as:

Theorem 8.2 *The coalition-safe protocol $t-1$-privately computes the extended benchmarking functionality in the semi-honest model.*

The extended benchmarking functionality is defined as:

Definition 8.3 *The extended benchmarking functionality comprises*

- *the statistics average, maximum, variance, median and best-in-class of the input values and*
- *the ranks and number of any duplicate (identical) input values.*

The proof follows the same arguments as the security proof in the semi-honest model of the benchmarking protocol, i.e. a simulator for each of the views of the subscriber and the service provider are given. These are standard and are omitted here, since they follow the same principle as the simulators in the benchmarking protocol.

To prove security against coalition it needs to be shown that the view of each coalition can be simulated. Therefore the following lemma is stated

Lemma 8.4 *All simulated values except the shares of plaintext are chosen independently.*

Lemma 8.4 follows directly from the view and can be proven by inspection of the distribution of the simulated values. It implies that no combination of such random value can be used to derive additional information, i.e. each combination can be simulated computationally indistinguishably by another random number.

For the shares another lemma is required.

Lemma 8.5 *To each party X_i or SP only shares from other parties X_j of plain texts that are later decrypted in the protocol are revealed.*

Lemma 8.5 also follows directly from inspection of the simulator and the protocol. It implies that no coalition of $t-1$ parties can obtain t shares of a plaintext and decrypt it.

It follows from Lemma 8.5 that the shares of the view of up $t-1$ parties can be simulated using a simulator for the shares from the plaintext and no additional information can be obtained. It follows from Lemma 8.4 that other values can be simulated using random numbers. Theorem 8.2 follows from the combination of Lemma 8.4 and 8.5, since then all values can still be simulated just providing input and output to the simulator.

8.4 Summary

The chapter describes two variations of the benchmarking protocol: constant cost benchmarking and coalition-safe benchmarking. Constant cost benchmarking trades median and best-in-class against constant communication cost per subscriber, i.e. the constant cost benchmarking protocols do not compute median and best-in-class, but only require constant communication cost per subscriber and therefore scale to larger peer group sizes. The problem of a common-key is addressed by a short-lived common key variation and the coalition-safe benchmarking protocol. The short-lived common key ensures security against off-line break-ins at the cost of anonymity. Coalition-safe benchmarking significantly improves security by replacing the common keys with a threshold encryption system, but trades this gain in security against collusion with the service provider for a quadratic communication cost per subscriber, i.e. it is not likely to be practical any more.

9 Conclusions

This chapter concludes this dissertation and summarizes its accomplishments in order to support the main thesis. It provides an outlook on future topics giving a critical discussion of the generalization of the accomplishments.

9.1 Thesis Summary

Chapter 3 introduces the problem of a privacy-preserving benchmarking platform. It describes the statistics necessary and introduces the central communication model. The first conclusion presented is that a privacy-preserving protocol for the five statistics average, variance, maximum, median and best-in-class in the central communication model is required to realize a privacy-preserving benchmarking platform.

In addition, a first outline of the problem is given and it is shown that in order to keep the KPIs private (from the service provider), a single peer group model is the only feasible one.

In Chapter 4 such a benchmarking protocol is given. The protocol is the first that satisfies all the requirements and is asymptotically more efficient than comparable previous work. The necessary key distribution is reduced to the common public-key infrastructure model.

The protocol is proven secure in the semi-honest security model, such that it is comparable to related work. Furthermore, it is proven secure in the constrained malicious model that supports the economic assumptions of the service provider model.

In conclusion, Chapter 4 shows that a privacy-preserving benchmarking platform can exist in theory and provides building instructions for the concrete realization of one. The fact that such a benchmarking platform is feasible in theory is not surprising given previous work, yet the protocol also contributes its most efficient solution.

Chapter 5 extends the construction of the benchmarking protocol with an efficient technique for privacy-preserving comparison. Its practical performance is measured and clearly superior to related work, but it is based on randomization and as such its practical leakage needs to be evaluated. This dissertation introduces practical assessment of leakage by experimen-

tation. Its measured leakage is very acceptable in practice and it is therefore adopted in the remaining benchmarking protocols.

Chapter 6 analyzed how the benchmarking protocol can be built into a software system. It first identified the need to separate statistics retrieval from statistics computation. This allows for a more flexible design and immediate customer interaction after registration.

Then it solved the problem of peer group formation under the constraints arising from the need for a single peer group model. Each subscriber needs to be assigned a peer group before the benchmarking protocol can start. This peer group assignment needs to be complete and satisfy the minimum size requirement for each peer group. An appropriate algorithm based on data clustering is given.

Furthermore, it analyzed the deployment of the benchmarking platform and the corresponding orchestration models. It is shown that the benchmarking protocol can be expected to complete within one polling interval in a polling model where all subscribers act as clients only.

In conclusion, Chapter 6 shows that it is possible to build a privacy-preserving benchmarking platform around a privacy-preserving benchmarking protocol.

In cooperation with the solution management at SAP, such a platform has been implemented. The results of the evaluation with the system are presented in Chapter 7. It is shown that the completion time of the benchmarking protocol is acceptable under realistic peer group sizes and network conditions. Dependence on the security parameter is also analyzed.

This dissertation demonstrates that a privacy-preserving benchmarking platform is feasible in theory and in practice under real-world requirements and conditions. Not only theoretic results are developed, but an implementation of the platform is also evaluated. One can therefore conclude that a privacy-preserving benchmarking is practically feasible.

9.2 Outlook

This dissertation explores the procedure for building a specific privacy-preserving business application (i.e. benchmarking). Although the application was anticipated in theory, it led to a novel computer systems application and implementation. This dissertation makes few attempts at generalizing the experience gained by building a benchmarking application to other business applications.

It is nevertheless a strong belief of the author that this experience is very valuable in building other privacy-preserving business applications. First of all, the methodology can be applied to other business applications as well. This statement can be supported by the fact that a significant research grant application by the author has been approved which follows the same

methodology for privacy-preserving supply chain management.

In supply chain management (SCM) one can plan the optimal supply chain for a collaboration of companies. It is well-known that a globally optimized supply chain reduces costs compared to local decision making. Despite this, collaborative supply chain planning requires the revelation of sensitive data, such as costs and capacities, and is therefore not adopted in practice. Privacy-preserving SCM avoids this problem by computing the optimization using a secure computation protocol.

This paragraph will summarize the methodology used throughout this dissertation and outline how it could be applied to privacy-preserving SCM.

1. Define the computations necessary and gather all non-functional requirements.

   Supply chain master planning (SCMP) is the most general form of global optimization and it requires a multi-party computation of linear programming problems. The number of participants in supply chain planning is usually low and a service provider model using a central server can be adopted for a 4th party logistics provider.

2. Analyze state-of-the-art and build novel secure computation protocols.

   Existing SCM protocols are two-party and existing linear programming protocols as well. A novel solution is therefore needed.

3. Design and architect a system using the secure computation protocol.

   SCMP requires certain planning periods. These need to be chosen in correspondence to the security requirements. As well, it needs to be shown that the solution does not reveal the inputs. An incentive-compatible distribution of the gains also needs to be identified, such that everybody is inclined to provide the correct inputs.

4. Implement and evaluate the system using a prototype.

   Practical performance is not guaranteed for a complex problem such as linear programming. A practical evaluation is therefore required.

In summary, in case of success one could then conclude that the thesis is true that a practical privacy-preserving SCM is feasible.

Second, one can expect that when applying this dissertation's methodology, similar or related questions will arise, e.g. a low number of constant rounds will probably be required for practical performance. Quadratic communication cost seems to be acceptable, especially in problem with a low number of participants, such as privacy-preserving SCM. Supply chain planning systems can fortunately be built off-line, such that a decoupling of user interaction and protocol execution is already foreseen.

9.3 Summary

This chapter shows that the dissertation supports the thesis that a privacy-preserving benchmarking platform is practically feasible. Then an outlook on how to apply this dissertation's methodology to other privacy-preserving business applications is given.

Bibliography

[1] G. Aggarwal, N. Mishra, and B. Pinkas. Secure Computation of the kth-Ranked Element. *Proceedings of EUROCRYPT*, *Lecture Notes in Computer Science 3027*, pp. 40–55, 2004.

[2] M. Atallah, M. Bykova, J. Li, K. Frikken, and M. Topkara. Private Collaborative Forecasting and Benchmarking. *Proceedings of the ACM Workshop on Privacy in an Electronic Society*, pp. 103–114, 2004.

[3] E. Bach, and J. Shallit. Algorithmic Number Theory. *MIT Press*, 1996.

[4] D. Beaver, S. Micali, and P. Rogaway. The Round Complexity of Secure Protocols. *Proceedings of the 22nd ACM Symposium on Theory of Computing*, pp. 503–513, 1990.

[5] J. Benaloh. Verifiable Secret-Ballot Elections. *PhD thesis, Yale University*, 1987.

[6] K. Bennett, P. Bradley, and A. Demiriz. Constrained K-Means Clustering. *Microsoft Technical Report*, 2000.

[7] M. Ben-Or, and A. Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation. *Proceedings of the 20th ACM symposium on theory of computing*, pp. 1–10, 1988.

[8] P. Bogetoft, I. Damgard, T. Jakobsen, K. Nielsen, J. Pagter, and T. Toft. A Practical Implementation of Secure Auctions Based on Multiparty Integer Computation. *Proceedings of Financial Cryptography*, *Lecture Notes in Computer Science 4107*, pp. 142–147 2006.

[9] P. Bogetoft, and K. Nielsen. Internet Based Benchmarking. *Group Decision and Negotiation 14(3)*, pp. 195–215 2005.

[10] T. Bray, J. Paoli, C. Sperberg-McQueen, E. Maler, and F. Yergeau. Extensible Markup Language (XML) 1.0. *W3C Recommendation*, 2006. Available at *http://www.w3.org/TR/xml/*.

[11] H. Chang, and M. Atallah. Protecting Software Code by Guards. *Proceedings of the ACM Workshop on Security and Privacy in Digital*

Rights Management, *Lecture Notes in Computer Science 2320*, pp. 160–175, 2001.

[12] R. Chinnici, J. Moreau, A. Ryman, and S. Weerawarana. Web Services Description Language (WSDL) 2.0. *W3C Proposed Recommendation*, 2007. Available at *http://www.w3.org/TR/wsdl20/*.

[13] S. Chow, P. Eisen, H. Johnson, and P. van Oorschot. A White-Box DES Implementation for DRM Applications. *Proceedings of the ACM Workshop on Security and Privacy in Digital Rights Management*, pp. 1–15, 2002.

[14] S. Chow, P. Eisen, H. Johnson, P. van Oorschot. White-Box Cryptography and an AES Implementation. *Proceedings of the 9th International Workshop on Selected Areas in Cryptography*, *Lecture Notes in Computer Science 2595*, pp. 250–270, 2002.

[15] R. Cramer, I. Damgard, and U. Maurer. General Secure Multi-Party Computation from any Linear Secret-Sharing Scheme. *Proceedings of EUROCRYPT*, *Lecture Notes in Computer Science 1807*, pp. 316–334, 2000.

[16] R. Cramer, I. Damgard, and J. Nielsen. Multiparty Computation from Threshold Homomorphic Encryption. *Proceedings of EUROCRYPT*, *Lecture Notes in Computer Science 2045*, pp. 280–299, 2001.

[17] C. Crepeau. Equivalence Between Two Flavours of Oblivious Transfers. *Proceedings of CRYPTO*, *Lecture Notes in Computer Science 293*, pp. 350–354, 1987.

[18] J. Crotts, B. Pan, and C. Dimitry. Hospitality Performance Index: A Case Study of Developing an Internet-based Competetive Analysis and Benchmarking Tool for Hospitality Industry. *Proceedings of Conference of Travel and Tourism Research Association*, 2006. Available at *http://www.ota.cofc.edu/pan/TTRA2006HPI.pdf*.

[19] I. Damgard, and Y. Ishai Constant-Round Multiparty Computation Using a Black-Box Pseudorandom Generator. *Proceedings of CRYPTO*, *Lecture Notes in Computer Science 3621*, pp. 378–394, 2005.

[20] I. Damgard, and M. Jurik. A Generalisation, a Simplification and some Applications of Paillers Probabilistic Public-Key System. *Proceedings of International Conference on Theory and Practice of Public-Key Cryptography*, *Lecture Notes in Computer Science 1992*, pp. 119–136, 2001.

[21] G. Di Crescenzo. Private Selective Payment Protocols. *Proceedings of Financial Cryptography*, *Lecture Notes in Computer Science 1962*, pp. 72–89, 2000.

[22] G. Di Crescenzo. Privacy for the Stock Market. *Proceedings of Financial Cryptography*, *Lecture Notes in Computer Science 2339*, pp. 269–288, 2001.

[23] W. Diffie, and M. Hellman. New Directions in Cryptography. *IEEE Transactions on Information Theory 22(6)*, pp. 644–654, 1976.

[24] W. Du, and M. Atallah. Privacy-preserving Cooperative Statistical Analysis. *Proceedings of the 17th Annual Computer Security Applications Conference*, pp. 102–112, 2001.

[25] D. Dolev, and A. Yao. On the security of public key protocols. *IEEE Transactions on Information Theory*, pp. 198–208, 1983.

[26] D. Eastlake, J. Reagle, T. Imamura, B. Dillaway, and E. Simon. XML Encryption Syntax and Processing. *W3C Recommendation*, 2002. Available at *http://www.w3.org/TR/xmlenc-core/*.

[27] D. Eastlake, J. Reagle, D. Solo, M. Bartel, J. Boyer, B. Fox, B. LaMacchia, and E. Simon. XML Signature Syntax and Processing. *W3C Recommendation*, 2002. Available at *http://www.w3.org/TR/xmldsig-core/*.

[28] T. ElGamal. A Public-Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms. *Proceedings of CRYPTO*, *Lecture Notes in Computer Science 196*, pp. 10–18, 1984.

[29] S. Even, O. Goldreich, and A. Lempel. A randomized protocol for signing contracts. *Communications of the ACM 28(6)*, pp. 637–647, 1985.

[30] J. Feigenbaum, B. Pinkas, R. Ryger, and F. Saint-Jean. Secure Computation of Surveys. *Proceedings of the EU Workshop on Secure Multiparty Protocols*, 2004. Available at *http://www.cs.yale.edu/homes/jf/SMP2004.pdf*.

[31] C. Ferris, and J. Farrell. What are Web Services? *Communications of the ACM 46(6)*, p. 31, 2003.

[32] Marc Fischlin. A Cost-Effective Pay-Per-Multiplication Comparison Method for Millionaires. *RSA Security Cryptographer's Track*, *Lecture Notes in Computer Science 2020*, pp.457–471, 2001.

[33] K. Frikken, and M. Atallah. Privacy Preserving Electronic Surveillance. *Proceedings of the 2nd annual Workshop on Privacy in the Electronic Society*, pp. 45–52, 2003.

[34] O. Goldreich. Secure Multi-party Computation. Available at *www.wisdom.weizmann.ac.il/˜oded/pp.html*, 2002.

[35] O. Goldreich. Foundations of Cryptography (1): Basic Tools. *Cambridge University Press*, 2001.

[36] O. Goldreich. Foundations of Cryptography (2): Basic Applications. *Cambridge University Press*, 2004.

[37] O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game. *Proceedings of the 19th ACM conference on theory of computing*, pp. 218–229, 1987.

[38] S. Goldwasser. Multi party computations: past and present. *Proceedings of the 16th ACM symposium on principles of distributed computing*, pp. 1–6, 1997.

[39] S. Goldwasser, and S. Micali. Probabilistic Encryption. *Journal of Computer and Systems Science 28(2)*, pp. 270–299, 1984.

[40] J. Gosling, B. Joy, G. Steele, and Gilad Bracha. Java Language Specification, 2nd Edition. *Addison-Wesley*, 2000.

[41] M. Gudgin, M. Hadley, N. Mendelsohn, J. Moreau, H. Nielsen, A. Karmarkar, and Y. Lafon. SOAP Version 1.2. *W3C Recommendation*, 2007. Available at *http://www.w3.org/TR/soap12/*.

[42] B. Horne, L. Matheson, C. Sheehan, and R. Tarjan. Dynamic Self-Checking Techniques for Improved Tamper Resistance. *Proceedings of the ACM Workshop on Security and Privacy in Digital Rights Management*, *Lecture Notes In Computer Science 2320*, pp. 141 - 159, 2001.

[43] M. Huhns, and M. Singh. Service-Oriented Computing: Key Concepts and Principles. *IEEE Internet Computing 9(1)*, pp. 75–81, 2005.

[44] A. Jain, M. Murty, and P. Flynn. Data Clustering: A Review. *ACM Computing Surveys 31(3)*, pp. 264–323, 1999.

[45] A. Juels, and M. Szydlo. A two-server, sealed-bid auction protocol. *Proceedings of the 6th Conference on Financial Cryptography*, *Lecture Notes in Computer Science 2357*, pp. 72–86, 2002.

[46] J. Jonas. Advanced Analytics in the Anonymized Data Space. Available at *http://jeffjonas.typepad.com/jeff_jonas/2006/03/index.html*, 2006.

[47] E. Karnin, J. Green, and M. Hellman. On Secret Sharing Systems. *IEEE Tranactions on Information Theory 29(1)*, pp. 35–41, 1983.

[48] F. Kerschbaum. Building A Privacy-Preserving Benchmarking Enterprise System. *Proceedings of the 11th IEEE International EDOC Conference*, to appear, 2007.

[49] F. Kerschbaum, and O. Terzidis. Filtering for Private Collaborative Benchmarking. *Proceedings of the International Conference on Emerging Trends in Information and Communication Security*, *Lecture Notes in Computer Science 3995*, pp. 409–422, 2006.

[50] J. Kilian. Founding Crytpography on Oblivious Transfer. *Proceedings of the ACM Symposium on Theory of Computing*, pp. 20–31, 1988.

[51] E. Kiltz, G. Leander, and J. Malone-Lee. Secure Computation of the Mean and Related Statistics. *Proceedings of Theory of Cryptography Conference*, *Lecture Notes in Computer Science 3378*, pp. 283–302, 2005.

[52] D. Herrmann, F. Scheuer, P. Feustel, T. Nowey, and H. Federrath. A Privacy-Preserving Platform for User-Centric Quantitative Benchmarking. *Proceedings of the 6th International Conference on Trust, Privacy and Security in Digital Business*, *Lecture Notes in Computer Science 5695*, pp. 32–41, 2009.

[53] A. Machanavajjhala, J. Gehrke, D. Kifer, and M. Venkitasubramaniam. l-Diversity: Privacy Beyond k-Anonymity. *Proceedings of the 22nd International Conference on Data Engineering*, p. 24, 2006.

[54] J. MacQueen. Some Methods for classification and Analysis of Multivariate Observations. *Proceedings of 5th Berkeley Symposium on Mathematical Statistics and Probability*, pp. 281–297, 1967.

[55] D. Malkhi, N. Nisan, B. Pinkas, and Y. Sella. Fairplay - A Secure Two-party Computation System. *Proceedings of the USENIX security symposium*, pp. 287–302, 2004.

[56] D. Naccache, and J. Stern. A New Public-Key Cryptosystem Based on Higher Residues. *Proceedings of the ACM Conference on Computer and Communications Security*, pp. 59–66, 1998.

[57] A. Nadalin, C. Kaler, R. Monzillo, and P. Hallam-Baker. Web Service Security: SOAP Message Security 1.1. *OASIS Standard Specification*, 2006. Available at *http://www.oasis-open.org/committees/wss/*.

[58] M. Naor, and B. Pinkas. Oblivious Transfer and Polynomial Evaluation. *Proceedings of the ACM Symposium on Theory of Computing*, pp. 245–254, 1999.

[59] M. Naor, and B. Pinkas. Efficient Oblivious Transfer Protocols. *Proceedings of the 12th ACM-SIAM Aymposium on Discrete Algorithms*, pp. 448–457, 2001.

[60] M. Naor, B. Pinkas and R. Sumner. Privacy Preserving Auctions and Mechanism Design. *Proceedings of the 1st ACM Conference on Electronic Commerce*, pp. 129–139, 1999.

[61] J. Nielsen, and M. Schwartzbach. A Domain-Specific Programming Language for Secure Multiparty Computation. *Proceedings of ACM Workshop on Programming Languages and Analysis for Security*, pp. 21–30, 2007.

[62] T. Okamoto, and S. Uchiyama. A new public-key cryptosystem as secure as factoring. *Proceedings of EUROCRYPT*, *Lecture Notes in Computer Science 1403*, pp. 308–318, 1998.

[63] P. Paillier. Public-Key Cryptosystems Based on Composite Degree Residuosity Classes. *Proceedings of EUROCRYPT*, *Lecture Notes in Computer Science 1592*, pp. 223–238, 1999.

[64] A. Pfitzmann, and M. Köhntopp. Anonymity, Unobservability, and Pseudonymity – A proposal for Terminology. *Proceedings of the International Workshop on Design Issues in Anonymity and Unobservability*, *Lecture Notes in Computer Science 2009*, pp. 1–9, 2000.

[65] B. Preneel. Cryptographic hash functions. *European Transactions on Telecommunications 5(4)*, pp. 431–448, 1994.

[66] M. Rabin. How to exchange secrets by oblivious transfer. *Technical Memo TR–81, Aiken Computation Laboratory*, 1981.

[67] R. Rivest, A. Shamir, and L. Adleman. A Method for Obtaining Digital Signatures and Public-Key Cryptosystems. *Communications of the ACM 21(2)*, pp. 120–126, 1978.

[68] L. Rizzo. Dummynet: a simple approach to the evaluation of network protocols. *ACM Computer Communication Review 27(1)*, pp. 31–41, 1997.

[69] J. Sakuma, and S. Kobayashi. A genetic algorithm for privacy preserving combinatorial optimization. *Proceedings of the Genetic and Evolutionary Computing Conference*, pp. 1372–1379, 2007.

[70] P. Samarati and L. Sweeney. Protecting privacy when disclosing information: k-anonymity and its enforcement through generalization and suppression. *Technical Report SRI-CSL-98-04, SRI Computer Science Laboratory*, 1998.

[71] T. Sander, A. Young, and M. Yung. Non-Interactive CryptoComputing For NC1. *Proceedings of the 40th Symposium on Foundations of Computer Science*, 1999.

[72] B. Schneier. Applied Cryptography (2nd edition). *John Wiley & Sons*, 1996.

[73] A. Shamir. How to share a secret. *Communications of the ACM 22(11)*, 1979.

[74] M. Silaghi. SMC: Secure Multiparty Computation Language. Available at *http://www.cs.fit.edu/ msilaghi/SMC/tutorial.html*, 2004.

[75] D. Stinson. Cryptography: Theory and Practice (3rd edition). *Chapman & Hall/CRC*, 2006.

[76] H. Subramaniam, R. Wright, and Z. Yang. Experimental Analysis of Privacy-Preserving Statistics Computation. *Proceedings of the Workshop on Secure Data Management*, *Lecture Notes in Computer Science 3178*, pp. 55–66, 2004.

[77] L. Sweeney. k-anonymity: a model for protecting privacy. *International Journal of Uncertainty, Fuzziness, and Knowledge-based Systems*, pp. 557–570, 2002.

[78] W. Vickrey. Counterspeculation, Auctions, and Competitive Sealed Tenders. *Journal of Finance 16*, pp. 8–37, 1961.

[79] C. Wang, J. Davidson, J. Hill, and J. Knight. Protection of Software-Based Survivability Mechanisms. *Proceedings of the International Conference on Dependable Systems and Networks*, pp. 193–202, 2001.

[80] A. Yao. Protocols for Secure Computations. *Proceedings of the annual IEEE Symposium on Foundations of Computer Science* 23, pp. 160–164, 1982.