COM SYS
Chair of
Communication and Distributed Systems

RWTH AACHEN UNIVERSITY

Martin Henze

Accounting for Privacy in the Cloud Computing Landscape

Reports on Communications and Distributed Systems
Editor: Prof. Dr.-Ing. Klaus Wehrle

VOL 17

Accounting for Privacy in the Cloud Computing Landscape

Von der Fakultät für Mathematik, Informatik und Naturwissenschaften
der RWTH Aachen University zur Erlangung des akademischen Grades
eines Doktors der Naturwissenschaften genehmigte Dissertation

vorgelegt von

Diplom-Informatiker

Martin Henze

aus Mönchengladbach

Berichter:

Prof. Dr.-Ing. Klaus Wehrle
Prof. Dr. Thomas Engel

Tag der mündlichen Prüfung: 22. 11. 2018

Reports on Communications and Distributed Systems

edited by
Prof. Dr.-Ing. Klaus Wehrle
Communication and Distributed Systems,
RWTH Aachen University

Volume 17

Martin Henze

Accounting for Privacy in the Cloud Computing Landscape

Shaker Verlag
Aachen 2018

Bibliographic information published by the Deutsche Nationalbibliothek
The Deutsche Nationalbibliothek lists this publication in the Deutsche Nationalbibliografie; detailed bibliographic data are available in the Internet at http://dnb.d-nb.de.

Zugl.: D 82 (Diss. RWTH Aachen University, 2018)

Printed in Germany.

ISBN 978-3-8440-6389-9
ISSN 2191-0863

Shaker Verlag GmbH • P.O. BOX 101818 • D-52018 Aachen
Phone: 0049/2407/9596-0 • Telefax: 0049/2407/9596-9
Internet: www.shaker.de • e-mail: info@shaker.de

Abstract

Cloud computing enables service operators to efficiently and flexibly utilize resources offered by third party providers instead of having to maintain their own infrastructure. As such, cloud computing offers many advantages over the traditional service delivery model, e.g., failure safety, scalability, cost savings, and a high ease of use. Not only service operators, but also their users benefit from these advantages. As a result, cloud computing has revolutionized service delivery and we observe a tremendous trend for moving services to the cloud. However, this trend of outsourcing services and data to the cloud is limited by serious privacy challenges as evidenced by recent security breaches and privacy incidents such as the global surveillance disclosures. These privacy challenges stem from the technical complexity and missing transparency of cloud computing, opaque legislation with respect to the jurisdiction that applies to users' data, the inherent centrality of the cloud computing market, and missing control of users over the handling of their data.

Overcoming these privacy challenges is key to enable corporate and private users to fully embrace the advantages of cloud computing and hence secure the success of the cloud computing paradigm. Indeed, we observe that cloud providers already account for selected privacy requirements, e.g., by opening special data centers in countries with strict data protection and privacy legislation. Likewise, researchers propose technical approaches to enforce certain privacy requirements either from the client side, e.g., using encryption, or from the service side, e.g., based on trusted hardware. Despite these ongoing efforts, the necessary technical means to fully account for privacy in the cloud computing landscape are still missing.

In this dissertation, we approach the pressing problem of privacy in cloud computing from a different direction: Instead of focusing on single actors, we are convinced that overcoming the inherent privacy challenges of cloud computing requires cooperation between the various actors in the cloud computing landscape, i.e., users, service providers, and infrastructure providers. All these different actors have clear incentives to care for privacy and, with the contributions presented in this dissertation, we provide technical approaches that enable each of them to account for privacy.

As our first contribution to support users in exercising their privacy, we raise awareness for their exposure to cloud services in the context of email services as well as smartphone apps and enable them to anonymously compare their cloud usage to their peers. With privacy requirements-aware cloud infrastructure as our second contribution, we realize user-specified per-data item privacy policies and enable infrastructure providers to adhere to them. We furthermore support service providers in building privacy-preserving cloud services for the Internet of Things in the context of our third contribution by enabling the transparent processing of protected data and by introducing a distributed architecture to secure the control over devices and networks. Finally, with our fourth contribution, we propose a decentralized cloud infrastructure that enables users who strongly distrust cloud providers to completely shift certain services away from the cloud by cooperating with other users.

The contributions of this dissertation highlight that it is both promising and feasible to apply cooperation of different actors to strengthen users' privacy and consequently enable more corporate and private users to benefit from cloud computing.

Kurzfassung

Cloud Computing ermöglicht es Dienstebetreibern auf die Ressourcen von Cloudanbietern zurück zugreifen, anstatt eine eigene Infrastruktur betreiben zu müssen. Dabei bietet Cloud Computing viele Vorteile gegenüber dem traditionellen Betrieb von Diensten, z. B. Ausfallsicherheit, Skalierbarkeit, Kosteneinsparungen und Benutzerfreundlichkeit. Von diesen Vorteilen profitieren nicht nur die Dienstebetreiber selbst, sondern auch deren Nutzer. Infolgedessen beobachten wir einen deutlichen Trend zur Verlagerung von Diensten in die Cloud. Allerdings wird dieser Trend durch gravierende Privatsphäreprobleme eingeschränkt. Dies zeigen beispielsweise aktuelle Privatsphäreverstöße, wie die globale Überwachungsaffäre. Diese Privatsphäreprobleme resultieren aus der technischen Komplexität und der mangelnden Transparenz von Cloud Computing, Unklarheiten über die für Nutzerdaten geltenden Rechtsvorschriften, dem zentralisierten Markt von Cloudangeboten sowie der fehlenden Kontrolle von Nutzern über den Umgang mit ihren Daten in der Cloud.

Diese Privatsphäreprobleme zu lösen ist entscheidend, damit möglichst viele Unternehmen und Privatanwender von den Vorteilen des Cloud Computings profitieren können. In der Tat beobachten wir beispielsweise, dass Cloudanbieter bereits heute spezielle Rechenzentren in Ländern mit strengen Datenschutzbestimmungen betreiben. Aus wissenschaftlicher Sicht existieren zudem technische Ansätze zur Stärkung der Privatsphäre, beispielsweise durch Verschlüsselung auf der Nutzerseite oder basierend auf vertrauenswürdiger Hardware auf der Diensteseite. Trotz dieser stetigen Bemühungen fehlen nach wie vor die notwendigen technischen Mittel, um Privatsphäre im Cloud Computing umfassend zu adressieren.

In dieser Dissertation gehen wir die drängenden Privatsphäreprobleme des Cloud Computings aus einer anderen Perspektive an: Anstatt uns auf einzelne Akteure zu fokussieren, konzentrieren wir uns auf Kooperationen zwischen den verschiedenen Akteuren, d.h. Nutzern, Dienstebetreibern und Infrastrukturanbietern, um die inhärenten Privatsphäreprobleme zu bewältigen. Alle diese Akteure haben klare Anreize, sich um Privatsphärefragen zu kümmern. Im Rahmen dieser Dissertation präsentieren wir technische Ansätze, die es jedem von ihnen ermöglichen, dies umzusetzen.

Als ersten Beitrag unterstützen wir Nutzer indem wir ihre Cloudnutzung im Kontext von E-Mail-Diensten und Smartphone-Apps aufdecken und ihnen ermöglichen, ihre Cloudnutzung anonym miteinander zu vergleichen. Mit unserem zweiten Beitrag realisieren wir benutzerdefinierte Privatsphäreregeln für einzelne Datenstücke und ermöglichen Infrastrukturanbietern, diese Regeln umzusetzen. Zudem unterstützen wir mit unserem dritten Beitrag Dienstebetreiber bei der Entwicklung von sicheren Clouddiensten für das Internet der Dinge, indem wir die transparente Verarbeitung geschützter Daten ermöglichen und eine verteilte Architektur zur abgesicherten Kontrolle von Geräten und Netzwerken bereitstellen. Schließlich präsentieren wir mit unserem vierten Beitrag eine dezentrale Cloudinfrastruktur, die es Nutzern mit starkem Misstrauen gegenüber Cloudanbietern ermöglicht, bestimmte Dienste durch Kooperationen mit anderen Nutzern außerhalb der klassischen Cloud zu realisieren.

In dieser Dissertation zeigen wir das Potenzial sowie die Machbarkeit von Ansätzen zur Stärkung von Privatsphäre durch die Kooperation verschiedener Akteure auf und geben somit mehr Nutzern die Möglichkeit, von Cloud Computing zu profitieren.

To Laura

Acknowledgments

This dissertation concludes an important chapter of my life. There were many people who accompanied me on my way and by doing so directly or indirectly influenced me both on a personal and a professional level. All of them deserve a big and heartfelt *thank you*! This dissertation would not have been possible without your contributions, input, and support. Although I am quite confident that I will not be able to name all of you, I want to at least thank those of you that had the most influence on both me and my dissertation.

First of all, I want to thank Klaus for offering me the possibility to join COMSYS. I especially appreciate the freedom he gave me in choosing and working on my own research topic. Eventually, he entrusted me with guiding my colleagues in the security and privacy group and I am deeply grateful for this opportunity and his confidence in me. During my years at COMSYS, I truly learned a lot regarding research, teaching, mentoring students, paper and proposal writing, organization, and life in general. I also want to thank Thomas, who not only generously agreed to act as the second opponent for my dissertation but also hosted me as a research intern in Luxembourg before I started my endeavors at COMSYS. Furthermore, I would like to thank Gerhard Woeginger and Thomas Noll who agreed to serve on my dissertation committee (the latter on rather short notice, thank you!).

I owe special gratitude to a number of people for advice and guidance at different stages of my career. Florian offered me the opportunity to work on an extremely exciting topic for my Diploma thesis and sparked my interest in pursuing a PhD. Andriy invited me to Luxembourg for a research internship and introduced me to a different approach towards research. René not only put me on the right track as a young, green colleague but also introduced me to the secret of Taiwanese dumplings. Finally, Henrik shared most of my time at COMSYS and often acted as a much-needed counterpart to reflect on my ideas and writing skills.

In hindsight, I could not have asked for more brilliant and motivated students. Here, I would like to especially mention Arthur, Benedikt, David, Erik, Jens, Johannes, Sascha, and Sebastian who pushed their individual thesis topics to the limits and thus provided much-valued contributions to my dissertation. To all 28 thesis students, I am grateful to your contributions and learned a lot from each of you. Further, I had the honor to work with two research interns, Mary and Ritsuma, who provided a different perspective on my work and brought an international flair to the group. Finally, I would like to thank all student research assistants with whom I had the pleasure to work. I am especially thankful for the hard work of Erik, Jan, Ina, and Roman to push our results closer to publication.

I am particularly honored that three of my thesis students decided to join COMSYS to pursue a PhD themselves. Jan, Jens, and Roman are excellent colleagues and were a big help in writing the publications underlying this dissertation. Additionally, I am proud of my other thesis students who decided to start a PhD: Andreas, Arthur, Erik (at other groups at RWTH Aachen University), Asya (at University of Luxembourg), and David (at University of Stuttgart). I am sure that sooner or later, I will have the pleasure to see all of you defending brilliant dissertations yourselves.

COMSYS is a great place to be at because of the other people there. I could not have wished for better office mates than Henrik, Jens, Mónica (with Alejandra), and René. In your own individual ways, all of you made coming to work a pleasure every single day. Henrik and René always had my back and offered much-valued advice. Jens ensured that the office was pre-heated when I arrived and locked the door after I left. Jan, Roman, and Torsten were always available for a good (and distracting) soccer discussion. Claudia, Dirk, Janosch, Kai, Petra, Rainer, and Ulrike always worked hard to keep any organizational and technical issues as distant as possible. Dirk, Kai, and Rainer ensured that the group's work-life balance always remained in order. Besides the people at COMSYS, I am grateful to Andreas, Andriy, Asya, Fabian, Thomas, and everyone else to welcome me to Luxembourg (and Dagstuhl) at different occasions throughout the past years. Furthermore, I had the opportunity to collaborate with many interesting people during the past years. I am especially thankful for having had the opportunity to work with Daniel, Lars, and Michael.

This dissertation would not have been possible without the tremendous help of every single of my co-authors. I learned a lot from working with each for you and cannot thank you enough for your contributions to this dissertation. I am deeply grateful to Benedikt, Jan, Jens, Henrik, Lina, Martin, René, Roman, and Torsten who took up the burden of proof-reading (parts) of this dissertation. Your feedback helped to further improve my line of argumentation and ruled out many inaccuracies and linguistic errors. I take full responsibility for any remaining glitches.

At one of my first jobs (still during high school), a wise man told me that "somehow the salami has to get on the bread". I am deeply grateful to the Federal Ministry of Economic Affairs and Energy (BMWi), the Excellence Initiative of the German federal and state governments, the state of North Rhine-Westphalia, the Federal Ministry of Education and Research (BMBF), as well as the European Union's Horizon 2020 research and innovation program for providing the funds to cover my salary, conference travels, and the much-valued help of student research assistants.

Last but most importantly, I would like to thank my family and friends for their love, friendship, and support. Above all, I am deeply grateful to my wonderful wife Lina for supporting my dream of pursuing a PhD at COMSYS, even if that meant moving to Aachen. Without your (and during the final steps also our lovely daughter Laura's) patience and understanding as well as the real-world perspective and balance you provided, I would not have been able to finish this dissertation.

Contents

1

Introduction

Over the last years, cloud computing has revolutionized service delivery on the Internet: Instead of operating own infrastructure, service providers rely on resources centrally realized by cloud providers in large data centers. To this end, the cloud computing paradigm promises abstracted access to a huge pool of virtually unlimited resources such as processing, storage, and networking. Hence, service providers can easily scale the amount of utilized resources, e.g., to handle spikes in demand while not having to pay for underutilized resources outside peak loads. Furthermore, cloud providers replicate resources to increase availability of cloud-hosted data and services, e.g., in the case of energy outages or networking failures.

Not only services providers, but also corporate and private users of these services benefit from the advantages of cloud computing. Cloud services (i) are often offered for free (especially for private use) or at an affordable price without huge upfront investment, (ii) allow access from nearly everywhere, (iii) offer failure-safe and redundant storage of data and provisioning of computing power, (iv) provide high usability through transparent integration into many devices and applications (e.g., smartphones and web browsers), and (v) obviate the need of maintaining or operating own infrastructure. For example, cloud computing allows companies to operate their email services more flexible, scalable, and cost-efficient [BL07]. Likewise, private users use cloud storage services, such as Dropbox and Google Drive, for storage and synchronization of files [ISKČ11]. The advantages of cloud computing are especially important when considering the limited resources in computing, storage, and power of mobile devices, such as smartphones, or of devices in the Internet of Things (IoT) and Cyber-physical Systems (CPS) [HHK+16, HHH+17], where cloud services are often used to synchronize data across devices and networks.

However, these benefits come at a price: Outsourcing services and data to the cloud leads to serious privacy challenges. In contrast to traditional IT outsourcing, the cloud computing landscape is technically more complex and opaque: Cloud services

often subcontract other cloud services [PP15], e.g., to avoid operating their own infrastructure, to cover peak demands, or to strengthen resilience against attacks. This indirect use of resources leads to a situation where users of cloud services are forced to trust an unknown number of third parties with their sensitive data. As a consequence, it is often unclear under which jurisdiction users' data falls, hence providing users with only very limited legal protection [FM12]. Furthermore, users might not even be aware that they are using cloud resources. From a different perspective, the cloud computing paradigm leads to a centralization of data at a small number of cloud services [Sky16], rendering those to valuable targets for attacks [HHHW16]. The imminent privacy risks of cloud computing hinder the adoption of cloud services for both, corporate and private users [ISKČ11, TPPG13, Rig17].

Importantly, these privacy concerns are not merely an academic problem. Recent privacy incidents, such as the global surveillance disclosures emanating from Edward Snowden [Gel13], demonstrate the fundamental privacy issues of today's public cloud services [TPPG13]. Resulting privacy concerns, missing trust, or legal restrictions on data locality and data ownership make private and corporate users seek for alternatives [ISKČ11, PB10]. To further emphasize these concerns, a survey from the Intel IT Center among 800 IT professionals revealed that 78 % of organizations are concerned that cloud services are unable to meet their privacy requirements [Int12]. In consequence, 57 % of organizations refrain from outsourcing regulated data to the cloud. Hence, the lacking control over the treatment of data when it is outsourced to cloud services scares away a large set of potential clients.

As a result, an inherent need to account for privacy in cloud computing surfaces. First and foremost, privacy is a fundamental human right [UN48] and everyone involved in delivering cloud services is ethically obliged to respect the privacy of individuals. Indeed, users expect that their privacy is respected [JLG08] and hence, respecting users' privacy reduces cloud providers risks for loss of reputation and credibility [Pea09]. Furthermore, providers of cloud infrastructure and especially cloud services are often bound by legal constraints. Neglecting legal obligations can lead to lengthy lawsuits and costly fines, e.g., the European Union's new General Data Protection Regulation (GDPR) imposes penalties up to 20 million Euro or 4% of a company's annual global revenue, whichever is greater, for not complying with data protection regulation [GDPR16]. Finally, we identify clear business incentives for providers of cloud infrastructures and cloud services to cater for privacy: Privacy presents a unique selling point to the untapped market of clients that are currently unable to outsource their data to the cloud as cloud services lack the technical mechanisms to account for privacy requirements [Int12].

Indeed, we observe that cloud providers in the past already adapted to a small set of privacy requirements. For example, to be able to sell its services to the US government, Google created the segregated "Google Apps for Government" and had it certified at the FISMA moderate level, which enables use by US federal agencies and their partners [Goo18b, MNP+11]. Furthermore, cloud providers open data centers around the world to address location requirements of their clients [BRC10]. From a research perspective, current efforts to increase the level of privacy in cloud computing are typically either deployed at the user side, e.g., using client side encryption or obfuscation [PSM09, YWRL10, LYZ+13] as well as distribution of data between

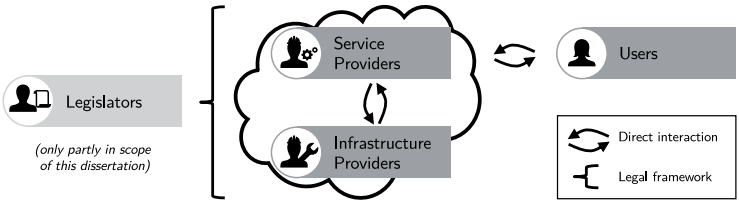

Figure 1.1 Compared to the traditional client-server model, the cloud computing paradigm consists of additional actors with more involved and often indirect interaction.

different cloud services [PP12, SMS13], or the service side, e.g., based on secure execution domains realized on top of trusted hardware [CGJ+09, IKC09, SCF+15].

Despite these efforts, the problem of accounting for privacy in the cloud computing landscape, i.e., considering privacy requirements and expectations during service delivery [BSPW17], is still pressing. In this dissertation, we argue that providing privacy in cloud computing often cannot be achieved without cooperation of the different actors which are involved in the delivery of services. To this end, we first identify clear incentives to account for privacy for all actors in the cloud computing landscape. Based on this, we postulate that overcoming the privacy challenges of cloud computing cannot be achieved by any of the actors alone. Instead, each actor has to contribute the technical means under their control to collaboratively account for privacy. Hence, in this dissertation, we consider the different perspectives on privacy in cloud computing and propose technical approaches to address privacy from the perspective of each actor in the cloud computing landscape.

1.1 Problem Analysis

To better understand the root causes for the privacy challenges of cloud computing, we first study the different actors in the cloud computing landscape. We then derive the different perspectives on privacy in cloud computing of the various actors, which paves the way for our identification of core problems for privacy in cloud computing.

1.1.1 Different Actors in the Cloud Computing Landscape

In contrast to traditional service delivery in the Internet [Han00], cloud computing involves additional actors. As shown in Figure 1.1, we identify four actors in the cloud computing landscape with tighter and often indirect interaction compared to traditional Internet services[1]. In the following, we describe these four actors, their tasks in delivering cloud services, as well as their relationships and interdependencies.

[1] As we discuss in more detail in Section 2.1.2, we slightly simplify the traditional tiered architecture of cloud computing to ease presentation in the context of this dissertation. Furthermore, we here limit our analysis to those four actors that are typically involved in delivering cloud services and hence have a huge impact on privacy. We provide a discussion of all actors in the cloud computing landscape and their roles in Section 2.1.3.

Infrastructure Providers. As the foundation of cloud computing, infrastructure providers deploy the necessary (physical) infrastructure for the realization of cloud services. Most notably, this infrastructure includes computing (often in form of virtual machines) and storage resources as well as broadband network connectivity.

Service Providers. Building on top of cloud infrastructure, service providers realize cloud services, i.e., applications targeting private and corporate users. Cloud services deployed by service providers can be accessed over the Internet.

Users. Utilizing cloud services, private and corporate users rely on resources delivered (directly) by service providers and thus (indirectly) by infrastructure providers. Often, private users access cloud services for free ("paying" with their private information instead, e.g., in the context of targeted advertising [Rob09, PHW17]), while corporate users are predominately charged for using cloud services [FM12].

Legislators. Finally, legislators provide the legal frameworks that govern the provisioning of cloud services and infrastructure. With respect to privacy, this most notably includes data protection legislation. Given the technical scope of this dissertation, we only cover the role of legislation when it directly influences technical decisions. Other aspects of legislation, e.g., policy issues involved in changing privacy regulations within the context of cloud computing, are considered out of scope.

These diverse actors do not only fulfill completely different roles in the cloud computing landscape but also have different perspectives on privacy in cloud computing.

1.1.2 Different Perspectives on Privacy in Cloud Computing

These different perspectives on privacy of the various cloud actors mainly result from different objectives and hence incentives to cater for privacy. Understanding these different perspectives is important for our goal of deriving technical approaches to account for privacy in cloud computing covering all these different perspectives.

Infrastructure and Service Providers

For infrastructure and service providers, the main motivation to cater for privacy is the obligation to adhere to legal regulatory frameworks. Most notably, this includes information privacy and data protection legislation that has now been established in 120 countries worldwide (more than 30 additional countries are currently working on establishing such legislation) [Gre17]. While the precise regulations in these countries show notable differences, we can derive basic principles that most information privacy and data protection legislation addresses [DEG+15, GDPR16]: (i) data on individuals should only be collected for an explicit and legitimate purpose, (ii) collected data on individuals cannot be disclosed to or shared with third parties without individuals' consent, (iii) stored data on individuals needs to be accurate and kept up to date, (iv) individuals should be able to review stored data about them, (v) stored data should be deleted as soon as it is no longer needed, and (vi) data cannot be transmitted to locations with a weaker level of data protection.

Most notably, data protection legislation of a specific jurisdiction can even be applicable if an infrastructure or service provider is located outside this jurisdiction. For example, the European Union's GDPR is applicable whenever the user whose data is being processed is based in the EU. Besides information privacy and data protection legislation, providers also need to cater for other legislation. As an example, the Health Insurance Portability and Accountability Act (HIPAA) [HIPA96] requires that subcontractors have to comply with the same requirements as their contractees when handling electronic health records [Gel09].

Infrastructure and service providers do not only have an incentive to respect privacy to avoid prosecution and punishment, but also to put themselves in favorable market positions. First, providers strive to avoid undesired consequences such as non-acceptance of services or damage to reputation [Pea09, ZGW14]. Second, supporting a wide range of privacy requirements (even beyond what is demanded by legislation) enables the migration of privacy-sensitive or highly regulated services and data to the cloud, hence opening new business opportunities [Int12]. While we identify clear benefits for cloud infrastructure providers and cloud service providers to account for privacy, resulting privacy-friendly cloud offers are virtually non-existing today.

Users

When considering the privacy perspective of users, we have to differentiate between private and corporate users. Private users are mostly concerned about an invasion of their privacy since they inadvertently give up control over their data when using cloud services [ISKČ11, TPPG13]. For example, users are aware that their data stored in the cloud could potentially be accessed by third parties, e.g., hackers, the provider of the cloud storage service, or public authorities, such as law enforcement agencies [ISKČ11]. Still, even if (experienced) users are aware of the consequences of cloud usage in general, they still do not know who exactly can access their data. This lack of knowledge and control is especially due to service providers' usage of own and third party infrastructure that hides who (companies and government agencies) has access to data in the cloud. Since most cloud providers are located outside the user's legislation, contracts and other legislative measures only have a very limited reach of binding applicability [FM12, Sil13]. Due to these concerns, private users ultimately tend to refrain from using cloud-based services, especially for (highly) sensitive data such as personal health records [GGJ17].

In contrast, for corporate users, the reluctance to using cloud services is mainly due to compliance and security concerns [Wal16]. Especially for businesses, compliance with legal and contractual obligations is important to avoid serious (financial) consequences [MNP+11]. German tax legislation, e.g., forbids the storage of tax data outside of Germany [Cor17]. Furthermore, the Sarbanes-Oxley Act (SOX) [SOX02] requires accounting firms in the United States to retain records relevant to audits and reviews for seven years. Contrary, the Payment Card Industry Data Security Standard (PCI DSS) [PCI15] limits the storage duration of data to the time necessary for business, legal, or regulatory purposes after which it has to be deleted. Finally, contracts often require that sensitive data is not colocated with competitors

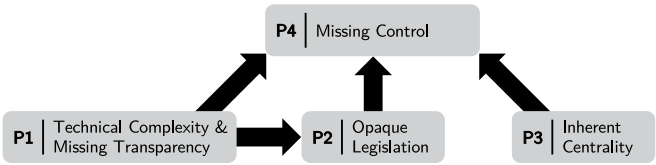

Figure 1.2 In this dissertation, we distill four core problems for privacy in cloud computing, culminating in a lack of control over data when it is outsourced to the cloud.

for fear of leaks or breaches [RTSS09]. Ensuring compliance with these requirements is incredibly difficult with today's cloud offers. Hence, corporate users often cannot benefit from the advantages of cloud computing.

Legislators

When considering legislators, we have to consider that legislation is typically technology agnostic. Hence, the task of legislators is to define and govern a trade-off between the privacy interests of data collectors, data processors, and users in general without regulating cloud computing per se. Still, legislation has to account for the specific setting of cloud computing. Most notably, legislators can be supported with technical approaches for implementing legal requirements, especially with respect to transborder data flows. Since legislation often also follows national interests, regional clouds, e.g., the "Europe-only" cloud currently discussed in the EU [SBC+14, HMR+14], do not only aim at increasing governance and control over data but are also a measure to strengthen the own economy.

1.1.3 Core Problems for Privacy in Cloud Computing

From these different perspectives of privacy, we distill four core problems for privacy in cloud computing which we consider most important [HHHW16, HPH+17] and analyze their interplay as visualized in Figure 1.2. We argue that overcoming these core problems is key to strengthen privacy and consequently, to overcome inherent adoption barriers. In the following, we discuss them in more detail.

P1: Technical Complexity and Missing Transparency

The cloud computing landscape is technically complex and lacks transparency: Most importantly, the abstraction of resources in the cloud computing architecture hides how (technically complex) cloud services are realized, leads to the indirect use of resources (e.g., cloud services realized on top of cloud infrastructure), and hence results in indirect and unknown contractual relationships. Indeed, cloud services often subcontract other cloud services or rely on cloud infrastructure [PP15], e.g., to avoid operating own infrastructure, to increase scalability, or to strengthen resilience against attacks. In this situation with missing transparency of the technical and

contractual realization of cloud services, users are forced to trust an unknown number of third party cloud services with their sensitive data—a situation that has become too complex for users and developers of these services to grasp [GGJ17]. Likewise, technical complexity and missing transparency make it difficult for users to assess which level of privacy can be optimally achieved for a certain cloud functionality.

P2: Opaque Legislation

Given the technical complexity and missing transparency of cloud computing (P1), it is often unclear under which jurisdiction users' data falls, hence offering users only very limited legal protection [FM12]. Furthermore, the jurisdiction under which data falls can change over time, e.g., when data is moved between data centers in different countries to balance load or to react to outages [LM10], especially if cloud providers do not offer to contain data to specific regions. However, the applicable legislation defines who can gain access to stored and processed data. For example, legislation in many countries allows own government agencies, e.g., law enforcement, to access and intercept data in the cloud [Gel13]. The resulting threat to users' privacy became evident with the 2013 global surveillance disclosures [Gel13]. At the same time, in the face of the technically and contractually complex realization of cloud services, even the providers of these cloud services often fail to know where, i.e., which other cloud services and cloud infrastructure, data (they are responsible for) flows to [AGM10]. As a result, users cannot derive which legislation applies to their data when it is stored and processed by a multitude of cloud services.

P3: Inherent Centrality

The cloud market is de facto centralized with a small number of services jointly dominating the market. For example, Skyhigh reports that Amazon Web Services (35.8 %) and Microsoft Azure (29.5 %) provide cloud infrastructure for more than half of the cloud applications deployed on cloud infrastructure in the fourth quarter of 2016 [Sky16]. This centralization of cloud services comes at a price. First, centralized services are a valuable target for attackers, exemplified by a reported 300 % increase in attacked Microsoft user accounts from 2016 to 2017 [Mic17]. Second, concentrating storage and processing of user data at a few providers eases operations for law enforcement agencies [PB10]. Finally, users only have a very limited set of alternative (potentially more privacy-friendly) cloud providers. Furthermore, the migration between cloud providers is nowadays severely hindered by technical incompatibilities and the lack of common standards [SHI+13]. Users are very much aware of the described imminent risks of the centralized cloud computing landscape and these risks significantly hinder the adoption of cloud computing [ISKČ11, TPPG13, GGJ17].

P4: Missing Control

Technical complexity and missing transparency, opaque legislation, as well as inherent centrality all lead to users' loss of control over their data when it is sent

to the cloud [CGJ+09, ISKČ11, TPPG13]. More precisely, any data that is transferred out of the control of its owner might be inadvertently forwarded to third parties, used for unintended purposes, or handled in violation of legal requirements [PB10, TJA10, ZGW14]. Furthermore, missing transparency makes enforcing existing requirements extremely difficult. These issues become especially problematic, since the transfer of data to the cloud often happens imperceptibly, especially for less technically proficient users. For example, mobile applications on smartphones nowadays increasingly rely on cloud services [MBK+12, PHW17]—often without the knowledge, let alone permission, of users. Notably, also cloud services experience the problem of missing control, as they cannot influence the underlying cloud infrastructure or steer the placement of resources, e.g., to prevent colocation with competitors in fear of accidental leaks or deliberate breaches [RTSS09]. As a result of these issues, missing control has been identified as one of the major problems and acceptance hurdles of cloud computing both for private [Pea09, ISKČ11] and corporate users [Int12, Clo15].

These core problems for privacy in cloud computing clearly highlight an inherent need to account for privacy in the cloud computing landscape. In the following, we derive research questions that pave the way towards our contributions to increase the privacy of cloud computing.

1.2 Key Observation and Research Questions

Besides offering enormous benefits, cloud computing also poses serious privacy challenges. To overcome these privacy challenges, we strongly believe that it is insufficient to only focus on a single actor in the cloud computing landscape and instead propose to rely on cooperation between the different actors to realize more privacy friendly cloud services. Nowadays, infrastructure providers have a decent understanding of the technical realization of their infrastructure but do not know about the privacy requirements of providers and users of cloud services realized on top of their infrastructure. Likewise, cloud service providers neither know about the privacy requirements of their users nor can they influence or at least derive information on how the underlying cloud infrastructure is technically realized. Finally, both private and corporate users have no means to influence how cloud services and cloud infrastructures are operated. Hence, the actors in the cloud computing landscape need to cooperate and *each* of the actors has to contribute the necessary technical means under their control to strengthen privacy. From this key observation and the four privacy challenges, we derive three research questions that we address with the contributions of this dissertation.

Q1: How can infrastructure providers support service providers and cloud users in executing control over privacy?

Only cloud infrastructure providers have detailed knowledge about and can control the underlying technical realization of cloud infrastructure. If they knew about the

privacy requirements of providers and users of cloud services, they could combine this knowledge with their detailed understanding of the infrastructure to account for their clients' privacy requirements while provisioning cloud infrastructure.

Q2: How can service providers build privacy-preserving cloud services on top of cloud infrastructure?

Cloud service providers are in a diametral position since they should account for the privacy requirements of their users but have no influence on the (technical) realization of the underlying cloud infrastructure, since major infrastructure providers nowadays do not offer configurability with respect to privacy. Still, when closely cooperating with their users, they can build and operate their cloud services as privacy-preserving as possible given the limited support they receive from cloud infrastructure providers today with respect to privacy.

Q3: How can users preserve their privacy when interacting with cloud services?

Cloud users are arguably the weakest actor in the cloud computing landscape since they cannot influence how cloud services and cloud infrastructure are delivered. Still, when provided with information on the characteristics of their cloud usage, they could decide which (privacy-friendly) cloud services to entrust with their data. Furthermore, they can support service and infrastructure providers by providing them with their privacy requirements. Ultimately, users could even decide to completely move or stay away from all cloud services for certain functionalities with high importance to their privacy.

In this dissertation, we provide answers to these research questions by proposing technical systems that are deployed by the different actors in the cloud computing landscape and address individual aspects underlying these questions. Hence, we make an important step forward to account for privacy in the cloud computing landscape and thus allow more private and corporate users to fully embrace the benefits of cloud computing without having to sacrifice their privacy.

1.3 Contributions

To address these three research questions and hence account for privacy in the cloud computing landscape, we present four distinct contributions in this dissertation:

C1: Transparency approaches to raise users' awareness for cloud usage with respect to the cloud exposure induced by email and smartphone usage based on networking features of cloud services and cloud infrastructure.

C2: Data handling requirements-aware cloud infrastructure which enables users to specify their privacy requirements and thus allows infrastructure providers to incorporate these requirements when selecting cloud storage nodes.

C3: A platform for developing and deploying privacy-preserving cloud services which supports non-security experts in protecting the privacy of users when providing cloud services, showcased in the context of the cloud-based IoT.

C4: A decentralized approach to cloud computing where a certain set of cloud services is shifted to resources that are provided in a secure peer-to-peer manner by trusted entities.

These contributions evolve around our key observation of the imperativeness to account for all actors in the cloud computing landscape when aiming towards providing strong and encompassing privacy for users of cloud services and cloud infrastructure.

To this end, Contributions C1 to C3 work in a setting where different actors collaborate to jointly provide privacy in cloud computing. This typically requires a certain level of trust into the other involved actors. In contrast, Contribution C4 works in a setting where users completely distrust cloud providers and hence collaborate among themselves to realize an alternative to the centralized cloud computing landscape. Together, our four contributions provide the technical means that infrastructure providers, service providers, and users can rely on to strengthen privacy in cloud computing. Furthermore, they jointly address the four core privacy problems of cloud computing. In the following, we summarize our four contributions.

C1: Raising Awareness for Cloud Usage

Users are often unaware of their usage of cloud services, e.g., when sending and receiving emails or when interacting with mobile apps on their smartphones. However, only if users are aware of (the extent of) their exposure to cloud services, they can make informed decisions and exercise their right to privacy. As the first contribution of this dissertation, we present approaches to provide users with transparency over their individual exposure to cloud services along two deployment domains for cloud services even less technically proficient users interact with on a daily basis.

MailAnalyzer, which we present in Section 3.2, targets the privacy risks of cloud-based email, especially when the use of cloud resources is hidden from users. To this end, we analyze header information of actually exchanged emails to detect cloud services that have been hit on the path from the sender to the receiver of an email. We use our approach to study 31 million emails, ranging from public mailing list archives to the personal emails of 20 users. Our results show that as of today, 13 % to 25 % of received emails are exposed to cloud services and that this exposure is often unobservable, especially for less technically proficient users.

CloudAnalyzer, which we present in Section 3.3, uncovers the cloud usage of mobile apps on off-the-shelf smartphones as our second deployment domain. Here, we locally monitor the network traffic produced by mobile apps running on users' devices and use observed communication patterns to detect utilized cloud services. We apply CloudAnalyzer to study the cloud exposure of 29 volunteers over the course of 19 days. In addition, we analyze the cloud usage of the 5000 most accessed mobile websites as well as the 500 most popular mobile apps from five different countries. Our results reveal an excessive exposure to cloud services: 90 % of mobile

apps use cloud services and 36 % of mobile apps used by our volunteers exclusively communicate with cloud services.

We round up our work on raising awareness for cloud usage by studying the feasibility and applicability of securely applying comparison-based privacy [ZHHW15] to nudge users on the cloud usage of their mobile apps. As a result, we enable users to compare their personal app-induced cloud exposure to that of their peers to discover potential privacy risks resulting from deviating from "normal" usage behavior.

C2: Data Handling Requirements-aware Cloud Infrastructure

Most data that is outsourced to the cloud has data handling requirements, such as storage location and duration, often imposed by law or other regulations. Our core idea to support infrastructure providers in offering support for these requirements is to let users annotate data accordingly before it is sent to the cloud. There, these annotations can then be used by the infrastructure provider to select storage nodes.

As a foundation for making cloud infrastructure data handling requirements-aware, we present *CPPL*, a compact privacy policy language, in Section 4.2. CPPL enables users to express their data handling requirements and then compresses resulting privacy policies by taking advantage of flexibly specifiable domain knowledge. Our evaluation shows that CPPL reduces policy sizes by two orders of magnitude compared to related work. We employ CPPL to realize highly privacy-relevant use cases in the context of the cloud-based IoT and cloud-enabled big data to further prove the large-scale feasibility of our approach.

To comply with expressed data handling requirements in cloud storage systems, we propose *PRADA* in Section 4.3. PRADA introduces a transparent data handling layer on top of commodity cloud storage systems, which empowers users to impose data handling requirements and enables providers of cloud storage systems to comply with these requirements. We implement PRADA on top of the distributed database Cassandra and show in our evaluation that complying with data handling requirements in cloud storage systems is practical in real-world deployments such as microblogging and distributed storage of email. In combination, these two approaches that form our second contribution overcome the communication and implementation of data handling requirements as a major adoption barrier of cloud computing for both corporate and private users.

C3: Privacy-preserving Cloud Services for the Internet of Things

Providers of cloud services have to adhere to various privacy regulations. However, since service providers cannot influence the underlying cloud infrastructure, accounting for privacy regulations is an extremely challenging task, especially for non-security experts. To illustrate how privacy-preserving cloud services can be realized on top of commodity cloud infrastructure, we select a platform for globally interconnected Internet of Things (IoT) devices as a use case, as the IoT requires especially strong privacy protection. Here, we address privacy challenges arising from managing data as well as devices and networks centrally in the cloud.

Based on a security architecture for IoT data in the cloud [HHCW12, HHM+13, HHMW14], we present *SCSlib* in Section 5.2. SCSlib is a security library that transparently handles all security functionality that is required to access protected IoT *data* in a user-centric and cryptographically enforced access control system. We thus enable domain specialists who are not security experts to realize privacy-preserving cloud services. As our evaluation shows, processing protected IoT data in a cloud service is feasible. Furthermore, SCSlib's caching scheme considerably improves processing time compared to a naïve implementation of security mechanisms.

To put users back in control over their IoT *devices and networks* when these are managed centrally in the cloud, we propose *D-CAM*, a distributed approach to configuration, authorization, and management of IoT devices and networks, in Section 5.3. With D-CAM, we provide strong security guarantees by reducing the cloud to a highly available and scalable store for control messages which realize configuration of individual IoT devices, authorization of access to these devices, and management of IoT networks. Our evaluation confirms that D-CAM adds only modest overheads and easily scales to large IoT networks. In summary, our third contribution empowers non-security experts to develop privacy-preserving cloud services.

C4: Decentralizing Individual Cloud Services

Finally, we acknowledge that—besides all our efforts—some users might have such strong privacy expectations and mistrust into cloud providers that they would prefer to completely refrain from using cloud services. Furthermore, not all types of cloud services, most notably individual services such as calendar and contact synchronization, require the massive scalability of the cloud. Hence, we strive for a different, arguably quite radical approach to delivering the remaining advantages of cloud computing such as availability and reliability for this class of services. With *PriverCloud* we present a secure peer-to-peer cloud platform in Section 6.2. PriverCloud utilizes idle resources of devices operated by users' close friends and family to realize a trusted, decentralized system in which cloud services can be operated in a secure manner. Our evaluation shows that commodity computing resources can indeed be utilized to securely run existing cloud applications in a decentralized system. By breaking up the inherent centrality of cloud computing, we enable even extremely privacy-cautious users to benefit from the advantages of cloud computing.

1.3.1 Interplay of Contributions

In the context of this dissertation, we consider two different cooperation scenarios. Within the scope of Contributions C1 to C3, we realize cooperation between different actors in the cloud computing landscape, which requires a certain level of trust into infrastructure and service providers. Contrary, Contribution C4 relies on cooperation solely between users to eliminate any trust assumptions for cloud providers. In the following, we discuss how the four contributions presented in this dissertation address the identified core privacy problems (Section 1.1) and our research questions (Section 1.2) in more detail. Subsequently, we highlight the relationship and interplay of our contributions.

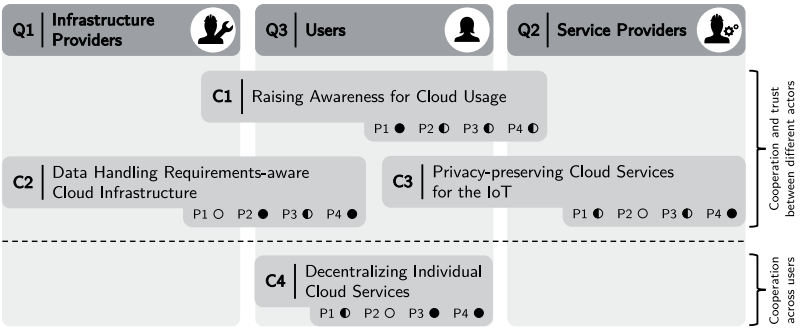

Figure 1.3 Our contributions address the underlying research questions and the core problems to privacy in cloud computing: technical complexity and missing transparency (P1), opaque legislation (P2), inherent centrality (P3), and missing control (P4). Each contribution fully addresses (●), partially addresses (◐), or does not address (○) one of the identified problems.

As shown in Figure 1.3, *Contribution C1* addresses the question of how users can preserve their privacy when using cloud services (Q3) by raising awareness for cloud usage. Providing users with information on their cloud exposure, this contribution mainly addresses the problem of technical complexity and missing transparency (P1). Still, we also raise users' awareness for the problems resulting from opaque legislation (P2) and inherent centrality (P3). The information provided by Contribution C1 serves as a foundation to overcome the problem of missing control (P4).

While focusing on enabling infrastructure providers to support service providers and users in executing control over privacy (Q1), *Contribution C2* also touches the question of how cloud users can preserve their privacy (Q3). By providing users with a mechanism to specify privacy requirements and using these to select complying storage nodes, this contribution puts users back in control over their data (P4) and addresses the problem of opaque legislation (P2). Finally, our privacy policy language can also be used to select cloud providers based on privacy requirements, paving the way towards breaking up the centrality of cloud computing (P3).

With the goal to support service providers in building privacy-preserving cloud services (Q2), *Contribution C3* also assists cloud users in preserving their privacy (Q3). By cryptographically protecting access to IoT data and the configuration of IoT networks, this contribution puts users back in control over their privacy (P4). Consequently, we provide users with transparency over who can access their data and control their networks (P1). Finally, by providing interoperability with different cloud services, we ease the migration away from a centralized cloud landscape (P3).

Proposing a disruptive approach, *Contribution C4* focuses on supporting users in preserving their privacy (Q3), mainly by breaking up the centrality of cloud computing (P3) and thus putting users back in control over their privacy (P4). To this end, this contribution enables users to move privacy-sensitive cloud services from cloud infrastructure to a decentralized system solely consisting of trusted infrastructure. In this process, we provide users with transparency over access to their data (P1).

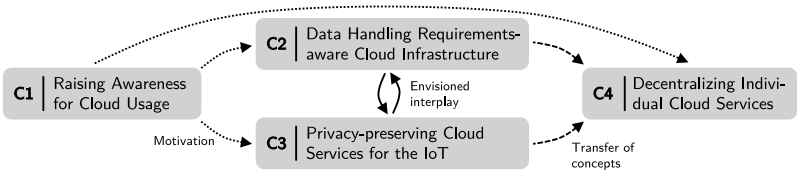

Figure 1.4 Contribution C1 raises awareness for cloud usage and hence motivates the other three contributions. Contributions C2 and C3 can be used in combination. Furthermore, concepts developed for Contributions C2 and C3 can also be applied to Contribution C4.

Regarding the relationships and interdependencies of our contributions, Figure 1.4 highlights the interplay between the individual contributions of this dissertation. MailAnalyzer and CloudAnalyzer (C1) educate users about the need for enforcing their privacy. These approaches make the necessity to account for privacy when using cloud services evident to users and thus motivate the need for Contributions C2, C3, and C4. Contributions C2 and C3 focus on infrastructure respectively service providers and are hence complementary to each other. Indeed, we envision privacy-preserving cloud services (C3) to interface with a data handling requirements-aware cloud infrastructure (C2) to pass privacy requirements down the cloud stack, where they could then, e.g., be considered when allocating storage resources. Likewise, CPPL, our privacy policy language (C2) could be used to specify requirements, such as the security level of cryptographic primitives, that would then be used by cloud services built on top of SCSlib (C3). Finally, the concepts developed in Contributions C2 and C3 can be transferred to our secure peer-to-peer cloud platform PriverCloud (C4). More specifically, the transparent data handling layer that we propose for PRADA (C2) could be equally beneficial to select storage and processing nodes based on privacy requirements in PriverCloud (C4). Similarly, D-CAM, our distributed control approach (C3) could ease the secure management of devices in a PriverCloud deployment (C4).

The contributions presented in this dissertation nicely motivate and complement each other. By combining them, and thus incorporating all actors in the cloud computing landscape, we can make an important step forward towards more privacy-friendly cloud computing.

1.3.2 Attribution of Contributions

Most parts of the contributions that we present in this dissertation have been developed in collaboration with students in the context of their Bachelor's or Master's theses, student assistant positions, or research internships. The resulting publications that form the foundation for most parts of this dissertation were created with the support of the respective co-authors of these publications. If not explicitly stated otherwise, the author of this dissertation is responsible for the initial ideas and concepts, the derived solution designs, the conceptualization of performed evaluations and measurements, as well as the final publication of results. In the following, we

briefly attribute the individual involvement of the respective students and co-authors to our contributions and the resulting publications.

Contribution C1 (Chapter 3) consists of three parts. The initial feasibility of the approach underlying *MailAnalyzer* (Section 3.2) has been studied by Mary Peyton Sanford during her UROP research internship [San16b]. For the subsequent publication of our results [HSH17], Oliver Hohlfeld contributed the active measurements, while the author of this dissertation reimplemented the approach, performed the passive measurements, and conducted the evaluation. An initial description of the idea underlying *CloudAnalyzer* (Section 3.3) has been published together with our collaborators in the TRINICS project [HKH+16]. Erik Mühmer implemented the core of CloudAnalyzer's functionality within his Bachelor's thesis [Müh14], David Hellmanns integrated CloudAnalyzer into Android as part of his Bachelor's thesis [Hel15], and Arthur Drichel realized the framework for the large-scale evaluation of mobile apps using CloudAnalyzer in his Bachelor's thesis [Dri16]. Student assistants Erik Mühmer and Jan Pennekamp subsequently further improved the implementation of CloudAnalyzer on Android. For the publication of our results [HPH+17], David Hellmanns and Jan Pennekamp set up the infrastructure for performing and evaluating the user study, Torsten Zimmermann contributed measurements of mobile websites, and Arthur Drichel contributed to the large-scale evaluation of popular mobile apps. The concept of *comparison-based privacy* used to nudge users on the cloud usage of their mobile apps (Section 3.4) has initially been proposed by Jan Henrik Ziegeldorf [ZHHW15] and was implemented by Patrick Marx in the context of his Master's thesis [Mar16]. The author of this dissertation adapted the security design to the requirements of studying cloud usage and Ritsuma Inaba prototypically implemented this approach during his UROP research internship [Ina17]. For our publication of first results [HIFZ17], student assistant Ina Berenice Fink revised the implementation and helped in performing the evaluation.

The abstract idea of Contribution C2 (Chapter 4) was first motivated [HHW13a] and later concretized [HGKW13] based on initial experiments performed in the context of the Bachelor's theses of Marcel Großfengels [Gro13] and Maik Koprowski [Kop13]. The design of *CPPL* (Section 4.2) evolved through numerous discussions with Jens Hiller and was implemented by Sascha Schmerling over the course of his Master's thesis [Sch15]. For the publication of CPPL [HHS+16], Jens Hiller contributed the analysis of related work, executed most of the evaluation, and developed the example presented in Appendix A.1. The design of *PRADA* (Section 4.3) was implemented on top of Cassandra by Johannes van der Giet in the scope of his Master's thesis [Gie14]. Student assistant Erik Mühmer subsequently improved and extended the implementation. Annika Seufert simulatively evaluated different load balancing schemes in her Bachelor's thesis [Seu15]. The author of this dissertation reimplemented the simulator and evaluated the influence of PRADA on load balancing. For the publication of our approach [HMH+17, HMH+18], Roman Matzutt set up the evaluation cluster, Erik Mühmer and Roman Matzutt executed the performance evaluation, and Jens Hiller contributed to the design of failure recovery.

The underlying motivation for Contribution C3 (Chapter 5) and corresponding background information (Section 2.4) have been published in cooperation with our collaborators in the IPACS and SensorCloud projects [HHK+14, EHH+14, HHK+16].

The security architecture for IoT data in the cloud that serves as foundation for this contribution (Section 5.2.2.2) was jointly designed by René Hummen and the author of this dissertation with the help of Daniel Catrein [HHCW12, HHM+13, HHMW14, HHMW16]. In the context of their Bachelor's theses, Roman Matzutt [Mat13] and Marc Seebold [See13] contributed to the initial implementation of this security architecture, an effort that was later continued by student assistants Benjamin Assadsolimani, Dominik Chmiel, Theo Dreßen, and Roman Matzutt. The design of *SCSlib* (Section 5.2) was mainly implemented within the scope of the Bachelor's thesis of Sebastian Bereda [Ber14], minor aspects with respect to access control were derived from the Bachelor's thesis of Aivar Kripsaar [Kri14]. For the publication of SCSlib [HBHW14], Sebastian Bereda performed most practical aspects of the evaluation. Our *D-CAM* approach (Section 5.3) was primarily implemented by Benedikt Wolters as part of his Bachelor's thesis [Wol14], minor aspects regarding efficient signature schemes were derived from the Master's thesis of Devran Ölcer [Ölc13]. In the context of our publication of D-CAM [HWM+17], Roman Matzutt and Benedikt Wolters jointly performed the evaluation.

The design of *PriverCloud*, our Contribution C4 (Chapter 6), was jointly developed by Jens Hiller and the author of this dissertation. Jens Hiller implemented and evaluated *PriverCloud* within the scope of his Master's thesis [Hil14]. Subsequently, Fritz Alder experimented with distributing the storage architecture of PriverCloud in the context of his Bachelor's thesis [Ald15]. We presented the motivation and design decisions underlying PriverCloud in a publication [HHHW16].

1.4 Outline

This dissertation is structured as follows. In Chapter 2, we provide the foundation for our work by introducing the cloud computing paradigm, discussing resulting privacy challenges, and introducing the concept of the cloud-based IoT which we use in selected parts of this dissertation to highlight distinct privacy challenges. Chapter 3 presents our first contribution that raises users' awareness for cloud usage along the two application domains cloud-based email and mobile cloud computing. Our results inform users about the necessity of considering privacy when using cloud services and hence provide the motivation for our remaining contributions described in Chapters 4 to 6: In Chapter 4, we describe our two approaches, a compact privacy policy language and a data handling requirements-aware cloud storage system, that jointly realize data handling requirements-aware cloud infrastructure. Chapter 5 presents our contribution to provide privacy-preserving cloud services for our application domain, the IoT, that consists of a security library that transparently handles security functionality on behalf of cloud services and a distributed approach to handle configuration, authorization, and management of devices and networks in the cloud-based IoT. Finally, we present our contribution to decentralize individual cloud services by shifting them to a peer-to-peer network over trusted infrastructure in Chapter 6. We conclude this dissertation with a summary of our contributions and insights as well as a discussion of future research challenges in Chapter 7.

2

Privacy in Cloud Computing

As the foundation for our contributions presented in this dissertation, we first provide an introduction into those topics relevant to understanding the concepts described in the remainder of this dissertation. To this end, we begin with a description of the cloud computing paradigm, its characteristics, service and deployment models, as well as its relevant actors (Section 2.1). Subsequently, we turn our view to defining privacy in the cloud computing context, derive different types of personal information that require protection, and describe the differences and similarities of privacy and security (Section 2.2). Based on this definition of privacy, we discuss the distinct privacy challenges of cloud computing, especially data handling requirements and legal obligations, resulting attack models, as well as key principles for designing and implementing privacy-preserving cloud services (Section 2.3). We introduce the concept of the cloud-based Internet of Things, which we use as an application domain in selected parts of this dissertation (Section 2.4), before we conclude this chapter with a brief summary (Section 2.5). During our description of the individual topics, we mainly focus on those aspects that are particularly relevant for the scope of this dissertation, i.e., the interaction and relationship of the individual stakeholders in the cloud computing landscape and resulting privacy challenges.

2.1 The Cloud Computing Paradigm

While there are many anecdotes on the history and emergence of the cloud computing paradigm, the underlying idea of time-sharing of computing resources dates back to the 1970s [Whi71, Pul15]. Yet, the perception of cloud computing as we know it today has arguably been most influenced by the launch of Amazon's Elastic Compute Cloud in 2006 [Mil16]. From the early 2000s, Amazon had already worked on concepts that later emerged into what is today known as Amazon Web Services (AWS) [Mil16], mainly to solve problems Amazon faced when deploying their own systems.

Given its emergence from the needs of companies and its rather young age of only ten years, there is no single accepted definition of the term "cloud computing". Hence, our presentation of the cloud computing paradigm in the following mainly combines the "Berkeley view of cloud computing" [AFG+09, AFG+10], the guidelines of the US-American National Institute of Standards and Technology (NIST) [MG11, LTM+11], the vocabulary standardized in ISO/IEC 17788 [ISO14], and the description from the textbook of Erl et al. [EMP13] to derive a broad picture of what makes up cloud computing.

2.1.1 Characteristics of Cloud Computing

The most widely accepted definition of cloud computing has been proposed by NIST, which considers "cloud computing [as] a model for enabling ubiquitous, convenient, on demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction" [MG11]. From this brief definition, NIST derived five essential characteristics of cloud computing [MG11], that have later been extended with a sixth characteristic by Erl et al. [EMP13]. We summarize these characteristics in the following and illustrate them with examples where appropriate.

On Demand Self-service and Usage. This characteristic enables cloud users to provision cloud resources (such as computing or storage) *themselves* as required—without the need for human interaction with each individual provider of cloud resources. More specifically, cloud users can unilaterally request resources whenever they require them and providers of cloud resources will automatically deploy these resources as requested. For example, cloud providers such as Amazon or Microsoft offer management consoles and application programming interfaces (APIs) to automatically create, configure, manage, and terminate virtual machines (besides other resources). When a user requests a new virtual machine, the cloud provider will typically deploy the requested resource within seconds to minutes [MH12].

Independence from Device and Location. Cloud computing has to ensure that its deployed resources are widely accessible, i.e., from a large range of devices and locations, a property often referred to as ubiquitous or broad network access. To this end, cloud computing mandates the use of standardized protocols and interfaces to access resources. These measures ease the integration of a heterogeneous set of devices—ranging from server-grade computers and desktop deployments over smartphones to embedded devices in the IoT and CPS. For example, file synchronization services, such as Dropbox or Google Drive, enable users to access their files on virtually any (Internet-capable) device from any location worldwide.

Resource Pooling and Multi-tenancy. In cloud computing, resources (such as processing, storage, memory, and network bandwidth) are shared or pooled between different users (referred to as multi-tenancy). To this end, resources are dynamically assigned and reassigned based on the current demand of the customers of a cloud provider. This handling of resources enables cloud providers to significantly

increase the utilization of their servers beyond the 5 % to 20 % estimated for traditional data centers [AFG+10]. Resource pooling typically is oblivious to the users of cloud resources, i.e., they typically remain unaware of the fact that other users are (currently) using the same resources. One key aspect of resource pooling (that highly influences privacy) is that users have little influence of controlling properties of deployed resources (e.g., the exact location). In the best case, users can control resource properties at a coarse granularity, e.g., resources can often be requested in a specific so-called availability region or zone that groups different data centers which are in close proximity and typically in the same jurisdiction.

Rapid Elasticity. As one of the key advantages of cloud computing, users can automatically scale up and down the cloud resources they use to adapt to varying load demands. Typically, resources (such as virtual machines) can be requested and released in a timely manner (in the order of minutes) and at a fine granularity (e.g., one virtual machine at a time) [AFG+10]. For example, Animoto, a cloud service that turns user-uploaded images into music videos, was able to scale from 50 to 3400 Amazon EC2 instances (virtual machines) within only three days to keep up with sudden user demand [Bar08].

Scaling computing resources up (and later down) by nearly two orders of magnitude within days would have been impossible with traditional data centers [Bar08]. From the users' perspective, the elasticity of cloud resources often creates an impression of unlimited scalability. We refer to this phenomenon as "virtually unlimited resources" in the remainder of this dissertation.

Measured Service and Usage. All usage of cloud resources (such as storage space, processing time, network bandwidth, and number of user accounts) is typically measured at a certain level of abstraction (e.g., CPU hours). Based on the measured usage, cloud users are billed, often on a pay-per-use basis. As such, cloud users are only charged for the period of time and amount of consumed resources. For example, as of June 2018, Amazon charges between $0.0058 (general purpose `t2.nano` instance with 1 virtual CPU and 0.5 GB RAM) and $26.688 (RAM optimized `x1e.32xlarge` instance with 128 virtual CPUs and 3904 GB RAM) per hour for its Amazon EC2 on demand instances in its "US East" region [AWS18b]. These costs can be reduced significantly, e.g., by using longer-term contracts or by bidding on spare capacity.

Notably, measuring service usage is relevant beyond billing purposes as it provides users and providers with transparency on used resources as part of the general monitoring of IT resources. Hence, even cloud services that can be used free of charge typically measure the usage of resources. For example, file synchronization services, such as Dropbox or Google Drive, measure storage space to enforce quotas of their free to use offers.

Failover and Resilience. To guarantee availability and reliability of resources even in the face of outages or systems failures, failover and resilience mechanisms of cloud computing replicate these resources across different locations. When a defect of one resource instance is detected, all requests for this resource will automatically be served by one of the replicas. For example, the distributed cloud storage system Cassandra allows to replicate stored data in the same data center (but not in the

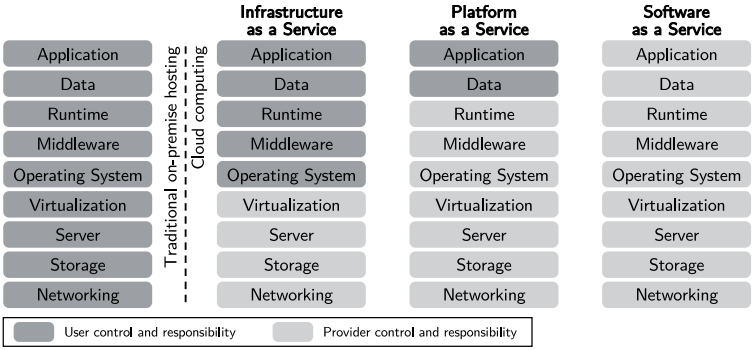

Figure 2.1 The different service models of cloud computing offer an increasing level of abstraction (especially in contrast to traditional on-premise hosting) and hence also shift more control and responsibilities from users to cloud providers (figure adapted from Chou [Cho10]).

same rack) or in a different data center [LM10]. Replication to different, often physically remote, data centers limits the impact of complete data center failures, such as the storm-related six-hour outage of a complete Amazon data center in June 2012 that impacted popular services, e.g., Netflix, Pinterest, and Instagram [McM12]. Furthermore, the intercloud paradigm even proposes to replicate resources across different cloud providers [BRC10], e.g., to protect against system failures resulting from programming or configuration errors such as the typing error causing a nearly six-hour outage of Amazon's storage service in February 2017 that impacted major parts of the Internet [Kin17].

To additionally differentiate cloud computing from the earlier concepts of grid and cluster computing, Buyya et al. [BYV+09] propose characteristics such as the type of physical computers, the size of systems, and application scheduling strategies. However, in the context of this dissertation, it is instead important to study the different service and deployment models of cloud computing to identify the relevant aspects and actors with respect to privacy.

2.1.2 Service and Deployment Models of Cloud Computing

While delivering the different characteristics of cloud computing, different service and deployment models define at which granularity resources can be consumed and who can gain access to these resources. In the following, we present and discuss these service and deployment models based on the definition of NIST [MG11] and the insights of Erl et al. [EMP13].

2.1.2.1 Service Models

Essentially, the different service models of cloud computing as shown in Figure 2.1 define how much control users have over the provided stack of resources. Conse-

quently, more control also implies more responsibility for the underlying technical realization and hence also for taking care of privacy protection measures. In traditional on-premise deployments (left side of Figure 2.1), users have control over and are responsible for the complete technology stack. In contrast, NIST defines three service models for cloud computing [MG11] that offer access to resources at different levels of abstraction (right side of Figure 2.1). We discuss these three service models in increasing level of abstraction in the following.

Infrastructure as a Service (IaaS). As the foundation of the cloud computing paradigm, the Infrastructure as a Service (IaaS) service model provides users with fundamental computing resources such as processing, storage, networking, and load balancing. While users of IaaS services lack direct control over the underlying (hardware) infrastructure, they have full control over the operating system, block and file-based storage facilities, and can deploy any desired software packages. Resources provided by IaaS are typically not pre-configured to allow for a high level of customization. This flexibility, however, places large parts of the administrative burden on the users of IaaS. From a technical perspective, IaaS is predominantly realized using virtual machines. With respect to networking resources, users of IaaS typically have limited control over selected networking functionality such as firewalls by utilizing dedicated high-level APIs. Examples for IaaS include the computing resources (i.e., virtual machines) offered by Amazon Elastic Compute Cloud (EC2), Microsoft Azure, Google Compute Engine, and Rackspace Cloud Servers.

Platform as a Service (PaaS). With the goal to ease the deployment of self-developed applications in the cloud, Platform as a Service (PaaS) services provide their users (i.e., application developers) with a fitting development infrastructure and environment. Hence, PaaS can be considered a combination of a software development kit (SDK) and a corresponding execution environment (including web server and storage) that runs massively distributed in the cloud. Most notably, as a result of the abstraction provided by PaaS users lack control over the cloud infrastructure (such as operating system, storage, or networking). Consequently, they are also not responsible to set up and administer the underlying infrastructure and its composition into a highly scalable system—a fact that is often considered as a major advantage of PaaS over IaaS. Still, users of PaaS are in control over the executed software (predominantly developed by themselves) and typically can configure or parameterize (to a limited extent) the execution environment used to run their software. Notable examples for the diverse range of PaaS services are AWS Elastic Beanstalk, Google App Engine, Heroku, and Force.com. These are then used by service developers as a foundation for realizing applications for end users.

Software as a Service (SaaS). Providing the highest level of abstraction, Software as a Service (SaaS) services provide users with access to applications that run in the cloud. Such applications can be accessed by a wide range of devices such as desktop deployments, smartphones, or embedded IoT devices and CPS over a web interface or a dedicated API. Typically, SaaS is used to provide a cloud service (following commercial interests) to a large group of (potential) users. In this setting, users do neither control the underlying cloud infrastructure nor the execution environment and individual services. Yet, users of SaaS services might have (limited) possibilities to configure or parameterize a service they use. From the perspective of users,

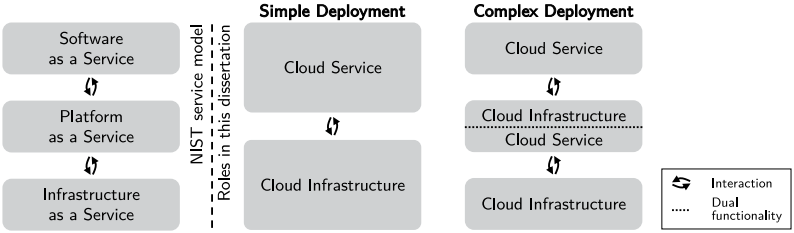

Figure 2.2 Deviating from the NIST service model with IaaS, PaaS, and SaaS, in this dissertation, we differentiate cloud offers by their role into cloud infrastructure and cloud service. In a complex deployment, one cloud offer can act as both, cloud service and cloud infrastructure.

interacting with a SaaS service is the same as interacting with any other web service, since the abstraction offered by the SaaS service model effectively hides that a specific application or service is run in the cloud. Examples for popular SaaS services include Google's G Suite (including Gmail, Google Docs, and Google Drive), Microsoft Office 365, Dropbox, and Slack.

Each of these three service models can be delivered by a different provider [AKK12], leading to an indirect usage of cloud resources where cloud providers often subcontract other cloud providers [PP15]. While these three service models are often visualized (and interpreted) as a layered stack [MG11, AKK12]—similar to the OSI model for communication systems [Zim80]—this is not implied by the cloud computing paradigm per se. For example, a SaaS service could also be directly realized on top of physical hardware instead of relying on the abstraction provided by IaaS and PaaS. Similar to performance optimizations in networking stacks by (partially) omitting layers [AHA+14], service providers of larger SaaS services strive to increase the performance of their services by avoiding the use of IaaS and PaaS. For example, FreeAgent, a UK-based online accounting SaaS provider, migrated from virtual machines deployed at different IaaS providers to colocation hosting to increase performance, reduce costs, and strengthen reliability [Hea17].

While the three service models of cloud computing, IaaS, PaaS, and SaaS, initially seem to be well-defined and the mapping of cloud offers to one of the service models straightforward, we—similar to Armbrust et al. [AFG+10]—argue that there is no clear line between IaaS, PaaS, and SaaS. Furthermore, from a privacy perspective, differentiating between service models should be mainly performed based on their interaction and data flows instead of vague technical boundaries. Hence, in the scope of this dissertation, we define a different view on delivering cloud services that focuses on the interaction between these services instead of the (technical) type of service they deliver. As shown in Figure 2.2 (middle), we only differentiate between *cloud infrastructure* and *cloud service*. Here, cloud infrastructure defines cloud offers that provide infrastructure to another cloud offer (predominantly IaaS and PaaS services, but can also apply to SaaS services used as a building block of another SaaS service). Likewise, cloud services use cloud infrastructure to provide a service to users and other cloud offers. This mainly applies to PaaS and SaaS in the NIST service model. Depending on the specific deployment scenario, each cloud offer

takes the role of cloud infrastructure or cloud service or both. Most importantly, a single cloud offer can act as both, cloud infrastructure and cloud service, when it uses cloud infrastructure (as a cloud service) and provides cloud infrastructure to another cloud offer at the same time (right part of Figure 2.2). By modeling roles of cloud offers using this model, we provide more flexibility than the NIST service model as we also support more complicated deployments such as a technology stack of four or even more cloud offers realized on top of each other.

2.1.2.2 Deployment Models

Orthogonal to different cloud service models, i.e., at which level of abstraction cloud services are provided, is the question how these services are deployed. More specifically, who owns and governs the cloud environment, how large the cloud environment is, and who can gain access to it. To classify different types of deploying cloud services, NIST defines the following four cloud deployment models [MG11].

Public Cloud. In a public cloud deployment, the cloud environment is provisioned by a third party cloud provider (corporate, academic, or government organization) over the Internet and in general is publicly accessible to anyone. Services and infrastructure provided in a public cloud are typically offered for a fee (especially for infrastructure) or are commercialized using other means, such as (targeted) advertisement [EMP13] or monetizing user profiles (especially for services). A public cloud is realized on the premises of the respective cloud provider, which is also responsible for setting up and maintaining the deployed services and resources. The public cloud deployment model is the predominant and most widely known deployment model for cloud services and all examples we presented for the different service models are realized as public clouds deployments. Hence, the largest cloud deployments rely on the public cloud model, which makes it the main focus of this dissertation.

Community Cloud. While similar to public clouds from a technical perspective, the notable difference from an organizational perspective is that access to service and resources in a community cloud is restricted to a specified group of users, called community. Typically, such communities have a common denominator such as shared concerns regarding security, policy, and compliance or specific performance and availability requirements. Users from outside the community generally cannot access the services and resources provisioned in a community cloud. A community cloud can either be operated and managed by one or more of the community members or provided by a third party. Often, community clouds are smaller in size compared to public clouds and raise fewer privacy concerns. Examples of community clouds include the ENX network of European vehicle manufacturers and the Sciebo storage cloud operated by universities and research institutions in the German state of North Rhine-Westphalia.

Private Cloud. In contrast to public and community clouds, a private cloud is exclusively used by a single organization. Hence, all users of the private cloud, e.g., different business units, are part of the corresponding organization. Private clouds are an attempt to benefit from the advantages of cloud computing such as flexibility and scalability, without having to give up control over the deployment of and access

to resources. In fact, private clouds a typically realized in own, on-premise or in traditional colocation data centers. Given the required scale for such deployments (and the resulting upfront investment), private clouds are predominantly utilized by large enterprises while small and medium-sized enterprises (SMEs) shy away from the resulting management and cost overheads. In private clouds, most of the privacy problems and concerns discussed in this dissertation do not apply. Organizations that revealed that they are relying on privates clouds include The Hartford [IBM14] and the Volkswagen Group [Plu17].

Hybrid Cloud. Finally, a hybrid cloud deployment combines at least two of the other deployment models (public, community, or private). Notably, the deployments that make up the hybrid cloud still remain independent and are often operated by different providers. Creating and managing hybrid cloud deployments is often a challenging and complex task because of differences between deployments, lack of standardized interfaces, and multiple providers involved. Motivations for a hybrid cloud deployment include keeping private data in-house while combining it with cloud services run in a public cloud or using public cloud infrastructure to handle temporary spikes in demanded capacity. For larger enterprises, combining public and private cloud in a hybrid cloud deployment model hence can offer one approach to benefit from the cloud advantages while catering for privacy and compliance requirements, e.g., by only outsourcing non-critical data to a public cloud. One example for a hybrid cloud deployment is the online accounting service FreeAgent, which operates its computing resources in a private cloud but still relies on public cloud services to provide storage and domain name system (DNS) [Hea17].

Out of these four deployment models, the public cloud deployment model is the most popular and widely used—especially by private and (smaller) corporate users that lack the resources to operate an own private cloud or participate in a community cloud. Likewise, the public cloud deployment model certainly is the most challenging one with respect to our goal of accounting for privacy. Thus, we focus on providing privacy in public cloud deployments in this dissertation and if not stated explicitly otherwise, use the term “cloud” synonymously for “public cloud” in the following.

2.1.3 Actors in the Cloud Computing Landscape

The different service and deployment models of cloud computing make evident that the cloud computing paradigm leads to more actors that are involved in delivering a (web) service compared to the client-server model prevalent on the Internet so far [Han00]. As part of its cloud computing reference architecture, NIST defines five major actors [LTM+11]: cloud consumer, cloud provider, cloud auditor, cloud broker, and cloud carrier. In the scope of this dissertation, we evolve this reference architecture with respect to the different actors to better cater for different responsibilities with respect to privacy. Most notably, we split NIST’s cloud provider into separate infrastructure and service providers, sharpen the definition of cloud consumer (user in our model), and add the role of a legislator.

In the following, we detail how the privacy-centric actor model derived for this dissertation (briefly introduced in Section 1.1.1) integrates into NIST’s cloud computing

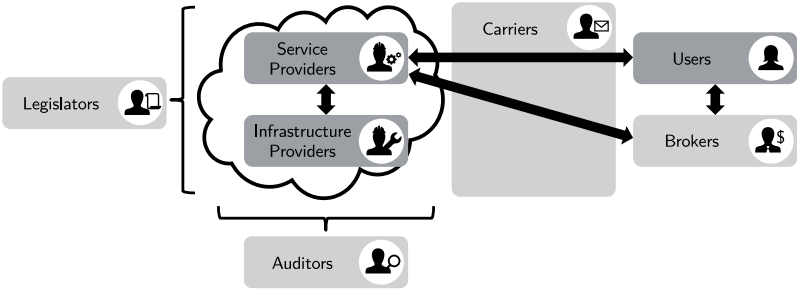

Figure 2.3 The cloud computing landscape consists of different actors and their relationships and interplay. We mark the three actors that are most important for privacy (from a technical perspective) and are hence in the focus of this dissertation in dark gray.

reference architecture. Figure 2.3 introduces the different actors and their relationships. Actors highlighted in dark gray play a major role in (technically) accounting for privacy in the cloud computing landscape and are thus in our focus.

Cloud Providers. To form the technical foundation for cloud computing, cloud providers make cloud offers available to all interested parties [LTM+11]. Notably, cloud offers are delivered at different layers of abstraction. To account for these different layers and hence the interaction and relationships of different cloud providers and resulting privacy challenges, we—in contrast to traditional actor models for cloud computing [LTM+11]—introduce a clear distinction between two different types of cloud providers: infrastructure providers and service providers. In this model, *infrastructure providers* deliver infrastructure to other cloud providers and hence mainly realize IaaS and PaaS, but also SaaS can be provided as infrastructure to other SaaS services. The infrastructure deployed by infrastructure providers consists of the (physical) resources required for operating cloud services, i.e., computing and storage resources as well as broadband network connectivity. With increasing level of abstraction, these resources also become more abstract, e.g., in the form of virtual machines and virtual network interfaces, distributed file systems and databases, as well as runtime environments and execution stacks. Using cloud infrastructure deployed by infrastructure providers, *service providers* realize cloud services that mostly consist of PaaS and SaaS in the NIST definition of cloud services [MG11]. Such cloud services either target private and corporate users or serve as a foundation for other cloud services (cf. Figure 2.2) and can usually be accessed over the Internet. Service providers typically rely on APIs and/or SDKs offered by infrastructure providers to realize their services. Cloud providers (both at the infrastructure and service level) play a vital role in accounting for privacy in the cloud computing landscape as they control the physical, technical, and organizational realization of cloud services. We propose technical mechanisms that infrastructure and service providers can deploy to account for privacy in Chapters 4 and 5, respectively.

Users. Cloud users are private and corporate actors which utilize cloud services that are deployed by service providers. Typically, cloud users and cloud providers

agree on some sort of business relationship or contract [LTM+11], irrespective of whether the service provider charges a fee for using its service or not. Cloud users rely on standardized interfaces and protocols to access cloud services. These interfaces and protocols range from traditional Internet protocols over web interfaces to dedicated APIs. While users typically only directly interact (and hold business relationships) with service providers, they are indirectly also exposed to the infrastructure providers that realize the foundation for their utilized cloud services. This indirect usage cannot be controlled by users nowadays and often occurs obliviously, especially for less technically proficient users. Yet, users are those actors in the cloud computing landscape that are impacted most with respect to their privacy and arguably often the weakest link. To make users aware of their impacted privacy and to put them back in control, we provide technical approaches that can be deployed by users to account for their own privacy in Chapters 3 and 6.

Legislators. Providing the underlying legal frameworks, legislators impose restrictions on service and infrastructure providers on how they can deploy their cloud offers. Likewise, also users are affected by legislation. While private users are mostly protected by legislation, e.g., through data protection laws, corporate users are often hindered from using cloud services due to various legislation. When focusing on privacy, we are mostly concerned with data protection legislation, but also other compliance concerns have to be considered. Furthermore, legislators often also have own (financial) interests and might impose laws and regulations not only for the greater good but also to strengthen their own economy, e.g., with regional cloud offers [SBC+14, HMR+14]. Within the technical scope of this dissertation, we cover legislators only when they directly influence technical decisions and consider other aspects such as policy issues out of scope. From our technical perspective, we support legislators by providing transparency over privacy problems (Chapter 3) and by introducing technical means to comply with legal requirements, especially with respect to transborder data flows (Chapter 4).

Auditors. The task of auditors is to act as independent and trusted third parties to verify that cloud offers are indeed provided according to agreed-upon service level agreements (SLAs). Typically, audits are performed by reviewing objective evidence, e.g., specially crafted audit logs [SK99, SYC04, WBDS04], and hence verifying that the operations of cloud providers conform to their promises [LTM+11]. While not specific to cloud computing, auditing is especially important for cloud computing with its complex, dynamic, and often indirect trust relationships. In the context of cloud computing, auditing can, e.g., be used to verify that a cloud provider indeed stores all data and does not delete data that is rarely or never accessed to cut down costs [WWRL10]. Furthermore, audits can assess that data retention policies are adhered to, data has not been modified, and data archival requirements are met [LTM+11]. Hence, auditing concerns a wide range of aspects that can be covered by SLAs—also outside privacy and security requirements. With respect to this dissertation, auditors nicely complement our approach for data handling requirements-aware cloud infrastructure (Chapter 4) to ensure that cloud providers indeed operate our approaches as intended. Furthermore, our awareness approaches presented in Chapter 3 are a valuable tool for auditors when checking the compliance of email offers and smartphone apps with privacy requirements in the cloud computing context.

Brokers. The concept of cloud brokers aims at a scenario where the cloud computing landscape becomes too complicated for users to manage the integration and composition of different services [LSW04, LTM+11], e.g., in the envisioned move towards intercloud deployments, where users combine resources of different cloud providers in an automated fashion [GB14]. In this setting, cloud brokers act as intermediaries between users and cloud providers to provision services and resources. More specifically, cloud brokers take care of managing usage and delivery of cloud services and resources by negotiating contracts on behalf of users and cloud providers [LTM+11]. Employing cloud brokers might provide economic advantages for all involved actors, i.e., users, service providers, and infrastructure providers [GGBM15]. When assessing the impact of brokers on privacy, we find that cloud brokers can assist users in selecting cloud offers based on privacy or compliance requirements, e.g., with respect to data location or storage duration [GGBM15]. In the context of this dissertation, brokers could rely on the privacy requirements expressed using our compact privacy policy language (Section 4.2) when choosing between different cloud offers.

Carriers. From a technical perspective, cloud carriers provide connectivity and transport of data between users and cloud services (and hence also the underlying cloud infrastructure) [LTM+11]. In today's public cloud deployments, the role of cloud carriers does not notably deviate from those of the carriers involved in delivering traditional Internet services. Hence, cloud carriers do not pose specific privacy challenges in addition to those of traditional carriers on the Internet. Still, with a possible move towards intercloud deployments [GB14], users (and possibly cloud service providers) might have the option to choose between multiple cloud carriers with different properties. In such a situation, the choice between different carriers could be influenced by privacy requirements, e.g., expressed using our compact privacy policy language (Section 4.2).

To conclude, our introduction into cloud computing makes evident that the cloud computing landscape is diverse and versatile. We have identified numerous ways of interplay between the different actors—and often interactions occur indirectly and are unobservable for the actual users whose privacy is then put at stake.

In the remainder of this chapter, we take a deeper look at the privacy challenges that result from the distinct characteristics of cloud computing.

2.2 Defining Privacy in the Cloud Computing Context

Besides many advantages, cloud computing—compared to traditional deployments in data centers—also introduces additional challenges with respect to privacy. In the following, we first review different definitions of privacy and then derive a common definition that serves as foundation for the remainder of this dissertation.

The definition of the term "privacy" widely varies across different fields and often depends on the specific context [RG10, Leh14, ZGW14]. Hence, it is important to understand these different perceptions of privacy as a foundation to judge on different approaches to account for privacy and to understand which aspects individual approaches address. Different authors propose valuable surveys, taxonomies,

and classifications [Sol06, Hol07, RG10, SDX11, FWF13, Leh14]. In the following, we summarize these definitions mainly along the lines of Finn et al. [FWF13] and vom Lehn [Leh14] to derive a definition of privacy for the context of this dissertation.

Already in 1890, Warren and Brandeis [WB90] formulated the "right to be let alone" as a response to the emergence of instantaneous photographs that were taken without prior consent—which was considered a serious invasion of individual privacy by Warren and Brandeis [RG10]. As a consequence, they expressed the need for establishing a right to privacy in law [Leh14]. Reacting to the emergence of computers, Westin in 1967 proposed to define privacy as "the claim of individuals, groups, or institutions to determine for themselves when, how, and to what extent information about them is communicated to others" [Wes67], a concept that we nowadays refer to as *information privacy* [Wes03]. This concept has further emerged into the OECD guidelines on the protection of privacy and transborder flows of personal data from 1980 [OECD80], which arguably constitute the first internationally agreed-upon collection of privacy principles.

Westin's definition of privacy also laid the foundation for the right to *informational self-determination*, which was established during the population census ruling of the German federal constitutional court in 1984 [HS09, Leh14]. Important key principles of this ruling, such as the concepts of data minimization and purpose specification, have been included in the EU data protection directive 95/46/EC [EU95], where they became binding for the complete EU. Recently, discussion and legislation in the EU and Argentina coined the "right to be forgotten" [Man13, Ros12] as an option for users to escape their past by having old data deleted, e.g., in public media or databases, or oppressed, e.g., from search results. This idea has been concretized as the "right to erasure" in the EU's new GDPR [GDPR16].

To fully embrace the scope of privacy, it is important to clearly identify different categories of privacy. As a first step in this direction, Solove [Sol06] and Kasper [Kas05] take a reactive approach to classifying privacy into different categories by studying different ways of how privacy can be breached [FWF13]. Solove [Sol06] provides a taxonomy to understand privacy breaches consisting of four categories: (i) *information collection* through surveillance and interrogation; (ii) *information processing* where breaches range from aggregation and identification over insecurity and secondary use to exclusion; (iii) *information dissemination* caused by breach of confidentiality, disclosure, exposure, increased accessibility, blackmail, appropriation, and distortion; and (iv) *invasion* resulting from intrusion and decisional interference.

Likewise, Kasper [Kas05] derives a typology of three privacy invasions from the different principal activities that lead to an invasion of privacy: (i) *extraction* by deliberately taking information from a person; (ii) *observation* through actively surveilling a person; and (iii) *intrusion* by directly interfering with the life of a person. As these approaches focus on classifying already occurring privacy breaches, they focus on stopping (known) harm (mostly through legislation). While this is valuable to overcome privacy challenges for existing fields, accounting for privacy in emerging technologies such as cloud computing instead requires to proactively establish technology agnostic privacy rights that prevent harm from yet unforeseen privacy risks in the first place [FWF13].

Personal information	**Personally identifiable information (PII)**	Key attributes	Name, social security number, ...
		Quasi-identifiers	Date of birth, address, IP address, ...
	Sensitive information	Membership	Political groups, religious groups, ...
		Demography	Gender, nationality, ...
		Interests and habits	Web activity, shopping history, ...
		Finance	Account balance, finan. transactions, ...
		Health	Medical records, diseases, ...
		Intellectual production	Ideas, inventions, ...

Table 2.1 Personal information can be classified into the stricter notion of personally identifiable information (PII) and the broader notion of sensitive information.

In contrast, proactively focusing on aspects of privacy that should be protected enables individuals, governments, and other organizations to evaluate the impact of their activities on users' privacy and hence develop and deploy appropriate measures to protect privacy [FWF13]. To this end, Clarke [Cla97] defines four categories of privacy: (i) *privacy of the person*—also known as bodily privacy—guarantees the integrity of a person's body, e.g., with respect to mandatory vaccination, body tissue sampling, or sterilization; (ii) *privacy of personal behavior*—also referred to as media privacy—concerns behavioral aspects ranging from political activities over religious practices to sexual orientation and preferences; (iii) *privacy of personal communication*—sometimes called interception privacy—enables persons to communicate by various means without being routinely monitored by any third party; and (iv) *privacy of personal data*—or information privacy—encompasses that data of persons is not automatically available to third parties and that persons stay in control over their data and its usage if it is in possession of any third party.

To account for recent technology advances, Finn et al. [FWF13] refine Clarke's four privacy categories and propose three additional categories: (v) *privacy of thoughts and feelings* allows persons to keep their thoughts and feelings private, especially if they do not (directly) lead to behavior; (vi) *privacy of location and space* enables persons to move around in public spaces without being tracked or monitored; and (vii) *privacy of association* allows people to freely associate with anyone they want.

These various definitions showcase the different perspectives and broad scope of definitions of privacy, leading to the necessity to focus on a specific concept of privacy. Within the technical context of this dissertation, it is especially important to focus on users' information—as this is what is ultimately transferred out of users' control. To further understand the importance and challenge of protecting users' information, we study different types of personal information in the following.

2.2.1 Types of Personal Information

As identified by Ghorbel et al. [GGJ17] and Pearson [Pea09], users' privacy generally covers different types of personal information. We provide an overview of the different categories and types of personal information together with illustrative examples in Table 2.1 and discuss them in more detail in the following.

First, *personally identifiable information (PII)* encompasses any information that can be used to identify a person [Pea09]. Here, each individual key attribute can be used to directly identify a person. Examples for key attributes include names, cell phone numbers, social security numbers, passport numbers, or email addresses [GGJ17, Pea09]. In contrast to key attributes, quasi-identifiers are a set of attributes that in combination can be used to (almost) uniquely identify a person [Swe00]. For example, combining date of birth and postal address almost uniquely identifies a person [GGJ17], while each attribute alone often does not suffice to identify a person. Hence, the notion of PII applies to all information that alone or in combination can be utilized to uniquely identify a person.

Contrary, *sensitive information* refers to the much broader field of information that can be linked to a certain person. In the following presentation of different types of sensitive information, we mainly rely on the categorization of Ghorbel et al. [GGJ17]. Yet, given the broad scope of sensitive information, this list should be considered rather as an illustration of the concept of sensitive information than as a definitive and comprehensive list. In this categorization, membership information refers to a person's affiliation with groups with respect to policy, religion, union, and community. Likewise, demography information encompasses all demographic characteristics of a person ranging from gender and nationality over level of education and professional status to potential criminal records.

Information on interests and habits deals with the activities and preferences of a person and can, e.g., be derived from web browsing activity or shopping history. Financial information consists of all aspects of a person's finances, such as bank account balance and bank account statements listing financial transactions. Health information covers data such as medical records, medical outcomes, diseases, prescriptions, and medical images. Finally, information on intellectual production refers to a person's ideas and inventions before they are made public. Besides these examples, sensitive information essentially covers any information that should remain private. When considering privacy at the level of enterprises, sensitive information, most notably, includes information on the enterprise itself as well as on its employees and customers [GGJ17].

These different types of personal information highlight that it is important but also challenging to protect this information when outsourcing it to cloud service. Hence, in the following, we derive an information-centric definition of privacy that is especially well-suited in the context of cloud computing.

2.2.2 Information Privacy in Cloud Computing

In the context of this dissertation, we further sharpen the definition of Westin's information privacy [Wes67]—centering around users and their information as foundation for informational self-determination—and evolve it to cater for the specifics of the cloud computing paradigm (inspired by Solove's taxonomy of privacy [Sol06] and the approach of Ziegeldorf et al. in the context of the IoT [ZGW14]): *Privacy in cloud computing guarantees individual users awareness and control over the collection, processing, and dissemination of their personal information.*

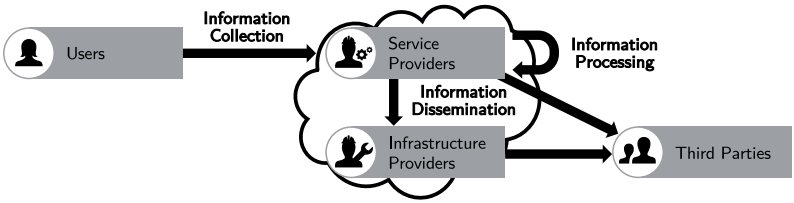

Figure 2.4 Privacy in cloud computing provides users with awareness and control over the complex and unwieldy interplay of the collection, the processing, and the dissemination (relaying) of their personal information.

We visualize this definition and especially the underlying information flows in Figure 2.4: Users' personal information is either intentionally or unintentionally collected and sent to cloud service providers. Intentional collection of personal information happens if a user willingly consumes a cloud service and provides access to her data in the context of this usage, e.g., by uploading a file to a cloud storage service. In contrast, by unintentional collection, we refer to any collection of information that is not knowingly and willingly triggered by a user. Such an unintentional collection of information can, e.g., happen through privacy-invasive smartphone applications or unobtrusive IoT devices.

After information has been collected, cloud service providers process received information either for intended or unintended functionality. Here, intended functionality refers to anything related to the service the user actually intends to use, e.g., file storage and synchronization, email and communication services, IoT backend infrastructure, but also personalization services such as Siri or Spotify. In contrast, unintended functionality covers any processing of information that is not related to the core functionality of the utilized service and encompasses, e.g., the processing of information to aid targeted advertisement. Finally, information could (unnoticeably) be disseminated from service providers to infrastructure providers and/or some third parties, e.g., government agencies.

As discussed in Section 1.1.2, the different actors in the cloud computing landscape each have an own distinctive perspective on privacy in cloud computing, mainly resulting from their differing objectives and hence motivations to cater for privacy. Hence, in this dissertation, we address these different perspectives on privacy. Since privacy is often related to security, we first discuss the similarities and differences of privacy and security to clearly set these two concepts apart.

2.2.3 Privacy vs. Security

Privacy and security are two related concepts and people often falsely assume that privacy is only about security of personal information [Hal16]. While there is a clear symbiosis between security and privacy [Hal16] and security is a valuable building block for achieving privacy, security alone is insufficient to protect privacy [HNLL04]. Already in 1975, Saltzer and Schroder drew a clear distinction between privacy and

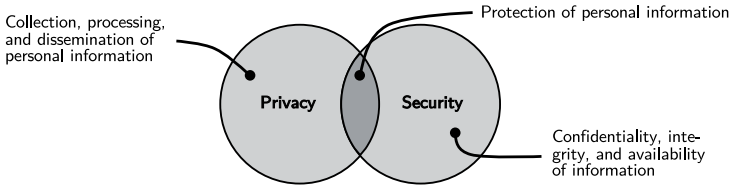

Figure 2.5 Privacy and security of information are two distinct yet related concepts. They intersect when considering the protection of personal information, i.e., providing privacy by means of security (presentation inspired by Brooks et al. [BGL+17] and Halter [Hal16]).

security [SS75]. While they define privacy as the ability to decide about the release of personal information, they consider security as the technical mechanisms that control read and write access to stored information. Given the similar but still clearly distinct definitions of privacy and security, it is important to understand both boundaries and overlap between privacy and security to identify how security mechanisms and techniques can be applied to protect personal information and to determine where such security mechanisms and techniques alone do not suffice to provide privacy [BGL+17]. This is especially important since the security mindset often significantly deviates from what is required to guarantee privacy [HNLL04].

In Figure 2.5, we highlight the differences and the overlap between privacy and security [BGL+17, Hal16]. Here, privacy—as defined in Section 2.2.2—concerns awareness and control over the collection, processing, and dissemination of personal information. In contrast, security is mainly concerned with the confidentiality, integrity, and availability of information [ISO13,KV10]: Confidentiality aims at preventing the unauthorized disclosure of information, both intentional and unintentional. Integrity guarantees that information is not modified in any way without proper authorization and is consistent across systems. This guarantee, most notably, includes that no outside entity can tamper with information that is being stored, processed, or transferred. Finally, availability allows access to information in a timely and reliable manner—a property that is especially important when outsourcing information to cloud services. In this setting, the intersection between privacy and security is hence concerned with the protection of personal information, which mainly is achieved by providing confidentiality of said information.

Another, slightly different, way for looking at the two related concepts of privacy and security is proposed by Flavián and Guinalíu in the context of loyalty with websites [FG06]. In their work, they consider privacy as legal requirements and good practices with respect to the handling of personal information. Security then encompasses the technical guarantees that these legal requirements and good practices are indeed met. This part of the overall field of security is what we highlight as the overlap between privacy and security in Figure 2.5.

Notably, privacy and security are two concepts that can also work against each other. For example, TLS client certificates enable web services to securely authenticate their clients during the initial handshake. However, as these client certificates are transferred in clear text, anyone on the communication path, e.g., internet ser-

vice providers (ISPs), can use them to track users [WSC17], thus clearly harming their privacy. Similar problems can be observed for other cryptographic identifiers, such as public keys. Likewise, protecting privacy can also negatively impact security. As one out of many examples, anonymous communication networks, such as Tor, which provide users with privacy when accessing resources on the Internet [PLZ+16, PMH+17], can also be (mis)used to carry out denial of service (DoS) attacks [Dri15], thus impacting security, especially with respect to availability. For our work presented in this dissertation, it is consequentially important to consider the impact of security measures on privacy as well as to ensure that our approaches to increase privacy do not negatively impact equally important security goals.

Now that we have introduced privacy in the cloud computing context, derived an information-centric definition of privacy in cloud computing, and set privacy apart from security, we are well-prepared to study the distinct privacy challenges introduced by the cloud computing paradigm in the following.

2.3 Privacy Challenges of Cloud Computing

To gain a deeper understanding of the privacy challenges of cloud computing, we first study the privacy risks faced by the different actors in the cloud computing landscape. Here, we rely on and adopt the privacy risk analysis of Pearson [Pea09], thereby focusing on those individual actors in the cloud computing landscape that are especially important with respect to privacy (cf. Section 2.1.3): For *private cloud users*, privacy risks with respect to cloud computing consist of potential exposure of personal information (cf. Section 2.2.1), e.g., either intentional or unintentional.

In contrast, *corporate cloud users* are mostly at risk regarding not complying with corporate policies or legislation (and thus enormous fines) as well as loss of credibility and reputation. When considering *cloud service providers*, risks with respect to privacy mostly concern non-compliance with legal obligations, loss of reputation, and unauthorized use of stored personal customer information by infrastructure providers. Risks for *cloud infrastructure providers* are concerned with unintended exposure of sensitive information stored on the infrastructure and resulting legal liability as well as loss of user trust, credibility, and reputation. Finally, *legislators* are at risk regarding being unable to enforce enacted privacy requirements as well as losing governance and control over data. Hence, the different actors in the cloud computing landscape each face different privacy risks and hence have a different perspective on privacy. We argue that these different perspectives need to be incorporated when building technical systems to increase the level of privacy offered by cloud computing.

These privacy risks result from four core problems for privacy in cloud computing as we identified in Section 1.1.3 and briefly recap here. First, *technical complexity and missing transparency* result from the layered architecture of cloud computing and resulting technically complex deployment models with often indirect utilization of resources, e.g., due to the tendency of cloud services to subcontract other cloud services [PP15]. As a result of this technically complex realization of cloud services

that lacks transparency for users, users have to trust an unknown number of cloud services. Furthermore, technical complexity and missing transparency also induce *opaque legislation*, where it is often unclear under which jurisdiction users' data falls.

Most notably, if data is moved between data centers (or even cloud services), e.g., to balance load or to recover from outages, the jurisdiction under which data falls can change during the lifetime of this data. As a result, not only (less technically proficient) users, but also providers of cloud services fail to know to which (other) cloud services data flows to [AGM10]. Due to the *inherent centrality* of the cloud computing market, a small number of cloud services jointly dominate the field and hence become a valuable target both for attackers and government agencies. Besides, users have only very limited alternatives for selecting a potentially more privacy-friendly cloud service. These three core problems culminate in *missing control* over information when it is collected, processed, stored, and disseminated by cloud services. More precisely, any information that leaves the control sphere of its owner could be used for unintended purposes, handled in violation of legal requirements, or inadvertently forwarded to any third parties [PB10, TJA10, ZGW14].

These core problems for privacy in cloud computing clearly highlight that accounting for privacy in the cloud computing landscape is an urging and important problem. Indeed, the various dimensions of the privacy challenges of cloud computing have been widely studied before. We briefly summarize the most important and influential approaches in the following and distill those challenges that are of special relevance for the approaches presented in this dissertation. First and on a general note, Cavoukian [Cav08] investigates privacy in cloud computing and states that it is impossible to fully realize the benefits of cloud computing without better protection of privacy. From a more technical perspective, Theoharidou et al. [TPPG13] examine the privacy risks resulting from migrating data, applications, or services to cloud services. Likewise, NIST [JG11] states that understanding procedures, policies, and technical measures employed by cloud services is key to assess resulting privacy risks.

Pearson and Benameur [PB10] identify privacy issues arising from cloud computing and propose measures ranging from data handling mechanisms over design for privacy to standardization to overcome these issues. Takabi et al. [TJA10] study privacy challenges resulting from cloud computing and identify data-centric privacy, trust management, and access control as promising approaches to address these challenges. Ghorbel et al. [GGJ17] survey privacy challenges as well as risks of public cloud computing and conclude that user control, policy enforcement, and the lack of user awareness are key open issues that need to be tackled. Pearson [Pea09] provides guidelines on how to design cloud services in a privacy-preserving manner, while Claycomb and Nicoll [CN12] argue that it is especially important to also focus on privacy challenges resulting from insider threats.

From a different perspective, Pearson [Pea13] argues that cloud business scenarios have to take into account that the collection, processing, storage, and dissemination of personal information is (heavily) regulated in many countries. Mather et al. [MKL09] study privacy challenges of cloud computing from an enterprise perspective, thereby specifically focusing on risk and compliance concerns. Focusing on the users

whose privacy is (potentially) impacted, Ion et al. [ISKČ11] derive privacy challenges of cloud storage by surveying users on their privacy concerns. De Filippi et al. [FM12] study the impact of cloud computing on society, especially with respect to privacy challenges resulting from its centralized deployment model and transborder data flows. In contrast, Gellmann [Gel09] and Millard [Mil13] focus on legal challenges resulting from the impact of cloud computing, especially with respect to the privacy of personal information.

Microsoft's policy considerations and recommendations [Mic16a] focus on clear and enforceable privacy frameworks established by governments to achieve important properties such as transparency, control, and consent. Similarly, Jaeger et al. [JLG08] study privacy challenges of cloud computing as part of a survey of the policy issues raised by cloud computing.

In the following, we detail those aspects of privacy in cloud computing that are of special importance within the context of this dissertation. To this end, we first study data handling requirements and legal obligations that have to be considered when collecting, processing, storing, and disseminating personal information in cloud services. Then, we derive potential attacks originating from different actors in the cloud computing landscape and outside entities. Finally, we present key principles and guidelines for realizing privacy-preserving cloud services.

2.3.1 Data Handling Requirements and Legal Obligations

With the increasing demand for sharing data and storing it with external parties [SV10], obeying with data handling requirements (DHRs) imposed by clients and lawmakers as well as accounting for other legal obligations becomes a crucial challenge for cloud services [WMF13]. DHRs involve constraints on the storage, processing, distribution, and deletion of data in cloud services. These constraints follow from legal [HIPA96, EU95], contractual [PCI15], or intrinsic requirements [BYV+09, RTSS09]. They range from restrictions on storage locations or durations [Gel09, JG11, PB10] to certain properties of the storage medium such as full disk encryption [GLBA99, SSFS12]. Especially for businesses, compliance with legal and contractual obligations is important to avoid serious (financial) consequences [MNP+11]. In the following, we discuss DHRs from the perspectives of cloud users and providers based on common classes [HMH+18].

Location Requirements

Privacy concerns regarding the storage and processing of personal information can depend on the location where this storage and processing takes places. First, location requirements can be imposed by legislation to address concerns raised when personal information is stored and processed outside of specified legislative boundaries [PB10]. Cory [Cor17] provides an extensive overview that contains most of the formal location requirements imposed by laws and regulations worldwide. The EU data protection directive 95/46/EC [EU95], e.g., forbids the storage of personal

information in jurisdictions with an insufficient (as specified by the directive) level of data protection. Similar data protection legislation and regulation that restricts the storage and processing of personal information in other jurisdictions have, e.g., been enacted in Argentina, Indonesia, and Malaysia [Cor17].

Also other legislation, besides data protection laws, can impose restrictions on the storage and processing location of personal information. German tax legislation, e.g., forbids the storage of tax data outside of the country (with certain exceptions for multinational companies) [Cor17]. Similarly, US regulations prohibit the disclosure of certain taxpayers' personal information, e.g., the social security number, to entities located outside the US [Gel09]. In France, all data created by public administration has to be stored within the country [Cor17].

Besides legislation, especially corporate users may impose location requirements themselves. To increase robustness against outages, a company might demand to store replicas of their data on different continents [BYV+09]. Furthermore, an enterprise could require that sensitive data is not colocated with data of competitors for fear of accidental leaks or deliberate breaches [RTSS09]. Similarly, customers from outside the US could prefer to not store their data in the US because of fears that government agencies might access their data under the Patriot Act [RFVE11].

Duration Requirements

Additionally, with respect to the storage duration of data, legislation and regulation impose various restrictions that have to be considered when outsourcing storage and processing of personal information to cloud services. For example, the Sarbanes-Oxley Act (SOX) [SOX02] requires accounting firms in the US to retain records relevant to audits and reviews for seven years, thus imposing a minimum storage duration. In contrast, the Payment Card Industry Data Security Standard (PCI DSS) [PCI15] limits the storage duration of cardholder data in the US to the time necessary for business, legal, or regulatory purposes after which it has to be deleted, hence dictating a maximum storage duration. A similar approach, coined "the right to be forgotten" or "right to erasure", is actively being discussed and turned into legislation in the EU and Argentina [Man13,Ros12,GDPR16]. From a user perspective, being able to properly limit the storage duration is important, since 63 % of users of online services surveyed by the Pew Research Center in 2008 were very concerned that cloud services keep a copy of files even if users try to delete them [Hor08].

Further and Future Requirements

Finally, also further requirements impose restrictions on how data should be stored and processed by cloud services. As an example, HIPAA [HIPAA96] requires health data to be securely deleted before a storage medium is disposed or reused. Likewise, for the banking and financial services industry in the US, the Gramm-Leach-Bliley Act (GLBA) [GLBA99] imposes the proper encryption of stored customer data. Additionally, to protect against theft or seizure, clients may choose to store their data only on volatile [JMR+14] or fully encrypted [SSFS12] storage devices.

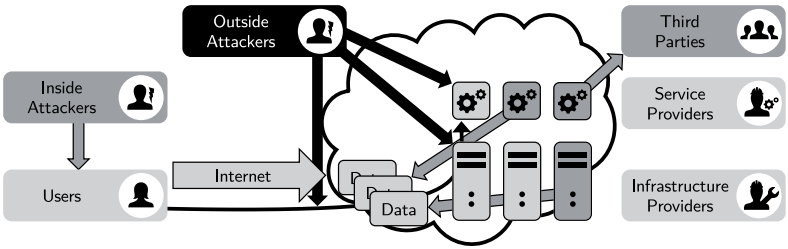

Figure 2.6 Attacks that threaten privacy in cloud computing originate from a variety of internal (dark gray) as well as external (black) entities.

Besides these already existing requirements, it is important to note that DHRs are likely to change and evolve just as legislation and cloud storage technologies are changing and evolving over time. For example, location requirements developed since cloud systems began to span multiple regions with different legislation. When tackling the challenges resulting from DHRs in cloud services, it is thus important to keep in mind that yet unforeseen requirements and legislative developments can and likely will emerge.

2.3.2 Attack Models

So far, we mainly discussed different privacy risks and challenges. Yet, for some of these risks, e.g., the exposure of personal information to third parties, it is also important to identify where exactly these risks and challenges originate from, i.e., from which entities. To identify these entities, we rely on attack models which are a standard tool to formulate security assumptions and considerations in the security and privacy research community. In the following, we summarize different attack models with respect to privacy in cloud computing as a foundation to better understand how we can protect against resulting threats and risks.

We provide an overview covering the different attackers and entities that pose threats and risks to privacy in Figure 2.6. When studying privacy risks in the cloud computing landscape, it is especially important to differentiate between inside and outside attacks [Beh11]: Inside attacks originate from all entities involved in the collection, processing, storage, and dissemination of personal information in the cloud computing context. In contrast, outside attacks evolve from all entities that are not involved in delivering cloud services per se, but potentially have interest in gaining access to personal information stored and processed in the cloud.

Attacks from Outside Entities

As shown in Figure 2.6, outside attacks mainly originate from hackers and attackers [Beh11]. First, on-path network attacks try to interfere with the network connectivity between a user and her cloud services, e.g., to gain access to the content of

the communication taking place. Since these attacks require access to the network infrastructure, typical entities from which such attacks originate include ISPs and government agencies. A second class of attacks directly targets cloud resources such as virtual machines by exploiting security vulnerabilities or cracking passwords. Such attacks can be executed by anyone since they merely require access to the Internet.

While these two threats are not specific to cloud services, a third, cloud-specific, threat originates from potentially insecure interfaces and APIs exposed by cloud services [CSA10]. Security vulnerabilities in these interfaces and APIs as well as weak authentication tokens can again be exploited by anyone since the interfaces and APIs exposed by cloud services are typically accessible from the Internet. While all these threats pose risks for personal information, they can be circumvented using standard security measures and are less relevant when focusing specifically on the privacy challenges of cloud computing [CSA10, Beh11]. From a different perspective, privacy measures can also protect against attacks originating from outside entities. For example, if the information on which cloud service or data center a (corporate) user utilizes is kept private, outside attackers cannot (easily) launch DoS attacks.

Attacks from Inside Entities

In contrast, attacks originating from inside entities are especially important with respect to privacy. As we highlight in Figure 2.6, these entities range from the cloud providers at the infrastructure and service level over their employees to malicious employees of a corporate cloud user [CSA10, CN12]. All these entities have in common that they can exploit their privileges to negatively affect confidentiality, integrity, or availability of personal information collected, processed, stored, and disseminated by cloud services [CMT12]. While the privacy threats and risks originating from malicious employees of a corporate cloud user are not specific to cloud computing, the threats and risks originating from cloud providers both at the infrastructure and service level as well as their employees clearly are.

To account for these threats and risks, we can rely on attack models that define security and privacy assumptions regarding these entities and hence conceptually define against which type of attacks we need to protect. Traditionally designed for the security landscape, attacker models are an extremely valuable tool to model different threats and risks resulting from cloud infrastructure and service providers. To this end, Ryan [Rya14] identifies four attacker models for the cloud computing landscape which we briefly present and relate to privacy in the following. The rather weak assumption of an *honest* cloud provider assumes that neither the cloud provider nor its employees launch any attack against personal information under their control. On the other end of the two extremes, the model of a *malicious* cloud provider assumes a fully malignant cloud provider (or its employees) that can launch any attacks and hence can even completely deny the service [Gol04]. This model is likewise unrealistic as business models, contracts, and thus legal liability are opposed to a fully malicious behavior [Rya14].

Therefore, we turn our focus to the two realistic and widely used attacker models in the cloud computing field that lie between these two extremes. First, the notion

of an *honest-but-curious* or *semi-honest* cloud provider refers to a cloud provider that conscientiously delivers its service as agreed upon but might keep record of all information it can get access to while doing so [Gol04, Rya14]. In other words, an honest-but-curious cloud provider will launch only passive attacks [Rya14].

However, it is not always realistic to assume that a cloud provider (or its employees) will not launch any active attacks. To this end, Ryan introduces the notion of a *malicious-but-cautious* cloud provider that is able to launch active attacks as long as these attacks do not leave any readily verifiable evidence [Rya14]. This assumption is similar to those of a covert adversary [AL07]. However, the model of a malicious-but-cautious cloud provider additionally—and in contrast to a covert adversary—assumes that the cloud provider will protect its users from outside attacks [Rya14].

Which of the two attacker models is appropriate for a given scenario highly depends on the assets at stake, and thus, we rely on both within the scope of this dissertation, always selecting the one that is more appropriate in the respective context.

Attack models define security assumptions and considerations, especially with respect to the entities from which security and privacy risks originate. In the following, we discuss how privacy threats and risks in cloud computing can be mitigated by designing cloud services in a privacy-preserving manner.

2.3.3 Key Principles for Privacy-preserving Cloud Services

Rounding up our discussion of privacy challenges in the cloud computing landscape, we now take our look forward and discuss how to deal with these privacy challenges when designing (new) cloud services. To this end, we present different key principles and actionable practices to design and implement privacy-preserving cloud services.

Based on the well-accepted concept of Fair Information Principles [CSA96, PIPE00, TPPG13], Pearson derives nine key principles for designing privacy-preserving cloud services [Pea09], which we briefly summarize in the following. First, *notice, openness, and transparency* mandate that anyone who collects personal information must inform users about the purpose and extent of information usage, especially if collected personal information is shared with third parties. Most notably, this principle includes that users have to be provided with understandable privacy policies.

Through *choice, consent, and control*, users are empowered to decide whether their personal information should be collected or not. By requiring users' consent for any collection of personal information, they are put back into control over their privacy. *Scope and minimization* of information collection require that personal information should only be collected if it is required for the intended purpose, which leads to the requirement of minimizing information collection. *Access and accuracy* imply that users should have access to all their personal information stored in the cloud to verify its accuracy. This implies that all stored personal information has to be accurate at all times. *Security safeguards* provide users with technical guarantees that their personal information is protected against unauthorized access, usage, modification, disclosure, and forwarding.

The option to *challenge compliance* empowers users to contest a cloud provider's privacy process. To this end, it is important that cloud providers comply with privacy and data protection legislation, especially with respect to the potential transborder flow of information. *Purposeful use* mandates that any personal information that has been collected is only used for the stated intended purpose. Likewise, *limiting use, disclosure, and retention* implies that the storage of personal information should be limited to the period of time necessary for fulfilling the intended purpose. Furthermore, data should only be shared with those third parties that the user explicitly authorized to receive her data. Finally, *accountability* ensures that cloud providers adhere to privacy policies and practices, which includes the necessary technical means to monitor and log read and write access to stored personal information as a foundation for auditing capabilities.

Based on these key principles, we derive seven actionable privacy practices for developing cloud services inspired by Pearson's six recommendations for software engineers [Pea09] and the seven principles underlying "Privacy by Design" [Cav11]. First, cloud services should be designed *proactive not reactive*, i.e., (potential) privacy incidents should be identified and prevented before they occur instead of having to react to evolving privacy incidents. To provide users with the best possible level of privacy, *privacy by default* ensures that the default setting for a cloud service always is the most privacy-friendly one, i.e., users' privacy is automatically protected without requiring their interaction with the cloud service. Furthermore, *protection of personal information*, e.g., using end-to-end security and access control based on encryption as well as integrity protection based on checksums and digital signatures, ensures that personal information is not inadvertently accessed by, disclosed to, forwarded to, or modified by unauthorized third parties.

As a foundation for establishing trust, *user-centric control* guarantees users that they stay in control over their personal information. Such control can, e.g., be achieved using a cryptographically enforced access control system where users steer access to encrypted personal information by releasing decryption keys (cf. Section 5.2). *Visibility and transparency* ensure that the storage and processing of personal information are indeed carried out as stated and promised. By making the usage and transfer of personal information visible and transparent, users can verify that their personal information is actually used as intended, hence lay an essential foundation for trust into a cloud service. *Minimized collection* of personal information limits the collection, processing, storage, and dissemination of personal information to what is absolutely necessary to fulfill the intended purpose and thus minimizes privacy risks. On a similar note, *specified and limited purpose* of usage of personal information ensures that collected personal information is not (mis)used for purposes that are unintended by the user.

We have now laid out the necessary background information on cloud computing and its privacy challenges as a foundation for this dissertation. However, the level of privacy that cloud computing has to achieve highly depends on the individual application domain. In the scope of this dissertation, we utilize the Internet of Things and Cyber-physical Systems as exemplary application domains with notably strong privacy requirements, especially in Chapters 4 and 5, to further motivate our approaches and privacy assumptions. To this end, we additionally introduce

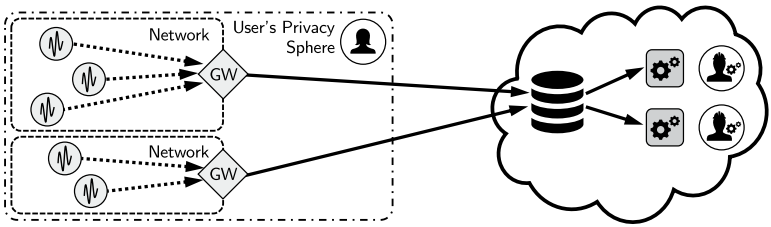

Figure 2.7 In the cloud-based IoT, each user operates one or more IoT networks. The IoT devices in these networks send data to the cloud via a gateway. The cloud stores data and provides it to services which are authorized by the user to access data.

the notion of the cloud-based Internet of Things and resulting ancillary privacy challenges, before we start presenting the contributions of this dissertation in detail.

2.4 The Cloud-based Internet of Things

Within the scope of this dissertation and especially in Chapters 4 and 5, we use the Internet of Things (IoT) and Cyber-physical Systems (CPS) as application domains for cloud services with extremely strict privacy requirements. In the following, we provide a brief introduction into the general motivation for the cloud-based IoT and CPS, typical network scenarios, as well as resulting privacy concerns and considerations that motivate the need for ancillary privacy measures.

The proliferation of the IoT and CPS, which enable the worldwide interconnection of an inconceivably large amount of smart things, allows to effectively realize systems that significantly improve everyday's life, ranging from pervasive healthcare and assisted living to smart cities [AIM10, GBMP13, ZGW14]. However, as these smart devices are often powered by battery, they often suffer from extremely constrained processing and storage resources, and a limited energy budget. To overcome these limitations, one of the most promising approaches is to interconnect the IoT with the cloud and thus benefit from the elastically scalable and always available resources provided by cloud computing [LVCD13, EHH+14, BDPP16, HHH+17]. Hence, utilizing the cloud-based IoT simplifies storage and processing of collected data, allows using the same data in multiple services, eases the combination of data from several users, and supports user mobility, while at the same time preventing fragmentation of information over several isolated silos.

In the following, we discuss the underlying network scenario of cloud-based IoT deployments as well as resulting privacy concerns and considerations [HHK+16].

2.4.1 Network Scenario

The underlying network scenario of the cloud-based IoT from a user-centric view is shown in Figure 2.7. Each user owns and thus operates one or more IoT devices,

also known as smart objects. We consider IoT devices that sense information from the environment, interact with the physical world, and—most importantly—allow communication using traditional Internet standards [GBMP13]. Prominent examples for IoT devices, e.g., in the context of assisted living, range from sensitive floors for movement monitoring and fall detection over smart textiles (e.g., shirts, wristbands, or shoes) monitoring various vital parameters and connected to emergency notification systems to advanced devices capable of monitoring specialized implants, such as artificial cardiac pacemakers [ZGW14, HHK+16].

Such IoT devices are typically realized using dedicated embedded platforms to reduce production costs and hence simplify deployment. As a result, IoT devices often have to cope with limited storage and processing resources. Especially in mobile settings, they furthermore suffer from limited connectivity and, as they are often powered by battery, a tightly limited energy budget [PDG+16].

The cloud-based IoT sets out to address these limitations by interconnecting the IoT with the cloud [LVCD13, EHH+14, BDPP16, HHH+17]. The core idea of the cloud-based IoT is to upload all sensed data to the cloud, where it is stored persistently. Users can then authorize specific cloud services to access and operate on their data and thus realize the desired functionality. Since the design of the cloud-based IoT aims to maximize availability (both of data and services), all functionality that operates on IoT data is realized directly in the cloud.

However, if the connection between an IoT device and the cloud is temporarily disrupted, data cannot be pushed to the cloud (immediately) and has to be cached locally and uploaded once the connection has been reestablished. Still, the cloud-based IoT allows accessing all data that is already stored in the cloud and operate services on it even during a disrupted connection between IoT device and cloud. As a result, availability of IoT data is significantly increased compared to solutions that propose to store and process IoT data locally (and hence become completely unavailable in case of connectivity issues).

As shown in Figure 2.7, IoT devices of one user are often grouped into one or multiple logically or even physically separated IoT networks, e.g., a home network consisting of assisted living devices and a body area network containing unobtrusive healthcare devices. Each of these networks is connected to the Internet (and thus the cloud) using a dedicated gateway. In the context of assisted living, this gateway typically consists of a home router while for public mobility assistance the user's smartphone acts as gateway.

The federated network of all IoT devices and networks of one user constitutes her privacy sphere, in which she trusts all devices and other network participants but does not want any personal information that is collected within this context to be available to unauthorized third parties. To guarantee this, she has to employ standard network security measures such as wireless channel encryption. IoT data sensed by the devices of a user is forwarded to the *cloud* via the dedicated gateway. The cloud stores all IoT data and makes it available to those cloud services that the user explicitly grants permission to access her data.

2.4.2 Privacy Concerns and Considerations

When realizing and implementing a scenario as described above, in which personal information collected by IoT devices is outsourced to the cloud, different privacy aspects have to be considered. We present and discuss these aspects in more detail in the following.

Data collected by IoT devices often consists of personal information that unauthorized third parties might be interested in [ZGW14]. As an example, data collected by a car-based telematics system can be extremely valuable for insurance companies, as this knowledge could be exploited to increase fees or even deny new contracts [Cou13]. Notably, not only sensed IoT data itself but also corresponding meta information has to be treated as sensitive [CRKH11], especially in the context of location privacy [ZVHW14] with meta information such as location fixes and time stamps collected by GPS, wireless networks, or NFC tags. Hence, users typically are reluctant to share data collected by their IoT devices with third parties [ZGW14].

These privacy concerns and issues further amplify when outsourcing this personal information to the cloud. Again, the major concern of users in this setting is the perceived loss of control over data when it is outsourced to the cloud [TJA10]. Furthermore, users are concerned about reasonable protection of their data, that legislation is adhered to, and scope and purpose of data usage [Smi12]. Due to these concerns, users ultimately tend to refrain from using cloud-based services for (highly) sensitive data, e.g., health-related information as it is stored and processed by cloud-based personal health records systems [LHL15]. To avoid this adoption barrier for users of cloud services in the context of the cloud-based IoT, cloud providers have to explicitly guarantee confidentiality and protection of stored and processed IoT data, because otherwise an adoption barrier can arise for users due to their individual privacy concerns.

Concluding our discussion of the cloud-based IoT, its network scenario, and resulting privacy concerns and considerations, we sum up that the cloud-based IoT is an application domain with strict privacy requirements. Hence, it is a prime candidate for showcasing solutions to account for privacy, especially in the context of our approaches presented in Chapters 4 and 5.

2.5 Summary

To summarize this chapter, we have identified that the cloud computing landscape is quite diverse and versatile. There are different ways of interplay and diverse relationships between the various actors in the cloud computing landscape and their interactions often take place indirectly and undetectable for the actual users whose privacy is consequently at risk. Based on the distinct characteristics of the cloud computing paradigm, we introduced the notion of privacy in the context of cloud computing. Most notably, we presented an information-centric definition of privacy in cloud computing and set privacy apart from security.

Our deeper look at the privacy challenges inherent to cloud computing revealed the importance of adhering to data handling requirements and legal obligations. Furthermore, the presented attack models enable us to properly define the inside and outside threats and risks to privacy that need to be addressed. The key principles and actionable practices for privacy-preserving cloud services provide the necessary guidance for designing and implementing our privacy approaches. Finally, we introduced the concept of the cloud-based IoT as an exemplary application domain for cloud services with strong privacy requirements. To provide the foundation for this example, we derived a typical network scenario for cloud-based IoT deployments and briefly discussed resulting privacy concerns and considerations.

As we have now laid out the necessary background information on cloud computing, privacy in general, and privacy challenges in the cloud computing landscape as well as the cloud-based IoT, we are now well-prepared and equipped with the necessary background knowledge for the main part of this dissertation. In the following, we apply this background knowledge when presenting the four distinct contributions of this dissertation that jointly address the three research questions underlying our approach to account for privacy in the cloud computing landscape (cf. Chapter 1).

3

Raising Awareness for Cloud Usage

In this chapter, we present approaches to provide users with transparency over their individual exposure to cloud services and hence raise their awareness for cloud usage and potentially resulting potential privacy risks. To this end, we first summarize the motivation for raising awareness for cloud usage as a foundation for users to make informed decisions and exercise their right to privacy as well as identify two prominent deployment domains for cloud services to showcase our work (Section 3.1).

As our first approach, we present MailAnalyzer [San16b, HSH17] to uncover the cloud exposure of email users based on information in received emails (Section 3.2). Our second approach, CloudAnalyzer [Müh14, Hel15, Dri16, HKH+16, HPH+17], likewise uncovers the cloud usage of mobile apps on smartphones by passively observing network traffic (Section 3.3). We round up our work on raising awareness for cloud usage by providing a feasibility study of an approach for privacy-preserving comparison of cloud usage [Ina17, HIFZ17] that enables users to contextualize their cloud usage through comparison with their peers (Section 3.4). Finally, we conclude this chapter with a brief summary and discussion (Section 3.5).

3.1 Motivation

A fundamental challenge with respect to privacy in cloud computing is its technical complexity and missing transparency, especially for less technically proficient users. This challenge becomes especially important since more and more everyday technology ranging from email over mobile apps on smartphones to the IoT relies on cloud resources, leading to a situation in which users' personal information is unconsciously exposed to cloud services [EHKR14], i.e., users are often unaware of (the extent of) the exposure of their personal information to cloud services. However, without even knowing that they are using cloud services, users cannot make informed decisions and exercise their right to privacy.

As a foundation to put users back into control over their privacy, we hence consider it necessary to uncover their cloud usage and raise their awareness of resulting privacy risks. To this end, we select two prominent deployment domains for cloud services with which even less technically proficient users (unconsciously) interact with on a daily basis: email and mobile apps on smartphones. With an estimated amount of 2.6 billion daily users sending 215.3 billion emails per day in 2016 [Rad16], email clearly constitutes a significant communication medium. Likewise, we observe a tremendous increase in the worldwide adoption of smartphones with more than 340 million units sold in the first quarter of 2017 [IDC17, PHW17]. These numbers highlight the importance and relevance of these two deployment domains for cloud services, which are used by a wide range of often less technically proficient users.

As we show in this chapter, both deployment domains significantly rely on a potpourri of cloud services nowadays. Our analysis of the cloud exposure of email users uncovers that as of 2016, 13 % to 25 % of received emails utilized cloud services and that between 30 % and 70 % of this cloud usage cannot be (easily) detected by non-expert users. Similarly, our study of the cloud usage of mobile apps for the Android platform reveals an excessive exposure to cloud services as 90 % of apps use cloud services (with an average usage of 3.2 cloud services per app) and 36 % of apps used by volunteers in our study exclusively communicate with cloud services. These exemplary numbers show that everyday technology, such as email and mobile apps on smartphones, significantly relies on cloud services, both in term of the fraction of cloud usage and the number of utilized cloud services.

However, it is extremely difficult for users to correlate their quantified cloud usage behavior to resulting potential privacy risks. To this end, the concept of comparison-based privacy [ZHHW15] allows users to compare themselves with their peers, i.e., like-minded individuals of similar social status—in our case with respect to the usage of cloud services in their immediate social contexts. While applying comparison-based privacy to nudge users on cloud usage is extremely promising, it also in itself introduces privacy concerns since cloud usage statistics constitute sensitive information. Hence, applying comparison-based privacy to compare the cloud exposure of users requires the design and implementation of a privacy-preserving system that protects contributed personal information.

3.1.1 Contributions

To provide users with transparency over their individual exposure to cloud services and thus, raise their awareness of the potential privacy risks resulting from this cloud usage, we present the following contributions in this chapter.

1) We present *MailAnalyzer* which uncovers the cloud usage of email by dissecting header information of received emails to detect cloud services on the path from the sender to the receiver. To this end, MailAnalyzer uses information publicly provided by a representative set of 31 cloud and email providers as well as patterns derived from the Internet infrastructure, such as DNS or BGP routing data. MailAnalyzer is especially valuable for the large fraction of hidden cloud usage

that cannot easily be observed by users. We employ MailAnalyzer to understand the cloud email infrastructure (contacted when sending email) by identifying email servers running on cloud infrastructure in the entire IPv4 address space and uncover cloud usage for all 154 million .com/.net/.org domains. Furthermore, we utilize MailAnalyzer to analyze the cloud usage of 31 million exchanged emails, ranging from public mailing list archives to the personal emails of 20 volunteer users (Section 3.2).

2) We present *CloudAnalyzer* which reveals the cloud usage of mobile apps on off-the-shelf smartphones by locally monitoring the network traffic produced by apps. To this end, CloudAnalyzer relies on a set of 55 representative cloud services that we derive from a thorough analysis of the landscape of cloud services utilized by mobile apps. Besides the cloud service(s) an app is directly using, CloudAnalyzer identifies the indirect use of cloud resources resulting from services realized on top of cloud infrastructure. We apply CloudAnalyzer to study the cloud exposure of 29 volunteer users over a period of 19 days, to analyze the cloud usage of the 5000 most popular mobile websites, and to compare the cloud usage of the 500 most popular apps when launched from five different countries (Section 3.3).
3) We realize the concept of *comparison-based privacy* [ZHHW15] in a privacy-preserving manner to enable users to compare their personal cloud exposure to that of their peers. To this end, we introduce a privacy proxy that hides users' identities and employs k-anonymity and differential privacy on encrypted cloud usage statistics to aggregate and to further protect user contributions from disclosure. We preliminarily study the feasibility and applicability of our approach by utilizing it to protect cloud usage statistics of the apps running on the smartphones of 29 volunteers over the course of 19 days (Section 3.4).

3.2 MailAnalyzer: Uncovering the Cloud Exposure of Email Users

Email is one of the oldest and most prominent Internet services and remains an important communication medium. Its significance is expressed by current usage statistics, e.g., Radicati [Rad16] estimates more than 2.6 billion email users sending 215.3 billion emails per day in 2016. To cope with the steady increase in usage, email is currently experiencing an architectural change from a largely *decentralized* medium towards a more *centralized* one [MLB+11]. The reason for this shift is the ongoing trend to outsource email services to cloud operators, either by hosting email servers inside the cloud or by adopting cloud email providers. Compared to the classical decentralized email infrastructure in which each organization operates its own email service, cloud email affords to run email services in a more flexible, scalable, and cost-efficient manner [BL07]. Email running in the cloud ranges from email servers running on cloud infrastructure, e.g., Amazon EC2 or Microsoft Azure, over cloud-based email security services, such as SPAM and DoS protection, to cloud-hosted email services for end users, e.g., Gmail and Outlook.com.

While cloud computing affords the flexible handling of increasing demands of email, it also raises privacy concerns. These concerns are rooted in the fact that emails

are inadvertently forwarded to third parties. This forwarding—and the disclosure of data to third parties—is often unknown to the sender (e.g., email addressed to a state-owned university can be handled by a public cloud provider). Due to this forwarding, exposed data can be used for unintended purposes (e.g., personalized advertising [Goo18c]), or can be handled and stored violating legal requirements. Furthermore, concentrating emails at a few large providers renders those valuable attack targets, as exemplified by the breach of all 3 billion Yahoo accounts [Per17]. From a different perspective, processing email by large cloud providers can raise jurisdiction and privacy concerns [ISKČ11, FKH15], especially when their usage is not visible to users, i.e., cannot be inferred from the sender or receiver address (cf. Section 2.3). Due to the centralized nature of cloud infrastructures, email stored in the cloud is further susceptible to governmental access, e.g., for safety, security, economic, or scientific purposes [Cun16]. Users have become aware of the resulting threats to their privacy by recent global surveillance disclosures [Gel13].

However, users have only very limited knowledge of their exposure to cloud email services, i.e., how much of their email is processed by cloud services. The goal of our work is thus to provide a comprehensive assessment of the prevalence of cloud email. We start by understanding the cloud email *infrastructure*, i.e., the set of email servers hosted in cloud environments. We, therefore, identify all publicly reachable SMTP servers in the entire IPv4 address space and further analyze email servers configured for all 154 million .com/.net/.org domains. While this first part provides us with an empirical understanding of email infrastructure hosted in the cloud, it does not provide insights on if and how this infrastructure is actually used. To analyze the *user exposure* to the cloud, we analyze actual email exchanges in the second and main part of our study. We thus analyze both (i) emails from public email archives providing longitudinal data and (ii) emails from personal mailboxes of volunteers in a user study, totaling to more than 31 million exchanged emails. To ease reproducibility of our results and to pave the way for further research, we make our source code, detection patterns for cloud services, as well as anonymized aggregated study results available under the MIT license[2].

3.2.1 Cloud-based Email and Privacy

We observe a steady trend of moving email services to the cloud to cope with the ever-increasing amount of emails being sent. Likewise, large corporations have also shifted their on-premise email infrastructure to the cloud. To further understand this trend, we first provide an overview of the different types of cloud email services and distill a representative set of services. Based on this analysis, we derive resulting privacy risks of cloud-based email deployments and discuss related work.

3.2.1.1 The Cloud-based Email Landscape

Prior to the emergence of cloud-based email services, outsourced email services were generally differentiated into email *providers* (i.e., services providing email services

[2] `https://github.com/COMSYS/MailAnalyzer`

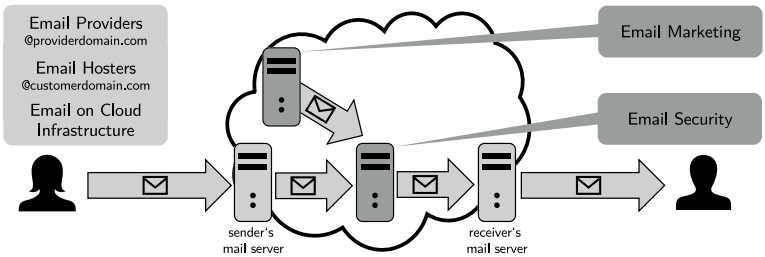

Figure 3.1 In the cloud-based email landscape, existing email infrastructure is either migrated to the cloud (light gray) or new types of infrastructure emerge (dark gray).

under their own domain) and email *hosters* (i.e., services that provide email services under the domain of the customer). As shown in Figure 3.1, when moving email services to the cloud, the landscape of email services becomes more diverse. In the following, we discuss the landscape of different types of cloud-based email services, consisting of traditional outsourced email services that have moved to the cloud and new services, that could only emerge because of the cloud. Furthermore, we compile a representative list of the most influential cloud services for each class. We provide the full list of the 31 cloud services that we selected in Table 3.1 and, in the following, focus on justifying the reasoning behind our selection of these services.

Email Providers. Email providers offer typical email services, i.e., a mailbox with the possibility to send and receive emails. Notably, email addresses served by email providers are bound to the domain of the individual provider (e.g., `@aol.com`). Email providers often offer services for free and finance their services through advertisements [Rob09]. Hence, the majority of their customers are private users. We base our selection of cloud-based email providers on a survey conducted by Adestra [Ade16]. In our analysis, we include the six most popular email providers as used by the 1200 study participants (US residents, all age ranges) as their primary email provider. These six providers account for 96 % of the participant's primary email providers.

Email Hosters. Email hosters offer basic email services under the domain of the customer, where each customer will use their own domain (e.g., `@example.com`). Typically, email hosters charge for their services, e.g., based on the size and amount of mailboxes. While private users also use hosters, the majority of customers are corporations and businesses. In contrast to email providers, it is not possible to derive the hoster directly from a hosted email address. We are especially interested in services hosting emails for a large number of domains. Hence, we rely on measurements performed by DomainTools on the most popular email servers according to the number of domains they serve [Dom16]. Based on these results, we include the top five hosters of popular email servers in our analysis.

Email on Cloud Infrastructure. Cloud computing enables the transformation of arbitrary services from own on-premise-hardware to virtualized infrastructure running in a cloud data center. Hence, cloud computing affords the transfer of previously self-hosted email servers to a cloud infrastructure. The main motivation for this

Service	Provider	Hoster	Infrast.	Security	Market.	Source(s)
1&1	○	●	○		○	[Dom16]
Adobe					●	[McA16]
Amazon		○	●		○	[LPGD16]
AOL	●					[Ade16]
AppRiver		○		●		[Clo16]
CenturyLink	○	○	●	○		[LPGD16]
Cisco		○	○	●		[Clo16]
Comcast	●	○				[Ade16]
Epsilon					●	[McA16]
Experian					●	[McA16]
Fujitsu		○	●	○		[LPGD16]
GoDaddy		●	○		○	[Dom16]
Google	●	●	●			[LPGD16] [Ade16] [Dom16]
IBM (SoftLayer)			●		○	[LPGD16]
iCloud	●					[Ade16]
MAX MailProtection				●		[Clo16]
McAfee				●		[Clo16]
Microsoft	●	○	●	○	○	[LPGD16] [Ade16]
Mimecast		○		●		[Clo16]
NTT Communications		○	●			[LPGD16]
Oracle			○		●	[McA16]
OVH		●	○			[Dom16]
Proofpoint				●		[Clo16]
Rackspace		○	●			[LPGD16]
Salesforce					●	[McA16]
Strato		●	○			[Dom16]
Symantec				●		[Clo16]
TrendMicro				●		[Clo16]
Virtustream			●			[LPGD16]
VMware			●			[LPGD16]
Yahoo	●	○				[Ade16]

Table 3.1 Our representative set of 31 services covers the different classes of cloud email services. We use ● to denote services that we consider representative for each class of cloud email services, while ○ denotes less prominent services for a class.

transition are cost reductions, lower maintenance efforts, and elastic scalability. As moving an email server to a cloud infrastructure still requires the setup and administration of an email server, this approach is mainly pursued by businesses. For our selection of cloud infrastructure (IaaS) providers, we build upon a market analysis performed by Gartner [LPGD16]. Based on this analysis, we select the ten cloud infrastructure services with the highest market share, as those jointly dominate the market according to Gartner [LPGD16].

Email Security. Mail servers are subject to a number of security threats, ranging from SPAM and malware to DoS attacks, against which cloud-based email security services promise better protection. To this end, they use the resources of the cloud to operate security proxies for incoming and outgoing email traffic, effectively hiding the identity of the actual email server. For our selection of cloud-based email security services, we rely on the analysis tools of CloudEmailSecurity.org [Clo16]. We include all eight services that are featured in their survey into our analysis, as we could not find reliable information on their market shares to further narrow down our selection.

Email Marketing. Cloud-based email marketing services enable the sending of large amounts of personalized emails for marketing purposes, e.g., to advertise products, engage with customers, or solicit donations. We base our selection of cloud-based email marketing services on an analysis of Forrester [McA16]. From these results, we derive the five services with the strongest market presence for our analysis.

It should be noted that these five different classes are neither unambiguous nor distinct. For example, larger email *providers* often additionally offer customers to *host* customer domains (while less known, e.g., Google and Microsoft also offer email hosting). Furthermore, a provider can offer more than one service, e.g., generic cloud infrastructure and email marketing in the case of Amazon. Hence, only an exhaustive picture of the landscape of cloud-based email services as derived in this section and summarized in Table 3.1 ensures a solid understanding of the impact of cloud computing on email users.

3.2.1.2 Privacy Problems of Cloud-based Email

Already traditional email hosted outside the cloud raises severe privacy concerns. In a survey conducted by Udo [Udo01], 55.1 % of 158 participants named privacy as their most important concern about email. Indeed, emails often contain private information, ranging from conversations about doctoral appointments to business secrets. Notably, even rather "uncritical" emails such as newsletters might reveal sensitive information, such as interests and habits [Goo18c]. Hence, users' privacy concerns are well-justified and it is reasonable for users to care about who has access to their emails. To counter these privacy concerns, users can protect their emails using encryption. However, applying end-to-end encryption to emails is cumbersome and only rarely used in practice [RKB+13]. Furthermore, while encryption can protect the content (body) of an email, insightful meta information such as subject, sender, receiver, and sending time contained in the header remain readable to any entity storing or forwarding an email.

When users' emails are exposed to the cloud, these already existing privacy problems further exacerbate (cf. Section 2.3). Most notably, there is no way for users to opt-out of being impacted by cloud computing when sending and receiving emails. While users have the choice to choose a non-cloud-based email service for themselves, they cannot influence which email services their communication partners are using. Hence, even if a user deliberately refrains from using a cloud-based email service, e.g., due to privacy concerns, such services still process a surprisingly large fraction of a user's total email communication.

As we show in the following, this processing of emails is especially problematic for the *hidden* usage of cloud-based email services. While a user can conclude from the sender and receiver addresses of an email whether an email is exposed to the cloud (e.g., for `@gmail.com` addresses), the absence of an obvious cloud-based email address does not guarantee that an email is not exposed to the cloud. Specifically, the usage of email hosters, cloud infrastructure, security services, and marketing services typically remains hidden from users.

3.2.1.3 Related Work

Different lines of research provide valuable input for our goal to uncover cloud exposure of email users. To this end, we structure our discussion of related work into approaches for understanding email, cloud computing, and cloud-based email.

Understanding Email Traffic. Ramachandran et al. [RF06] study the properties of SPAM emails based on network-level observations, e.g., IP address ranges used to send SPAM emails. They find that network-level characteristics can indeed be used to tell SPAM and legitimate email apart. Motivated by these findings, Hao et al. [HSF+09] propose a reputation engine for emails based on network-level characteristics. They report that their fully automated approach achieves comparable SPAM classification rates to hand-labeled blacklists. From a different line of research, Schatzmann et al. [SMSD10] strive to classify *webmail* traffic to gain a comprehensive view of the Internet email infrastructure. To this end, they develop flow-level techniques operating solely on passive network measurements to reliably tell webmail traffic and other HTTPS traffic apart.

Understanding Cloud Traffic. Bermudez et al. [BTMM13] utilize DNS responses to detect cloud services based on network traffic. Their approach proofs especially valuable with an increased fraction of encrypted network traffic. Their results reveal that the vast majority of traffic generated by Amazon Web Services originates from a single data center. Similarly, Drago et al. [DMM+12] study the properties of personal cloud storage services. They perform passive measurements and distinguish between different cloud storage services based on information contained in DNS and TLS network packets. He et al. [HFW+13] present a measurement study to understand the deployment of web service on cloud infrastructure. They rely on DNS probing to identify which popular web services use Amazon's and Microsoft's cloud offers and conclude that 4 % of the most popular web services run on infrastructure operated by Amazon and Microsoft. Likewise, Fiadino et al. [FSC15] discuss an analysis of WhatsApp based on passive measurements from the core of a cellular network and geo-distributed active measurements. They find that WhatsApp is hosted by a single cloud service, namely SoftLayer, in data centers in the US. From a completely different perspective and with the goal to optimize costs and performance of cloud storage systems, Liu et al. [LHFY13] analyze snapshots of the file system and an access trace of a campus cloud storage system.

Understanding Cloud-based Email. Willett et al. [WS14] performed a survey to quantify the adoption of cloud-based email services at higher education institutions in South Africa. They observed that the majority of institutions are using cloud-based email services or plan to do so in the near future. A study performed by Hsu et al. [HRL14] targets the cloud email adoption of the largest Taiwanese companies. Their results indicate that 44 % of the companies have migrated their email system to the cloud or plan to migrate within one year. Gartner [DM16] analyzed the DNS records of nearly 40 000 companies to check for Google or Microsoft usage as an email hoster. They discover that about 13 % of the studied companies use one of the two email providers. Xie et al. [XYA+07] analyzed Microsoft Hotmail traces to identify dynamic IP addresses for SPAM filtering. Finally, van Rijswijk-Deij et al. [RJSP16] analyze the growth of cloud-based email services based on DNS records for the .com zone. They observe that the largest (by the number of domains) cloud-based email hosters are Google, Microsoft, and Yahoo.

While these works highlight the importance of understanding the impact of cloud computing on email users, an empirical evaluation—which is the focus of our work—of both the cloud usage among the email infrastructure and the users' exposure to

cloud services, has not been done so far. Shedding light on this question is relevant to better understand potential cloud-related privacy exposures of email. Such understanding is especially relevant since even if users decide to refrain from using cloud-based email services themselves, their privacy can still be impacted by communication partners that are using cloud services. Hence, we argue that understanding the impact of cloud computing on email users is an important question, as it (i) ascertains whether user privacy is indeed at risk, (ii) provides insights into which cloud-based email services are the most often used, especially with respect to hidden cloud usage, and (iii) lays the foundation for deriving appropriate countermeasures. Following the classification derived in this section, we next introduce MailAnalyzer, our approach which we use to assess the cloud exposure of email users.

3.2.2 Detecting Cloud Usage of Emails

To detect cloud usage of emails, we present MailAnalyzer which reveals the cloud exposure of email users by analyzing received emails for the presence of cloud services. MailAnalyzer dissects the header information of emails and compares certain header information against patterns we derived for our set of 31 cloud services.

In the following, we identify the individual parts of an email header that can be used to discover the usage of cloud services, show how patterns to uncover cloud usage can be derived, and discuss limitations of our approach.

3.2.2.1 Dissecting Email Headers to Detect Cloud Usage

MailAnalyzer processes header information of received emails to detect cloud usage. To illustrate our approach, we partially depict the header of an email exchanged between a Gmail account and a university account in Listing 3.1. In the following, we identify the parts of an email header that can be used to reveal exposure to cloud services. We differentiate between information that directly allows the detection of cloud usage (green) and information that hints at potential cloud usage based on sender and receiver information (bright red), which can be used to rule out hidden cloud usage. To leverage information contained in email headers to detect cloud usage, we require patterns that enable the detection of a specific cloud service. Most notably, these patterns include information on the utilized IP addresses and DNS names of cloud-based email services. Hence, in the following, we do not only identify those parts of email headers that can be used to detect cloud usage, but also illustrate how the corresponding patterns can be derived from public information.

Received Lines. The main purpose of received lines is to aid debugging of email failures [Kle08]. To this end, each email server that receives an email (either for forwarding or for final delivery) has to prepend a received line to the email's header [Kle08]. While the exact format of received lines can deviate from the specification [Kle08], they typically contain the IP address (a typically unique identifier assigned to each networked computer [Pos81]) and DNS hostname (a human readable identifier of a networked computer that can be mapped to an IP address [Moc87]) of the current

```
1  Received: from mail-qk0-f169.google.com ([209.85.220.169])
2     by mx-2.rz.rwth-aachen.de with ESMTP/TLS/AES128-SHA;
3     07 Nov 2016 14:37:56 +0100
4  Received: by mail-qk0-f169.google.com with SMTP id n21so64861883qk↩
5     a.3 for <[redacted]@comsys.rwth-aachen.de>;
6     Mon, 07 Nov 2016 05:37:56 -0800 (PST)
7  DKIM-Signature: v=1; a=rsa-sha256; c=relaxed/relaxed;
8     d=gmail.com; s=20120113; h=mime-version:reply-to:from:date:↩
9     message-id:subject:to; bh=0i+V1[...]YJrA=; b=bb1p9[...]n0Bw==
10 X-Google-DKIM-Signature: v=1; a=rsa-sha256; c=relaxed/relaxed;↩
11    d=1e100.net; s=20130820; h=x-gm-message-state:mime-version:↩
12    reply-to:from:date:message-id:subject:to; bh=0i+V1[...]YJrA=;↩
13    b=hTvXs[...]aMA==
14 X-Gm-Message-State: ABUng[...]DCw==
15 X-Received: by [redacted] with SMTP id a10mr6457197qkh.66.1478525↩
16     874807; Mon, 07 Nov 2016 05:37:54 -0800 (PST)
17 Received: by [redacted] with HTTP; Mon, 7 Nov 2016 05:37:54 -0800↩
18    (PST)
19 Reply-To: [redacted]@gmail.com
20 From: [redacted] <[redacted]@gmail.com>
21 Date: Mon, 7 Nov 2016 08:37:54 -0500
22 Message-ID: <CADLj[...]2b+9g@mail.gmail.com>
23 Subject: [redacted]
24 To: [redacted] <[redacted]@comsys.rwth-aachen.de>
```

Listing 3.1 Information contained in email headers provides MailAnalyzer with different opportunities to detect exposure to cloud-based email services.

and the previous email server in the delivery chain as well as a human-readable timestamp (cf. Lines 1 to 6 in Listing 3.1).

The complete set of received lines in an email enables us to derive the complete path of email servers that this email traversed. Hence, we can use the set of corresponding IP addresses and hostnames to detect usage of cloud services. With respect to utilizing *IP addresses*, most, especially larger, cloud infrastructure services publish the IP addresses they use, e.g., to allow customers to configure their firewalls. We could retrieve information on used IP addresses for six cloud infrastructures directly from the service. Similarly, all eight cloud-based email security services make their IP addresses publicly available, as their customers must restrict their email servers to only accept incoming emails from these IP addresses.

All cloud-based email providers we study publish the IP addresses they use to send emails for two reasons: (i) to ease whitelisting in firewalls or (ii) to protect against forging of sender names, e.g., using the sender policy framework [Kit14]. For cloud-based email hosters, we were able to directly retrieve IP addresses for two of them. In contrast, we were not able to retrieve information on used IP addresses directly from the service for all five cloud-based email marketing services, three email hosters, and four cloud infrastructures. Only in these cases, we looked-up the autonomous system number(s) [HB96] used by these services and retrieved the associated IP address ranges from the border gateway protocol (BGP) information provided by `ipinfo.io` and `radb.net`. In the end, we were able to retrieve information on the utilized IP addresses for all 31 cloud services.

Similar to IP addresses, some cloud-based email services also publish the *DNS hostnames* they use. However, this fraction of services is significantly smaller. Hence, we require a different approach to obtain information on used hostnames. To this end, we augment the information we were able to retrieve directly from services with information from SenderBase [Cis16] and thus are able to retrieve the hostnames used by all 31 cloud services under study. In the context of our study, we consider hostnames to be more reliable than IP addresses, as they are more stable over time.

Notably, the standard defining the Simple Mail Transfer Protocol (SMTP) used for sending emails forbids removing or modifying any received lines from an email header [Kle08]. While email servers can violate this standard, effective countermeasures are widely deployed today [BE13]. We hence assume that the information in email headers has not been tampered with.

Custom Header Fields. Besides explicitly standardized header fields, email clients and servers can also include arbitrary custom header fields [Res01]. Typically, these custom header fields are prefixed with "X-" (cf. Lines 10 to 14 in Listing 3.1) and are utilized especially by larger email services. Furthermore, header fields initially used by only one email service, e.g., DomainKeys Identified Mail (DKIM) signatures [CHK11], emerged into now standardized and more widely deployed header fields. Such header fields are nowadays used by more than one email service and hence their mere existence does not directly point to a specific email service. Still, these header fields are valuable as they often contain information on the email service that added them (cf. Lines 7 to 9 in Listing 3.1). To identify custom header fields, we manually clustered the header fields present in a subset of our datasets and distilled those header fields unique to a cloud service. As a result, we were able to retrieve custom header fields for seven cloud services, which are mostly email providers.

Sender and Receiver Information. Each email contains information on the sender and the receiver(s) of this email (cf. Lines 19, 20, 22, and 24 in Listing 3.1). While this information is not reliable (it can easily be spoofed), it provides a visible indicator for cloud usage. For example, if a user receives an email from an `@gmail.com` address, she assumes that this email has been processed by Google's email servers. Although we cannot use sender and receiver information to detect cloud usage (due to its unreliability), we can use it to decide whether detected cloud usage is hidden from the user. Sender and receiver information are especially relevant for email providers, as the provider is visible in the email address. We manually identified the hostnames of email addresses used by all six email providers in our study. Additionally, we use the hostnames collected for all 31 cloud services to detect associated senders and receivers. This approach is very optimistic and can lead to false positives. As we use senders and receivers merely to preclude hidden cloud usage, false positives will only lower the fraction of hidden cloud usage. Hence, we still retrieve a lower bound for the prevalence of hidden usage of cloud-based email services.

3.2.2.2 Limitations

Our methodology for quantifying the prevalence of cloud computing by matching patterns in headers of received emails with information on cloud services is limited

in three ways. First, our approach is inherently restricted to *incoming emails*. As we rely on header information inserted by cloud services, our method cannot be used to detect usage of cloud services in outgoing emails. To partly account for this, our active measurements (cf. Section 3.2.3) uncover the cloud usage when sending emails, e.g., the email servers processing the complete set of .com/.net/.org domains. However, emails typically traverse multiple servers and from the outside we can observe only the first hop. Without the cooperation of the receiver of an email, this limitation likely cannot be solved.

Second, *detection patterns can change* over time. Hence, the patterns we derived to detect cloud usage might not be accurate for the past. However, we observe that information on hostnames and custom header fields remain relatively constant over time. With respect to IP addresses used by cloud services, we observed in past years (for big infrastructure providers), that their IP address ranges constantly grow and previously used IP addresses are not abandoned. To further account for this limitation, we randomly sampled a small subset of very old emails from our mailing lists dataset to verify that no false positives occurred. Finally, we restrict ourselves to a *limited set of 31 representative cloud services*. Enlarging this set is technically possible but requires manual curation of cloud services IP addresses and hostnames. Furthermore, we remark that service popularity can differ between different regions (geographic bias in data and patterns). To verify that our selection of services is representative, we manually checked undetected hostnames, custom header fields, and sender names for our mailing lists dataset to ensure that we did not miss any widely used service.

3.2.3 Prevalence of Cloud Email Infrastructures

We begin our analysis of the cloud usage of emails by assessing the prevalence of cloud services in the global email infrastructure, i.e., the share of email servers hosted in the cloud, contacted when *sending* email. To this end, we perform two large-scale active measurements.

Email Servers Running on Cloud Infrastructure. Our first measurement aims at assessing all publicly reachable email servers. This study utilizes a trace of a port scan on Port 25/TCP used by the email protocol SMTP performed on November 19, 2016, covering the entire IPv4 address space and subsequently grabbing SMTP banners [DAM+15]. Out of 16.3 million reachable IP addresses, we classify 6.4 million as valid SMTP servers indicated by a valid `250` status code in the SMTP EHLO banner sent by the server.

We then apply our collection of cloud *infrastructure* IP address ranges to identify email servers hosted by the ten most important cloud infrastructure providers. Our results in Figure 3.2 show that 1.44 % (93 k IP addresses) of the email servers on the Internet are operated in the networks of these cloud infrastructure providers. Notably, 60.13 % (56 k IP addresses) of these servers are operated on infrastructure provided by Amazon. These results indicate that cloud infrastructure is indeed utilized to provide email services. However, their footprint in terms of IP addresses is rather small and unlikely to serve as a proxy for usage or popularity.

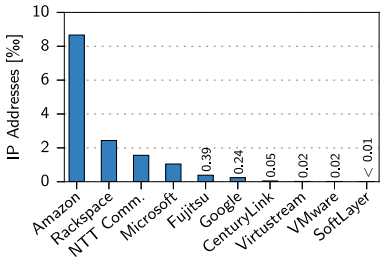

Figure 3.2 Cloud usage among all publicly reachable SMTP servers (in permil).

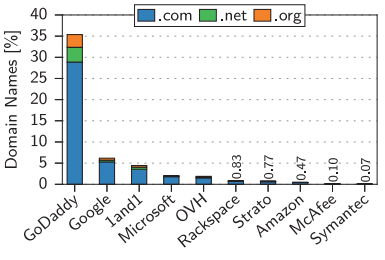

Figure 3.3 Cloud usage among all .com/.net/.org domains (in percent).

Cloud Usage by .com/.net/.org Domains. While the first measurement assesses the cloud usage of all publicly *reachable* email servers, it does not identify whether the identified IP addresses are actually in *use* for receiving email. That is, while the previously identified IP addresses are publicly reachable email servers, they do not necessarily have to be configured by any domain as mail exchange (MX) to actually receive email. To answer the question on the actual usage of cloud-based email infrastructure for receiving email, we performed a second measurement querying the MX DNS records of the complete set of 154 million .com/.net/.org domains (DNS zone files provided by Verisign and the Public Interest Registry) on Nov 20, 2016.

We obtained MX records for 140 million domains, while 1.2 million records were invalid and 12.8 million suffered from authoritative name server errors or timeouts. Out of the obtained 31.9 million distinct MX records, 30.6 million records could be resolved to 2.8 million distinct IP addresses. We remark that the number of detected IP addresses is lower as compared to our first measurement, since (i) not the entire DNS space was scanned and (ii) not every IP must be configured to act as MX. The intuition behind this measurement setup is that any email server configured as MX in the DNS is indeed intended to receive email.

Given this additional DNS information, we are now able to match IP addresses and hostnames against the set of 31 cloud-based email providers listed in Table 3.1 In Figure 3.3, we show the relative share of domains being served by email servers of one of these 31 cloud-based email services for all 154 million .com/.net/.org. Our results show that, in total, 52.27 % of the probed domains use a cloud-based email service. These numbers are largely dominated by GoDaddy, which accounts for 35.36 % of the domains served by a small number of servers (only 1732 distinct IP addresses for our vantage point). While the extent of GoDaddy's dominance surprises, the general trend is reasonable since GoDaddy is the world's largest domain registrar and also often used by domain parkers, i.e., people registering domains to sell them later on and not actually intending to use them (e.g., to receive email) [ME10].

The other widely used services are the all-purpose services Google and Microsoft, email hosters (1&1, OVH, Strato), cloud infrastructure providers (Rackspace, Amazon), and email security services (McAfee, Symantec). The dominance of Amazon in our first IP-based measurement is not reflected in our DNS measurement, i.e.,

Dataset	Period	Emails	Public	Comments
Mailing lists	**01/95–09/16**	**22 930 801**	●	—
Apache	*02/95–09/16*	*15 516 752*	●	*1507 open source lists*
Dovecot	*07/02–09/16*	*115 007*	●	*3 open source lists*
FreeBSD	*01/95–09/16*	*3 654 624*	●	*160 open source lists*
IETF	*01/95–09/16*	*2 043 606*	●	*949 standardization lists*
openSUSE	*05/06–09/16*	*1 600 812*	●	*85 open source lists*
WikiLeaks	**09/07–07/16**	**254 476**	●	—
AKP	*11/09–07/16*	*231 388*	●	*Internal emails*
DNC	*01/15–05/16*	*15 848*	●	*Internal emails*
Podesta	*09/07–03/16*	*7 240*	●	*Internal emails*
SPAM	**02/07–09/16**	**7 788 560**	○	**non-public SPAM traps**
Users	**10/01–09/16**	**873 587**	○	**emails of 20 users**

Table 3.2 We assembled different datasets of emails ranging from mailing lists to private emails of 20 volunteers that participated in a user study, in total accumulating to 31.85 million emails (number of emails obtained after cleanup).

Amazon is often used to host email servers in the cloud, but these email servers are not configured as MX for a large fraction of the tested domains. Further, email for a large number of domains can be handled by only a small number of public IP addresses. Subsequent infrastructure (e.g., email forwarded to another server after processing by a cloud-based security service) is not visible in this analysis since the analyzed MX records denote the *first* server hit when sending email to a domain. Our DNS analysis shows that an email sent to a random .com/.net/.org address has a more than 50 % chance to end up in the cloud.

This first study provides a broad assessment of the *prevalence* of cloud services in the global email infrastructure. It shows that scanning by IP addresses reveals a different cloud provider distribution than probing the DNS. However, it does not provide indications of usage *frequencies* or service popularities, which motivates us to analyze exchanged emails in our second study.

3.2.4 Real-World Cloud Usage of Received Emails

To understand the usage *frequencies* of cloud-based email infrastructure and hence impact on the privacy of users, we set out to detect cloud usage by applying MailAnalyzer to received emails. In the following, we first describe how we assemble different datasets of in total 31 million emails. We then apply MailAnalyzer to these emails to study the cloud usage of individual emails and uncover the hidden usage of cloud-based email services.

3.2.4.1 Datasets

A thorough study of the prevalence of cloud computing among email users requires the analysis of a sufficiently large set of exchanged emails. We, therefore, base our

analysis on a set of 31.85 million emails exchanged between 1995 and 2016, obtained from public mailing list archives, SPAM traps, WikiLeaks, and 20 volunteer users—covering a diverse user base. Since these datasets partly begin *before* the emergence of cloud computing, we can observe its growing adoption from the very beginning.

For our analysis, we only consider standard conform emails [Kle08], i.e., emails containing the mandatory message ID and date header fields. Furthermore, we only consider emails with at least one received line. By doing so, we eliminate emails only consisting of error messages. We summarize key characteristics of our datasets in Table 3.2 (number of emails obtained after cleanup).

Mailing lists. We downloaded the public mailing list archives from the Apache Software Foundation, Dovecot, FreeBSD, the Internet Engineering Task Force (IETF), and openSUSE. These emails mainly contain discussions and announcements regarding open source development and standardization efforts.

WikiLeaks. This dataset contains formerly private emails that have been made public by WikiLeaks [Wik16]. These emails originate from the Turkish Justice and Development Party (AKP), the US Democratic National Committee (DNC), and Hillary Clinton's campaign chair John Podesta.

SPAM. In this dataset, we combine emails collected by various SPAM traps (i.e., inboxes intentionally created to only receive SPAM) since 2007 [HGC12, SHKV14].

Users. We recruited 20 volunteers (mostly with a technical background) from Germany who agreed to run MailAnalyzer on their personal and (partly) professional emails. Besides communication with other people, these emails also contain automatically generated emails such as newsletters, commit messages, and SPAM.

Parts of our datasets are inherently biased to contain significant cloud usage when the *recipient* of the emails uses a cloud-based email service herself. We cope with this bias by ignoring those cloud services that have been used to receive the emails under study. Hence, we ignore AppRiver for *WikiLeaks DNC*, Google for *WikiLeaks Podesta*, and 1&1 for *SPAM*. Furthermore, we blacklist Google for *SPAM*, as we observed massive amounts of faked received lines for Google in this dataset. Finally, we asked our volunteers to blacklist those email services that they used themselves to receive their emails.

Ethical and Privacy Considerations

As we operate on potentially sensitive data of individual users, we designed all our experiments following the basic principles of ethical research [DK12] and of privacy by design [Cav11]. The goal of this work is to understand the prevalence of cloud computing among email users to then inform users about privacy risks, uncover the need for countermeasures, and hence, ultimately, increase privacy for email users. Having this goal in mind, we designed all experiments such that the risk of (inadvertently) harming the privacy of users is minimized. To this end, we excluded exact sender identifiers and the actual content of emails from our analysis. Thereby, we unlink potentially sensitive information from identities. Furthermore, we aggregate all our results in a way that prevents drawing conclusions about individuals.

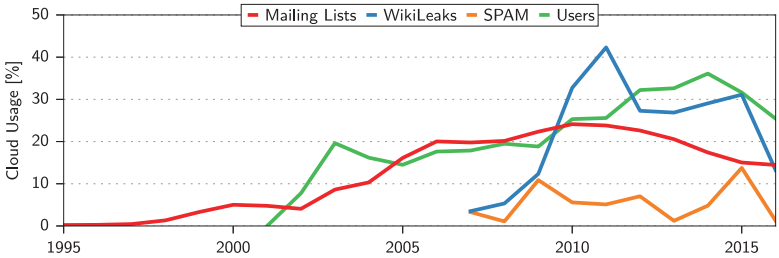

Figure 3.4 In the past, the cloud usage of emails steadily increased to 20–40%, but now shows a remittent tendency with a cloud usage of 15–25% in 2016.

3.2.4.2 Impact of Cloud Computing on Email Users

With MailAnalyzer, we set out to study the impact of cloud computing on email users. To this end, we first explore the exposure of individual emails to cloud services by inspecting the rise of cloud-based email services, identifying the cloud services with the highest usage, and investigating trending email services closer.

The Rise of Cloud-based Email Services

The first question we study is how large the usage of cloud-based email services is and how it has developed over time. To demonstrate this development, we report on the number of emails processed by cloud services per year for each data set in Figure 3.4. We consider an email to be processed by a cloud service if it was processed by an SMTP server [Kle08] of a cloud service covered by our analysis as listed in Table 3.1 on any hop between the sender and the receiver.

When looking at *mailing lists* (by far the largest dataset in our analysis with nearly 23 million emails), we observe that the rise of cloud-based email services first gained traction in the late 1990s with the early email offers of AOL, Microsoft, and Apple. This rise increased in 2004 when Google's Gmail was launched, peaking at 24.12 % in 2010. Since then, we observe a decrease of cloud usage, leading to a usage of cloud email services of 14.45 % in 2016. For the emails of our volunteer *users*, we observe a quite similar trend until 2010, with early-adopters of cloud email leading to a first peak of 19.63 % cloud usage already in 2003. In contrast to the mailing lists dataset, cloud usage of our volunteers continues to grow beyond 2010 to 36.11 % in 2014 before surprisingly dropping to 25.41 % in 2016.

While the data at our hands does not allow us to derive a definitive reason for this observation, one possible explanation is that persons involved in open source development and standardization efforts could be more privacy-sensitive and hence avoid large cloud-based email services. The *WikiLeaks* dataset shows a similar, yet more extreme trend with a peak of 42.31 % cloud usage in 2011. Here, the sudden decrease in cloud usage (to 13.21 % in 2016) can mostly be attributed to a decreasing cloud use in the emails from AKP in 2016. For *SPAM* emails, we assumed a lower

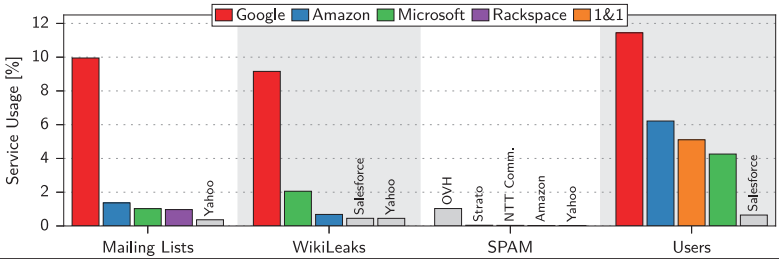

Figure 3.5 The cloud services with the highest usage in 2016 vary between our datasets, but Google plays an important role for the mailing lists, WikiLeaks, and users dataset.

fraction of cloud usage, as cloud-based email service providers have a strong interest in SPAM prevention. Indeed, we see little impact of cloud computing on SPAM emails, far less than in other datasets. The spike for 2015 corresponds to a significant increase in SPAM emails apparently received from the hoster OVH. Overall, we do not observe a significant impact of cloud computing on SPAM emails with a cloud usage of only 1.22 % in 2016.

Cloud Services with Highest Usage

Considering the trend of a rise in cloud email usage, an immediate question is which services contribute most to this cloud usage. To study this question, we consider the usage of *individual* cloud services in each of our datasets for the year 2016. Figure 3.5 shows the fraction of emails exposed to a specific service for each dataset.

For the *mailing lists* dataset, we identify Google as the service with the highest cloud usage: 9.95 % of mailing list emails were processed by Google in 2016. Amazon, Microsoft, Rackspace, and Yahoo already show a notable distance with a usage between 0.37 % and 1.37 %. We make similar observations for *WikiLeaks*, with Google (9.16 %) clearly leading in front of Microsoft (2.06 %) followed by Amazon, Salesforce, and Yahoo, each well below 1 %. Given the overall low cloud usage for the *SPAM* dataset in 2016, the results for the individual services provide limited insight. The top infrastructure used for SPAM, according to our data, is OVH (1.03 %).

For the emails in the *users* dataset, we again observe the highest cloud usage for Google (11.44 %), this time closely followed by Amazon (6.22 %), 1&1 (5.11 %), and Microsoft (4.26 %). The comparable high usage of 1&1 likely corresponds to our users being from Germany, where 1&1 is one of the leading email hosters and providers. The higher usage of Amazon services can partly be attributed to emails sent by Amazon's Simple Email Service, e.g., newsletters and other marketing emails, which naturally are more relevant for the users dataset than, e.g., the mailing lists dataset.

These results highlight that the use of individual cloud services depends on the dataset and, hence, the importance of combining information from different sources to gain a clear picture of the impact of cloud computing on email users. We thus consider all four datasets to derive the most used services for 2016, which provides

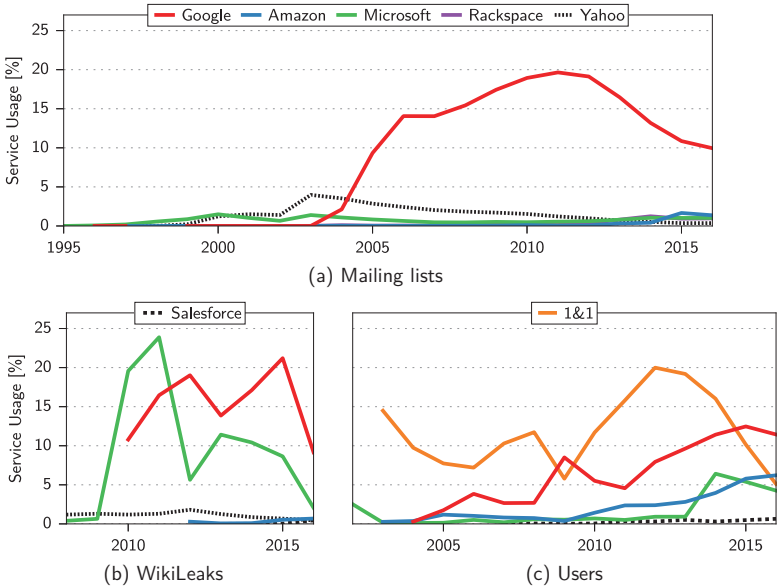

Figure 3.6 The usage of individual cloud services differs between the mailing lists, Wikileaks, and users datasets, but overall a small number of services clearly dominate.

us with Google, Amazon, Microsoft, Rackspace, and 1&1 as the five services with the highest fraction of emails exposed to them across all datasets.

Trending Email Services

Next, we study the question on how these five cloud services with the highest usage in 2016 emerged over time, shown separately in Figure 3.6 for the mailing lists, Wikileaks, and users datasets (we omit SPAM given its low overall cloud usage).

Cloud usage of the mailing lists dataset (Figure 3.6a) is nearly exclusively dominated by Google, surpassing Yahoo quickly after Gmail's launch in 2004. For the Wikileaks dataset (Figure 3.6b), Google and Microsoft are on par, each accounting for more than 20 % of the email traffic in some years and hence a large fraction of users' emails. While the users dataset (Figure 3.6c) initially is dominated by 1&1 (see above), we observe a steady increase for emails from Google and Amazon.

To conclude our study of the impact of cloud computing on email users, we observe a surprisingly high usage of cloud computing for email exchanges. Between 13.21 % (WikiLeaks) and 25.41 % (users) of received emails are processed by at least one cloud service in 2016. Here, it is important to remark that we only account for cloud services that are *not* utilized by the recipient herself (e.g., to host her emails), but for cloud services hit on the way to the recipient. Depending on the dataset, between

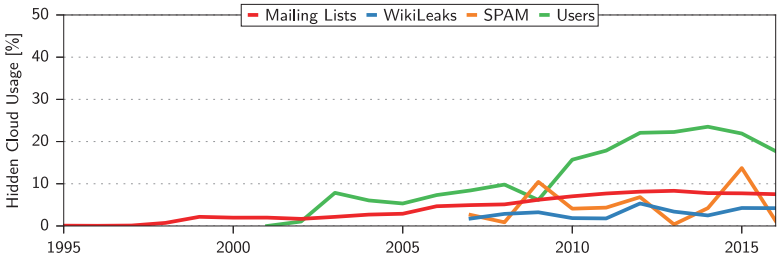

Figure 3.7 Emails with hidden cloud usage among the *total* set of emails. Hidden usage of cloud services follows a similar trend as cloud usage in general.

9.16 % and 11.44 % of received emails are processed by a *single* cloud service in 2016 (most notably Google, Amazon, and Microsoft). Hence, these services learn about a large fraction of the users' email communication. In this situation, MailAnalyzer supports users by uncovering a source for potential privacy risks.

3.2.4.3 Hidden Usage of Cloud-based Email Services

The usage of cloud email services on the way from the sender to the recipient can be *hidden* to the user. We define the usage of a cloud service as hidden if this cloud service is not obviously used as the email provider of the sender or any recipient, i.e., the cloud service cannot be inferred from email addresses in the sender or recipient fields. For example, if any sender or recipient address ends with `@gmail.com`, the usage of all services attributed to Google is *not* hidden. Hidden usage of cloud resources can raise privacy concerns, e.g., when communication (meta) data should not be exposed to a third party operator [MSWP14]. We, therefore, aim at understanding the extent to which hidden exposure of emails to cloud services happens and to which services we can attribute the most hidden cloud usage.

General Trend of Hidden Cloud Exposure

Again, we first study the general evolution of hidden cloud exposure over time for our different datasets in Figure 3.7. For each dataset, we plot the overall fraction of hidden cloud usage among the entire set of emails per year. We define cloud usage to be hidden if at least one of the utilized cloud services is neither detectable from the sender field nor from any of the recipient fields.

The hidden cloud exposure for the *mailing lists* dataset shows a steady increase, similar to the overall increase in cloud exposure. In 2016, we observe that 7.53 % of all emails in our dataset use cloud services hidden to the user (see Figure 3.7), which amounts to 52 % of all emails with cloud usage (i.e., 14.45 % of all emails in our dataset, see Figure 3.4).

Similarly, we observe that 70 % of cloud usage remains oblivious to users for the *users* dataset (i.e., 17.72 % emails with hidden cloud usage vs. 25.41 % emails with

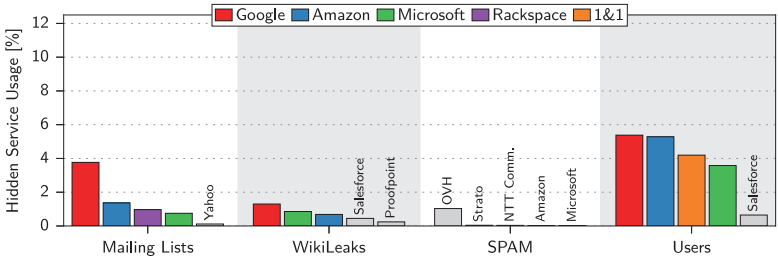

Figure 3.8 Hidden cloud usage across our four datasets mainly results from email hosters and cloud infrastructure offers as well as hybrid cloud-based email offers.

any cloud usage in 2016), raising privacy concerns. For *WikiLeaks* emails, we observe a lower hidden cloud usage than for the mailing lists and users datasets. Given the overall high fraction of cloud usage for the WikiLeaks emails, these results indicate that the cloud usage during this period can mostly be attributed to emails that originate from cloud-based email *providers*. Here, we observe that 32 % of the cloud usage cannot be observed by users (i.e., 4.21 % vs. 13.21 % in 2016). In contrast, for *SPAM* emails cloud usage happens nearly exclusively hidden, as seen by the nearly identical curves in Figures 3.4 and 3.7. This suggests that the cloud portion of SPAM does not originate from (potentially hacked) cloud email accounts but instead from email hosters or cloud infrastructure.

Cloud Services with Highest Hidden Usage

As a large portion of cloud exposure is hidden to users, the immediate question is to which services the most hidden cloud usage can be attributed. We thus study the hidden usage of individual cloud services in 2016 for each dataset in Figure 3.8.

Again, we observe the importance of covering different email sources, as the results for the hidden usage of specific services vary across the datasets. Nevertheless, we can derive what types of cloud email services (cf. Section 3.2.1.1) account for hidden cloud usage: (i) email hosters (e.g., 1&1 with 4.20 % in the users dataset) and (ii) cloud infrastructure (e.g., Amazon with 5.28 % in the users dataset). Furthermore, hybrid services such as Google and Microsoft that offer email hosting and cloud infrastructure have a significant impact on hidden cloud usage. As expected, we do not observe hidden usage of cloud-based email *providers* (e.g., AOL or Comcast) as usage of an email provider can directly be derived from an email address.

In summary, we observe that (less technically proficient) users remain oblivious to the hidden cloud usage of 30 % to 70 % of all emails exposed to the cloud. This hidden usage predominantly originates from email hosters and cloud infrastructure. When *sending* emails, some of this hidden usage could be a priori uncovered by analyzing DNS MX records of the recipient domains. Other cloud exposure (e.g., subsequent use of cloud services behind a security service or forwarding of emails to cloud services) cannot be detected by the sender.

3.2.5 Summary and Future Work

The goal of our work is to provide users with an understanding of their individual exposure to cloud-based email services. This topic is important since the ongoing transformation of the email architecture from a largely decentralized one towards a more centralized one can have consequences for privacy and security. To tackle this problem, we propose MailAnalyzer which uses public information to detect and quantify the usage of cloud-based email services and apply it in two studies. Our first study analyzes email *infrastructures* hosted in the cloud, i.e., servers hit when sending email. We analyzed all publicly reachable email servers obtained by scans of the entire IPv4 address space and by querying the complete set of 154 million .com/.net/.org domains. Our second study then focuses on understanding the *user exposure* to these infrastructures when receiving email by analyzing more than 31 million received emails. From our study results, we can derive three key observations.

First, we observe that exchanged emails tell a different story than infrastructure measurements. With regards to measurement studies, we show the difference between three perspectives on email: (i) size of the public-facing infrastructure (i.e., number of SMTP IP addresses hosted in cloud infrastructures), (ii) email servers configured for domains (i.e., DNS MX records), and (iii) exchanged emails. All three perspectives provide interesting insights: infrastructure studies yield insights into the adoption of cloud email services, both with respect to the number of email servers in the cloud and the number of hostnames using these servers. In contrast, our analysis of exchanged emails yields insights into the actual cloud exposure experienced by users. Thus, all these perspectives are relevant for future studies.

Second, we observe that users' emails are frequently exposed to the cloud. Between 13.21 % (WikiLeaks) to 25.41 % (users) of all emails received in 2016 were processed by cloud services. Regarding the email infrastructure, our DNS analysis shows that email sent to a random .com/.net/.org address has a more than 50 % chance to end up in the cloud. While the concrete services and their exposure level varies between the datasets (and users), we observe a concentration of few large infrastructures that process substantial fractions of the overall email traffic. This concentration thus opens users' questions on privacy and security implications of email becoming more centralized, i.e., single providers having access to large fractions of the overall set of exchanged emails.

Finally, our results show that the usage of cloud-based email services happens unobservable for users. Surprisingly, for 30 % to 70 % of the emails that are processed by the cloud, this cloud usage is hidden for (less technically proficient) users. That is, this cloud usage cannot be inferred from email addresses, e.g., `@gmail.com`. One reason for hidden cloud exposure is the ability to have a domain's MX record configured to a cloud email server (e.g., email intended for a state-owned university can be managed by a third party cloud operator).

Based on our results, we identify two promising directions for future work. First, when considering the cloud exposure in *received* emails, we can make cloud usage evident to users by implementing MailAnalyzer in email programs, thereby raising their awareness for cloud usage and especially the hidden usage of cloud resources. To

correlate the resulting quantified cloud usage to potential privacy risks, MailAnalyzer could be extended with the possibility to compare identified (hidden) cloud exposure to those of users' peers. We provide a first feasibility study of such a comparison approach in the context of the cloud usage of mobile apps in Section 3.4.

A second direction of future work could be concerned with the cloud exposure when *sending* email. Since email can be transparently forwarded to cloud services, e.g., to security cloud solutions for virus checking by the operator or to private cloud email accounts by the receiver, hidden cloud exposure often cannot be inferred by the sender of an email and hence can only be detected by the receiver through email header analysis. Hence, future work should be concerned with the question on whether email routing and processing should or can be made controllable, e.g., using a privacy policy language such as the one presented in Section 4.2.

Furthermore, future work could address the question of how end-to-end encryption for email and cloud services' access to parts of emails can be combined, similar to the concept of mcTLS for end-to-end encrypted network connections [NSV+15]. For example, by granting security services only access to selected parts of an email (e.g., to perform virus checking on executable attachments) security and privacy concerns could be moderated. Our work to understand the prevalence of cloud email provides the starting point for such highly necessary countermeasures.

By uncovering the cloud exposure of email users, we addressed cloud computing's core problem of technical complexity and missing transparency for email communication as a first prominent deployment domain of cloud services. In the following, we complement these efforts by applying a related approach to study the cloud usage of mobile apps on smartphones.

3.3 CloudAnalyzer: Uncovering the Cloud Usage of Mobile Apps

Smartphones have become an indispensable tool for storing and accessing personal information, ranging from contacts and calendar entries over pictures to work documents [EGH+14]. Additionally, smartphones *produce* data through their sensors which, e.g., enables localization or activity recognition [GCEC12, QG12]. With the right permissions, this abundance of sensitive data can be easily accessed by mobile applications (apps) through dedicated APIs [PHW17]. Indeed, app developers increasingly rely on user data to improve the functionality of their apps or to increase revenue with targeted advertisement [ISKČ11].

At the same time, major parts of apps' backend functionality, including tracking and advertising, are nowadays realized via cloud services [EGH+14,FKB+15]. These services range from cloud infrastructure and content delivery networks (e.g., AWS and CloudFront) over reporting, analytics, and advertisement services (e.g., Crashlytics, Flurry, and AdMob) to consumer services (e.g., YouTube and Facebook). In Section 3.3.3.3, we discover that the most popular apps on Google Play utilize 4.3 cloud services per app on average, which highlights the prevalence of cloud usage.

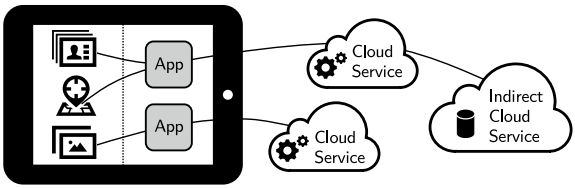

Figure 3.9 Today, mobile apps running on smartphones expose an abundance of private information they can access to cloud services, which often are built upon other cloud services.

While the utilization of these cloud services benefits app developers, users are confronted with severe privacy risks. In contrast to traditional privacy risks of cloud computing (cf. Section 2.3), these issues further exacerbate considering the abundance of privacy-critical data on smartphones [GCEC12, EGH+14]. As we illustrate in Figure 3.9, sensitive information ranging from contact lists over location information to private pictures is accessible by apps and can then be transferred to cloud services. In this situation, users have no knowledge about which cloud services are utilized by apps running on their smartphones. However, combining the sensitive data stored and sensed by smartphones with cloud computing—characterized by de facto monopolies, technical complexity, inherent non-transparency, and opaque legislation—raises severe privacy risks (cf. Section 2.3). Even worse, cloud services can be realized on top of each other, leading to indirect cloud exposure which is even harder for users to grasp. As an example, our work reveals that Unity (a popular game development platform) utilizes Amazon EC2 to (partly) deliver its services.

Any cloud service receiving sensitive information can use it for unintended purposes, e.g., personalized advertising [ISKČ11] or forwarding to other entities, which becomes especially problematic since typically multiple cloud services have access to sensitive information forwarded by apps on smartphones. Furthermore, users have no guarantee that their data is handled according to legal requirements [ISKČ11]. Resulting from the de facto monopolized landscape of cloud services, data is further susceptible to breaches (cf. Section 3.2). To put users back into control, we consider it important to raise their awareness of these risks [MPS+13] and provide them with the information required to take appropriate measures to protect their privacy.

Related work confirms the privacy risks of the access of apps to an abundance of private information. To assess and counter these risks, approaches presented in related work aim at detecting privacy leakage by analyzing traffic [SH15, RRL+16] or tracking apps' data flows [ARF+14, EGH+14]. These streams of related work provide information on *what* data is leaked. So far, a way for smartphone users to detect *where* (to which cloud services) their data is leaked by the apps on their smartphones, as a foundation to protect their privacy, is missing.

To bridge this gap between users' knowledge and information required to enforce their privacy, we present CloudAnalyzer, which provides users with detailed statistics of their personal cloud exposure caused by their smartphone apps. CloudAnalyzer *locally* monitors the network traffic produced by apps running on a user's device and compares observed communication patterns to 55 representative cloud services.

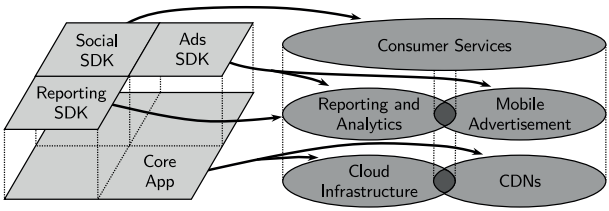

Figure 3.10 In the landscape of mobile cloud services, services on upper layers can, but not necessarily have to, rely on services on lower layers to provide their functionality.

Apart from revealing the exposure to cloud services caused by smartphone apps, CloudAnalyzer also detects the prevalent indirection in cloud usage where cloud services subcontract each other to realize their functionality. Based on CloudAnalyzer's observations, we support users in critically reviewing their exposure to cloud services and, as a result, change their app usage behavior or even decide to refrain from using certain apps. Likewise, CloudAnalyzer is a valuable tool for researchers to understand the characteristics of the usage of cloud services by smartphone apps and the relationships between cloud services.

CloudAnalyzer is available for users of Android devices via the Google Play store[3]. Furthermore, we provide access to its source code as well as to the detection patterns for mobile cloud services under the GNU GPL license (version 3)[4].

3.3.1 Mobile Cloud Services and Privacy

Developers for mobile platforms increasingly rely on cloud services [XEG+11]. Their motivation ranges from reduced effort over cost reductions to the possibility to integrate third party services, e.g., advertising networks. We first provide an overview of the landscape of mobile cloud services and derive a representative set of services. Based on this analysis, we distill privacy risks in the face of potentially sensitive data collected by smartphones and discuss related work.

3.3.1.1 The Landscape of Mobile Cloud Services

To understand the extent of cloud exposure through mobile apps and the resulting privacy risks, we identify classes of mobile cloud services and their interweaving. As shown in Figure 3.10, a major portion of cloud usage originates from SDKs that app developers include to realize functionality ranging from interaction with social networks over crash reporting to targeted advertisement [BHJ+14]. Depending on the individual SDK, different cloud services are utilized.

In the following, we discuss five different classes of mobile cloud services we identified and point our their relationships. Furthermore, we compile a representative list of

[3] `https://play.google.com/store/apps/details?id=de.rwth.comsys.cloudanalyzer`
[4] `https://github.com/COMSYS/CloudAnalyzer`

the most influential services for each class. We provide the full list of the 55 cloud services that we selected in Table 3.3.

Cloud Infrastructure (CI). Developers of mobile apps use cloud infrastructure (i.e., computing and storage resources) to operate their apps' backends (instead of using own servers). In our work, we consider the most important infrastructure providers as identified by Canalys' revenue analysis [Can17] and Skyhigh's study of application deployment [Sky16]. The services covered by these studies account for a market share of 68.7 % respectively 85.7 %. Both studies agree that Amazon and Microsoft jointly dominate the market, with a combined market share of more than 50 %.

Content Delivery Networks (CDN). To reliably, scalably, and timely deliver static content, content delivery networks (CDNs) rely on globally distributed infrastructure. They can either be realized on top of cloud infrastructure or built on dedicated infrastructure. We analyze all CDNs that have a market share of more than 1 % in Datanyze's measurements of 1 million popular websites [Dat17a]. These measurements identify Amazon CloudFront as the most widely used CDN, followed by KeyCDN, Cloudflare, and Akamai. Together, the CDN services in our analysis have a market share of more than 90 %.

Reporting and Analytics (R&A). To support app developers with statistics on errors and app usage, reporting services track errors (e.g., crashes) of apps while analytics services gather statistics on the usage of apps (ranging from gathering user statistics to tracking user interaction). We cover all services behind the reporting and analytics libraries with more than 1 % of installations according to AppBrain's measurements [App17b, App17c]. Libraries that do not operate own cloud services are excluded from our analysis (this is the case, e.g., for ACRA). The most influential crash reporting service is Crashlytics with 11.6 % of installations, while Flurry is the leading analytics service with 16.9 % of installations.

Mobile Advertisement (MA). App developers often rely on mobile advertisement services to monetize their apps [SDW12]. These services are usually realized on cloud infrastructure and/or CDNs. In our work, we include the services of advertisement network libraries with more than 1 % of installations in AppBrain's statistics [App17a, App17e]. In addition, we incorporate the advertisement companies with the highest traffic share as derived from a measurement study of Pujol et al. [PHF15], i.e., AppNexus and Criteo.

Consumer Services (CS). Services directly addressing and interacting with consumers, e.g., social networks or communication and video platforms, often rely on cloud infrastructure and CDNs. Such consumer services (e.g., Facebook and Twitter) can often be integrated into apps through an SDK. To capture this effect, we include the social network libraries with more than 1 % of installations according to AppBrain [App17d] into our analysis. Furthermore, we cover the services with the highest amount of mobile traffic in North America according to Sandvine [San16a]. Finally, we incorporate the 20 most prominent consumer services ranging from Facebook over Flickr to Evernote as identified by Skyhigh [Sky16].

Service	Source(s)	CI	CDN	R&A	MA	CS	Additional Brand Names
AdColony	[App17a] [App17e]				●		
Adjust	[App17a] [App17b]			●	●		
Akamai	[Dat17a]		●				
Alibaba	[Can17]	●	○	●		○	Umeng
Amazon	[App17a] [Can17] [Sky16]	●	●	○	●	●	Amazon Mobile Ads, Amazon S3, Amazon Web Services (AWS), Cloudfront, Twitch
Appboy	[App17d]			○		●	
Apple	[San16a]					●	iCloud, iTunes
AppLovin	[App17a]			○	●		
Appnext	[App17a]				●		
AppNexus	[PHF15]				●		
AppsFlyer	[App17a] [App17b]			●	●		
Apteligent	[App17c]			●			Crittercism
Chartboost	[App17a] [App17e]				●		
Cloudflare	[Dat17a]		●				
comScore	[App17b]			●			ScorecardResearch
Criteo	[PHF15]				●		
Dropbox	[Sky16]					●	
Evernote	[Sky16]					●	
Facebook	[App17d] [San16a] [Sky16]				○	●	Atlas, Instagram, Facebook Messenger, WhatsApp
Fastly	[Dat17a]		●				
GitHub	[Sky16]					●	
Google	[App17a] [App17c] [App17d] [Can17] [San16a] [Sky16]	●	○	●	●	●	AdMob, Crashlytics, DoubleClick, Fabric, Gmail, Google Analytics, YouTube
imgur	[Sky16]					●	
Incapsula	[Dat17a]		●				
InMobi	[App17a]				●		
KeyCDN	[Dat17a]		●				
Kochava	[App17a] [App17b]			●	●		
Leadbolt	[App17a]				●		
LinkedIn	[Sky16]					●	
Localytics	[App17b]			●			
Microsoft	[App17c] [Can17] [Sky16]	●	●	●	○	●	Bing, HockeyApp, Microsoft Azure, Office, OneDrive, Outlook, Skype
Mixpanel	[App17b]			●			
Netflix	[San16a]					●	
Oracle	[Can17]	●			○		
Pinterest	[Sky16]					●	
Rackspace	[Dat17a] [Sky16]	●	●				
RNTSMedia	[App17a] [App17d]				●	●	Fyber, HeyZap
Smaato	[App17a]				●		
Snap	[San16a]					●	SnapChat
SoftLayer	[Can17] [Sky16]	●	○		○		
SoundCloud	[Sky16]					●	
StackPath	[Dat17a]		●				Highwinds, MaxCDN
StartApp	[App17a] [App17d] [App17e]			○	●		
StumbleUpon	[Sky16]					●	
Supersonic	[App17a]			○	●		IronSource, mobileCore, StreamRail
Tapjoy	[App17a]				●		
Tune	[App17a] [App17b]			●	●		MobileAppTracking
Twitter	[App17a] [App17d]				●	●	MoPub, Vine
Unity	[App17a] [App17e]			○	●		Applifier
Verizon	[App17a] [App17e] [Dat17a] [Sky16]	○	●	●	●	●	AOL, EdgeCast, Flickr, Flurry, Millennial Media, Nexage, Tumblr, Yahoo
Vimeo	[Sky16]					●	
VK	[App17d]					●	
Vungle	[App17a] [App17e]				●		
WeChat	[App17d]					●	
Yandex	[App17b] [App17c]			●		○	

Table 3.3 Our derived set of 55 cloud services covers the different classes of mobile cloud services. We use ● to denote representative services for each class of mobile cloud services, while ○ denotes less prominent services for this class.

3.3.1.2 Privacy Risks of Mobile Cloud Services

When considering the landscape of mobile cloud services, we observe that the challenge of protecting privacy is more complex and important for cloud-based app compared to traditional deployments [PHW17]. First, smartphones are equipped with a large number of sensors, facilitating detailed monitoring and tracking [GCEC12]. For example, by reading the GPS sensor, an app can accurately derive and track the position of the smartphone user. Second, users interact with their smartphones throughout the day, leading to a growing amount of sensitive information and metadata [GCEC12]. Thus, smartphones increasingly cover important aspects of private life and protecting against the leakage of private information is important for a wide range of users. When outsourcing potentially sensitive data to cloud services, these privacy risks further amplify—mainly due to the centrality, technical complexity, non-transparency, and opaque legislation of cloud computing (cf. Section 1.1.3).

Modern computing power—as it is made readily available by cloud services in abundance today—allows processing large amounts of information collected from smartphones near real-time, e.g., multiple sources of information can be combined to create complex profiles of individual users [EGH+14]. Thus, a messenger app can not only keep track of with whom its users are communicating but additionally rely on GPS information to also derive from where users are communicating [PHW17]. Most notably, since more and more tasks—ranging from shopping over maintaining the calendar to the tracking of fitness and health—are realized on smartphones [EGH+14], private information stored on, processed by, and sensed from smartphones becomes evermore valuable and hence requires protection. Such valuable information is one key reason for developers of smartphone apps to ignore the privacy of their users [PFNW12]. Because of the huge competition in the market for smartphone apps, apps are often offered for free and monetized through advertisements [SDW12]. Here, access to personal information allows app developers to increase their revenue since advertisers pay more for personalized advertisement instead of presenting the same advertisement to every user [PFNW12].

As a result of these privacy risks, users perceive a loss of control over their data when their sensitive data is sent to cloud services (cf. Section 1.1.3). Hence, providing users with the information required to quantify this loss of control as a foundation to take appropriate countermeasures is an important challenge.

3.3.1.3 Related Work

Different lines of research provide valuable input for our goal to uncover cloud usage of apps. We classify related work into approaches studying (i) mobile network traffic, (ii) cloud traffic, (iii) mobile advertising, and (iv) data flow tracking.

Mobile Network Traffic. Xu et al. [XEG+11] study the usage behavior of apps in a cellular network. ProfileDroid [WGNF12] studies Android apps to understand their network behavior. Freudiger [Fre15] studies the WiFi probe requests of mobile devices to quantify resulting location privacy risks. AntMonitor [LVL+15] and Haystack [RVS+16] realize mobile measurement platforms that enable researchers to

investigate the network usage of apps at large scales. Envisioned use cases of these platforms include network classification and the detection of privacy leaks. With the goal to detect leaked private data, PrivacyGuard [SH15] and ReCon [RRL+16] intercept network traffic of apps. They show that it is possible to detect the leakage of private information such as a device's IMEI (globally unique identifier of a phone) or location purely by observing network traffic. Ferreira et al. [FKB+15] study the network behavior of apps to differentiate between (in)secure connections and the location of communication endpoints.

These works focus on the patterns and content of apps' network communication (and partially on resulting privacy risks). They provide a solid foundation for our work since they derive an understanding of the network-level behavior of mobile apps and offer mechanisms to detect leaked private data in network traffic. In contrast to our work, these works neglect the added privacy risks of the complex and non-transparent interweaving of mobile apps with cloud services common today.

Cloud Traffic. Bermudez et al. [BMM+12] identify DNS responses as viable input to identify cloud services. Subsequently, they detect network traffic flowing to Amazon Web Services [BTMM13]. Drago et al. [DMM+12] rely on DNS and transport layer security (TLS) packets to study cloud storage systems based on passive network observations. To understand if and how web services are realized on top of cloud infrastructure, He et al. [HFW+13] perform DNS probing for popular web services. These works perform large-scale measurements to understand the anatomy of cloud services and their methodology provides valuable input to detecting cloud usage on smartphones. However, these approaches do not consider the privacy risks of smartphones communicating with cloud services, which is our main focus.

Mobile Advertising. Vallina-Rodriguez et al. [VSF+12] study mobile advertising based on network traffic observed within the network of a mobile carrier. Focusing on advertisement *libraries* on Android, Book et al. [BPW13] analyze the use of permissions for mobile advertising. From a different perspective, Chen et al. [CUKB14] investigate the privacy risks of mobile analytics services. Seneviratne et al. [SKS15] focus on the privacy risks of paid apps. Complementing these works, Vallina-Rodriguez et al. [VSR+16] study mobile advertising and tracking based on network traces of volunteers. Finally, Brookman et al. [BRAY17] measure the capability of advertisers to link users *across* different devices. These works highlight privacy risks of forwarding data to advertising services. However, mobile advertising is only one part of the mobile cloud landscape and privacy risks further exacerbate when looking at the *complete* mobile cloud landscape.

Data Flow Tracking. Tracking the flow of data within smartphone apps allows to detect the leakage of sensitive data to third parties, even if apps try to obfuscate that they are sending out sensitive data. AndroidLeaks [GCEC12] and FlowDroid [ARF+14] are *static* flow tracking systems that are used ahead of time to detect potential leaks of sensitive information by covering all possible execution paths of an app. In contrast, *dynamic* flow tracking systems, such as TaintDroid [EGH+14] and TaintART [SWL16], track data flows during execution of an app to identify actual data leakage that occurs while executing an app. One challenge of data flow tracking is to identify whether an identified data flow is benign or constitutes a privacy risk.

To this end, AppIntent [YYZ+13] identifies data flows that have not been triggered by the user and marks those as critical.

Mobile operating systems today counter privacy risks by measures ranging from access control to sandboxing [Ele14, ADD+14]. These protect against malicious apps, but do not prevent privacy invasive apps from exploiting *granted* permissions. Hence, users' privacy is insufficiently protected [SSY+16], especially since users remain oblivious of their exposure to a plethora of cloud services. Related work that addresses this challenge primarily focuses on detecting which private *content* is leaked from smartphones [YYZ+13, EGH+14, SWL16]. In contrast, we study the privacy risks resulting from the *destination* of leaked content, especially in the context of cloud computing.

3.3.2 Detecting Cloud Usage of Apps

Given the privacy risks when data is sent from smartphones to the cloud, users must be empowered to effectively assess these risks to make an informed decision about which apps to use or not. To this end, users need detailed information about the quality and extent of cloud exposure induced by apps. However, existing approaches today primarily focus on detecting the leakage of sensitive information, irrespective of where data is communicated to. Additionally, cloud exposure of users through their apps is highly individual, depending on the utilized apps and users' behavior when interacting with these apps. Hence, users are in need of an *individual* assessment of the privacy risks resulting from the cloud usage of their apps.

To achieve this goal, we present CloudAnalyzer, our transparency approach that uncovers the cloud usage of smartphone apps by passively observing network traffic directly on users' devices. Consequently, we neatly complement existing work, especially on data flow tracking, since we enable the attribution of privacy leaks to responsible cloud services. This attribution empowers users to adequately assess their individual privacy risks and take appropriate countermeasures, e.g., uninstall a certain app or change their usage behavior.

In the following, we first describe the overall architecture of CloudAnalyzer. We then present our methodology for dissecting network traffic to detect cloud usage and describe how we realize CloudAnalyzer on off-the-shelf Android devices.

3.3.2.1 System Overview

CloudAnalyzer operates on network traffic of smartphone apps to detect communication with cloud services. We realize all functionality for uncovering cloud usage solely within the control of the user, i.e., directly on her device. Since network traffic itself is extremely sensitive, processing it outside users' control would strongly contradict our goal of *improving* user privacy.

Our system for uncovering cloud usage of smartphone apps, CloudAnalyzer, operates as shown in Figure 3.11. Whenever an app uses one of the communication interfaces

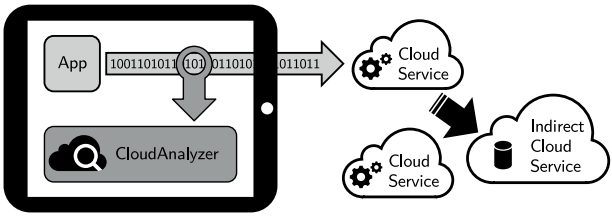

Figure 3.11 To uncover cloud usage, CloudAnalyzer analyzes network traffic created by apps directly on users' smartphones to detect communication with cloud services.

(cellular or WiFi) of the smartphone to contact an Internet service, CloudAnalyzer *locally* obtains a copy of the network traffic. Subsequently, CloudAnalyzer dissects the captured traffic to identify contacted cloud services based on properties of network traffic. Based on this information on contacted cloud services, CloudAnalyzer attributes the complete communication flow to one or multiple identified cloud services. CloudAnalyzer collects aggregated statistics on the number of network packets and the amount of traffic that has been sent to and received from a specific cloud service, thereby differentiating between the direction of communication, encrypted and unencrypted communication, user-initiated and background traffic, as well as the used communication interface (cellular or WiFi).

3.3.2.2 Dissecting Traffic to Detect Cloud Usage

At the core of CloudAnalyzer sits our methodology to detect cloud usage based on network traffic. As the foundation of this methodology, we comprehensively analyze the communication behavior of smartphone apps to derive different approaches for reliably identifying contacted cloud services.

IP Addresses. IP addresses are identifiers assigned to each networked computer [Pos81] and hence also to each server that is used to realize cloud services. Hence, IP addresses can be used to identify the operator of the infrastructure a service is realized on (cloud infrastructure or CDN, cf. Section 3.3.1.1). To determine that a contacted server is operated by a cloud service, we rely on information supplied by cloud providers: Many cloud services (e.g., Amazon, Microsoft, Google, SoftLayer) make their IP addresses public, e.g., to enable customers to configure firewalls (cf. Section 3.2.2.1). Often, such published information contains a (textual) description of the location of the data center, enabling us to also identify the corresponding jurisdiction. While IP addresses often allow us to detect infrastructure services, we additionally have to analyze application layer protocols to also detect services that fail to publish their IP addresses as well as services realized at higher layers, i.e., on top of cloud infrastructure.

DNS Responses. The domain name system (DNS) translates (human readable) domain names to IP addresses [Moc87]. Whenever a smartphone app requests a resource from a specific domain name, the Android system transparently issues a DNS request to translate this domain name to an IP address. By observing subsequent

DNS responses from a DNS name server, we derive the actual contacted service(s) [BMM+12, DMM+12]. We mark all subsequent communication with this IP address as belonging to the identified cloud service. Using this approach, we are even able to identify multiple services in the case of indirect cloud usage. Furthermore, some cloud services (e.g., Amazon) use domain names that contain information about the data center location, easing the detection of the applicable jurisdiction for data sent to this cloud service.

Server Name Indication. With the increasing use of encryption, server name indication (SNI) enables operators to still serve multiple domain names from one IP address. Support for SNI is available for the widely deployed TLS protocol [Eas11] and the evolving QUIC protocol [LC16]. Since clients send the SNI in plaintext, we can observe this information and utilize it, similar to DNS responses, to identify contacted cloud services.

TLS Certificates. When using TLS encrypted connections, servers have to authenticate themselves to clients using a TLS certificate [DR08]. This certificate typically identifies the institution operating a service. To establish trust into certificates, they have to be validated by a trusted certificate authority. Hence, the information in TLS certificates, especially domain names and the owner of the certificate, constitutes an especially reliable source for identifying cloud services.

Detecting Cloud Usage for Traffic Flows

In CloudAnalyzer, we use the above approaches to detect cloud exposure for traffic *flows* as follows. Whenever one of the above approaches detects a cloud service, we mark any future packets of the same traffic flow as being exposed to this cloud service as well. Strictly working on traffic flows prevents false classification that might result from more lenient approaches, such as analysis of traffic patterns. Most notably, the combination the above approaches also enables the detection of indirect cloud usage, i.e., one cloud service realized on top of another. In this case, we assign one traffic flow to more than one cloud service and use the most specific information available on these different cloud services, e.g., when assigning data center locations.

Detecting Usage of Specific Cloud Services

To apply the above approaches to detect *specific* cloud services, we need to create patterns for each cloud service. For example, we need to know which IP addresses a cloud service uses or how a cloud service's TLS certificate looks like. To this end, we researched these patterns for our 55 representative cloud services (cf. Section 3.3.1.1). Here, we relied on information provided by cloud services as well as other public information (e.g., filter lists for advertisement). Subsequently, we verified that our selection of cloud services and detection patterns is indeed representative by checking IP addresses, DNS and SNI domain names, as well as TLS certificates for a random subset of our measurements of the most used apps (cf. Section 3.3.3.3).

Our approach of creating patterns for representative cloud services might not necessarily detect all cloud services. However, given our goal to support users in em-

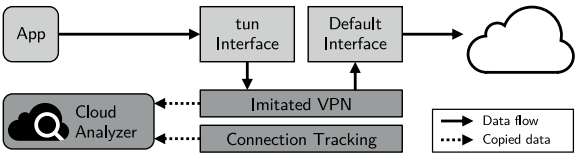

Figure 3.12 CloudAnalyzer accesses network packets by locally imitating a VPN using Android's `VPNService`.

powering their privacy, we strive for correctness over completeness. Our rationale is to keep users clear of incorrect information which might result from probabilistic approaches, such as the topological analysis of autonomous systems [FBL15] or IP geolocation databases [PUK+11]. Instead, the information provided by CloudAnalyzer constitutes a lower bound for the usage of cloud services. In return, we accept that we might not be able to detect cloud exposure to a few less important and seldom used cloud services. Additionally, cloud services might deliberately try to obfuscate their network communication. However, during our tests of CloudAnalyzer, we observed only a single attempt to obfuscate a mobile advertising service.

3.3.2.3 Integrating CloudAnalyzer into Android

The core idea of CloudAnalyzer is to detect the usage of cloud services based on network traffic. Since network traffic itself is of sensitive nature, we consider it imperative to realize CloudAnalyzer directly on users' devices. However, mobile operating systems such as Android, in contrast to traditional operating systems, lack an interface to access network traffic without system modification (i.e., rooting the device or installing custom firmware). Since we mostly target non technically-minded users, we cannot dictate modifications to the operating systems in contrast to related work [EGH+14, SWL16]. Instead, we aim at a solution that enables users to uncover their cloud exposure simply by installing an app through well-established channels (e.g., using the Google Play store).

To achieve this goal, we use an indirect path to access network traffic on *unmodified* Android devices: We realize an *imitated virtual private network (VPN)* to gain access to the device's network traffic using the `VPNService` of the Android SDK [RVS+16, LVL+15, SH15]. As shown in Figure 3.12, the `VPNService` enables us to create a `tun` interface that redirects all network traffic of the Android device into our imitated VPN. The imitated VPN receives raw IP packets and performs two tasks: (i) it creates a copy of each received network packet which is then forwarded to CloudAnalyzer for further processing and (ii) it forwards the raw IP packets to their original destination.

The latter proves technically difficult since Android prohibits the creation of raw sockets. Hence, we implement the essential parts of a Layer 3 and 4 network stack to forward data from the `tun` interface over a normal Java socket to an Internet host. This approach includes memorizing the state of all open connections to be able to re-translate payload *received* on a socket to corresponding IP packets to send them

back to the application over the `tun` interface. Related work shows that this can be realized at modest throughput and energy costs [RVS+16, LVL+15, SH15], which we verified through independent measurements.

Besides protecting privacy, capturing and analyzing network traffic directly on users' devices gives us an additional advantage: It allows correlating network packets to the application they originate from. To this end, we track connections by extracting the user ID of the app that started a specific network flow from the kernel's `proc` directory. Subsequently, we translate this user ID to the package name of the app using Android's `PackageManager` API.

CloudAnalyzer's way of utilizing Android's `VPNService` prevents users from using an actual VPN connection. This limitation can be circumvented by integrating CloudAnalyzer either into the VPN client or server. On a different perspective, CloudAnalyzer asks for permission to access sensitive network traffic and hence users need to trust CloudAnalyzer not to misuse this privilege. This requirement holds for all privacy enhancing technologies working on network traffic and we are convinced that increased privacy outweighs the required trust. Furthermore, unlike related work [RVS+16, SH15], we do not require users to install a Certificate Authority (CA) certificate to perform man-in-the-middle analyses. Hence, CloudAnalyzer intentionally remains oblivious of the *content* of encrypted sensitive communication.

In summary, by using Android's `VPNService` and keeping track of connections, we can observe network traffic on off-the-shelf Android devices (Version 4.4 and newer) without the need for system modifications. Furthermore and in contrast to in-network traffic monitoring, we are able to associate network packets to the app they originate from. Hence, we can use CloudAnalyzer to check the network traffic of individual apps for communication with cloud services.

3.3.3 Real-World Cloud Usage

We now set out to uncover the cloud usage of mobile apps using CloudAnalyzer. To this end, we first discuss our observations derived from running CloudAnalyzer on devices of volunteers. Subsequently, we report on additional measurements of popular mobile websites and the most used apps in multiple countries to highlight different aspects of cloud usage at larger scales.

3.3.3.1 Cloud Usage on User Devices

We begin our study by analyzing the cloud usage of actual users on their mobile devices. To this end, volunteers installed CloudAnalyzer on 29 devices (it is possible that volunteers participated with more than one device each) and collected statistics on the cloud exposure caused by their apps over the course of 19 days.

Study Design

We advertised our study using mailing lists and personal contacts, but did not offer monetary incentives for participating in our study. People were already motivated to

participate through the opportunity of gaining interesting insights into their exposure to cloud services. Study participants could at any time pause CloudAnalyzer's traffic analysis or examine their cloud usage through a graphical user interface (GUI). As a result, volunteers could have changed their usage behavior based on the information provided by CloudAnalyzer. However, since our focus in this work lies on untangling the mobile cloud landscape, our experiments were not explicitly designed to capture these effects. Still, one volunteer contacted us to report on uninstalling an app based on the information provided by CloudAnalyzer, and we plan to further study such aspects in future work.

We collected aggregated statistics on cloud usage detected by CloudAnalyzer as well as general statistics, such as the amount of time CloudAnalyzer was running and the total amount of network traffic (serving as a baseline). For our analysis, we only consider data from days where CloudAnalyzer was running for at least 20 hours (to prevent partial measurements). In total, we were able to collect data for 347 days of mobile device usage covering 383 apps (we only collect information on apps that produce network traffic).

Privacy and Ethical Considerations

As the goal of CloudAnalyzer is to empower users to execute their right to privacy, we designed our study such that the risk of (inadvertently) harming the privacy of our volunteers is minimized. To this end, we followed the principles of privacy by design [Cav11] and ethical research guidelines [DK12]. We are only interested in technical usage characteristics of cloud services, not in user behavior. Hence, we neither collected personally identifiable information nor other statistics on our volunteers. In fact, we do not even know who participated in our study (unless volunteers actively disclosed their participation). We strictly minimized the collection of data to the amount necessary and aggregated all statistics directly on the volunteers' devices at a granularity of one day (to minimize the risk of de-anonymizing users based on temporal information).

Users were educated about the extent and purpose of data collection and had to explicitly agree to these conditions. We obliged ourselves to not share collected data with third parties. Furthermore, we gave users the possibility to exclude specific apps from the analysis. Finally, we offered them the option to disable automatic uploads of their statistics to manually review collected information before sending it to our measurement server.

Overall Cloud Usage

We begin our study by investigating the overall characteristics of cloud usage caused by the apps of our volunteers. In Figure 3.13, we show the complementary cumulative distribution function (1−CDF) for the *number of used cloud services per app* across all devices. On average, each app connects to 3.2 cloud services and 89.8 % of all apps contact cloud services, highlighting the potential privacy risks of cloud computing. Naturally, web browsers contact many cloud services (e.g., Chrome with 37 services)

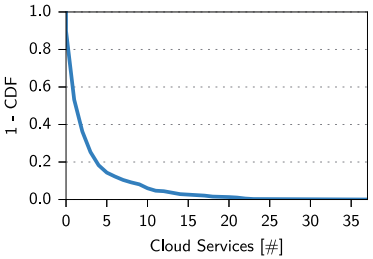

Figure 3.13 Number of cloud services accessed per app on user devices.

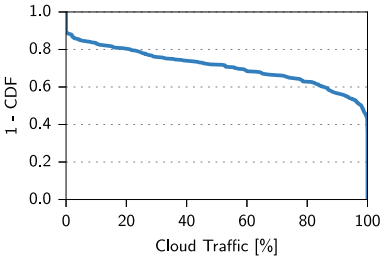

Figure 3.14 Fraction of cloud traffic of individual apps on user devices.

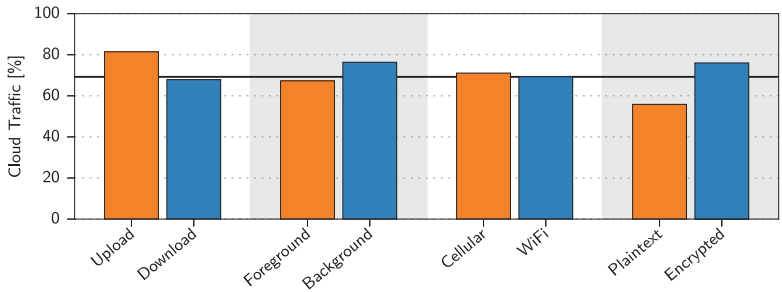

Figure 3.15 The fraction of cloud usage varies across the different dimensions of network traffic on user devices (solid line = overall fraction of cloud usage).

as users can visit a wide range of different websites that can rely on cloud services, but also less obvious candidates, e.g., the fitness tracking apps `com.withings.wiscale2` (12) and `com.myfitnesspal.android` (11), contact a large number of cloud services.

When looking at the *fraction of cloud traffic per app* in Figure 3.14, we make an even stronger observation. While 89.8 % of apps produce cloud traffic, 53.8 % of apps send 95 % or more of their traffic to cloud services. Notably, 35.5 % of apps *exclusively* communicate with cloud services. These numbers show that cloud entanglement is a real problem, concerning a majority of apps and often leading to the complete exposure of apps' communication to cloud services.

Different Dimensions of Cloud Traffic

Cloud traffic can be generated in various ways, e.g., directly triggered by users through interaction with an app or automatically by background processes, leading to different privacy risks. We study the *different dimensions of cloud traffic* in Figure 3.15, where we compare the fraction of traffic to and from cloud services along different dimensions of network traffic to the overall fraction of cloud usage (solid line). We observe a higher fraction of cloud usage in uploaded (81.4 %) than

Service	Traffic	Apps	Service	Traffic	Apps
Google	34.66 %	54.57 %	StackPath	0.45 %	7.57 %
Facebook	9.71 %	24.80 %	Microsoft	0.25 %	8.62 %
Amazon	8.76 %	65.27 %	Chartboost	0.19 %	0.26 %
Akamai	5.92 %	27.94 %	Dropbox	0.10 %	1.31 %
Fastly	5.54 %	13.32 %	SoundCloud	0.07 %	2.87 %
imgur	3.04 %	4.18 %	GitHub	0.05 %	2.87 %
Cloudflare	1.27 %	12.27 %	AppNexus	0.04 %	7.57 %
Snap	1.07 %	1.04 %	Criteo	0.04 %	5.74 %
Twitter	0.60 %	9.14 %	Netflix	0.03 %	0.52 %
Verizon	0.60 %	16.45 %	Tapjoy	0.03 %	0.26 %

Table 3.4 Fraction of overall traffic and app penetration for the 20 cloud services that account for the most traffic on user devices.

in downloaded traffic (67.9 %). These numbers indicate that a large fraction of data, potentially containing sensitive information, that leaves a smartphone is sent to cloud services. The higher cloud usage of 76.4 % for background (not directly triggered by users) compared to 67.3 % for foreground traffic (users interacting with the app) likely corresponds to synchronization tasks, e.g., updates of apps, typically happening in the background[5]. We do not observe a large difference in the cloud usage of traffic sent over cellular compared to WiFi networks. Furthermore, we observe that cloud usage is more prevalent for encrypted (76.0 %) than for plaintext traffic (55.8 %). While this observation most likely indicates that cloud services are faster in adopting security technology, it could also mean that data sent to cloud services is of more sensitive nature and thus requires encryption.

Most Prevalent Cloud Services. Given the overall high fraction of cloud traffic, we take a closer look at the *individual* cloud services that cause this traffic. In Table 3.4, we list the 20 cloud services with the highest fraction of cloud traffic across all devices. We witness that several providers receive a large portion of traffic generated by the apps on the mobile devices of our volunteers. Most notably, Google (also the developer of Android) accounts for 34.7 % of traffic and is accessed from more than half of all apps. While Amazon accounts for significantly less traffic, Amazon is contacted by nearly two-thirds of all apps. These numbers highlight that few cloud services have a high market penetration, both in terms of traffic and numbers of apps. This distribution can be especially problematic considering the imminent privacy risks resulting from a centralized cloud landscape (cf. Section 1.1.3).

Individual Perspective on Cloud Usage

To showcase that cloud entanglement has an individual component, we evaluate how users' selection of and interaction with apps influence their cloud exposure. To this end, we study the *per-device cloud traffic for the 20 most installed apps* on our volunteers' devices in Figure 3.16. Here, we exclude system apps, such as keyboards

[5]Differentiating between foreground and background traffic on Android is not straightforward. We use the broadcasts `ACTION_SCREEN_ON` and `ACTION_SCREEN_OFF` which (despite the slightly misleading name) indicate whether the device is sleeping (and hence non-interactive) or not [And18a].

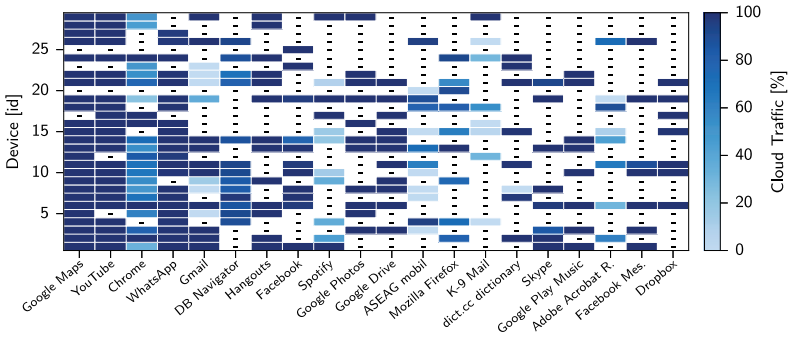

Figure 3.16 While most apps cause the same cloud entanglement across devices, certain apps' cloud traffic highly varies across different devices (varying shading for different devices).

or contact synchronization to not clutter the results. For each combination of device and app, we provide the fraction of cloud traffic ("–" denotes that an app did not produce *any* traffic on this device, likely because it was not installed).

Comparing the apps used on different devices, we notice that Devices 20, 23, and 25 use little to none of the 20 most popular apps. When looking at the apps used on these devices in more detail, we observe that these devices lack (the full stack of) Google apps, e.g., because of using custom firmware. For these devices, we directly witness a lower fraction of cloud usage. However, Device 25 is a notable exception which seems to be running Amazon's adaptation of Android, leading to a cloud usage comparable to the cloud usage of devices with installed Google services. When looking at the cloud traffic for the same app *across* different devices, we observe two classes of apps: The first class contains a large number of apps where the fraction of cloud traffic is the same across all devices. Among the 20 most used apps, this class covers apps that nearly exclusively use cloud services. Nevertheless, we also found less common examples that produce no cloud traffic at all (e.g., the client for the self-hosted Nextcloud or banking apps).

For the second class, we observe apps where the fraction of cloud traffic for the same app heavily deviates across devices, e.g., for web browsers and email clients. Hence, we discovered apps where cloud functionality is either built-in or not and others, where user behavior influences exposure to cloud services. For apps where cloud usage does not depend on user behavior, users can only decide to stop using a specific app if they deem its cloud usage too excessive. In contrast, for apps where cloud usage depends on user behavior, users might be able to change their behavior to also change cloud usage, e.g., by switching to a non-cloud hosted email provider.

3.3.3.2 Cloud Usage of Mobile Websites

To further understand the impact of varying user behavior on cloud usage, we now focus on the cloud exposure caused by visiting mobile websites. As we have seen in

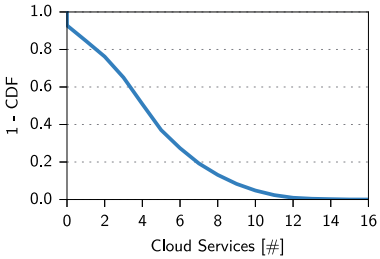

Figure 3.17 Number of cloud services accessed per popular website.

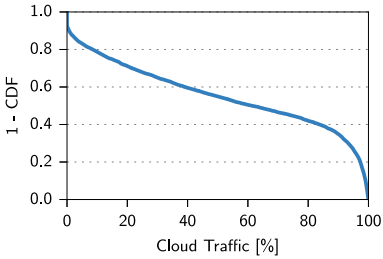

Figure 3.18 Fraction of cloud traffic produced by popular websites.

our previous measurements, web browsers are an important group of apps for which user behavior has a considerable influence on the level of cloud exposure. To gain a deeper understanding of this phenomenon, we analyze the cloud usage of the most popular websites for the cloud exposure they cause.

Measurement Setup

We mimic the mobile Chrome browser of a Google Nexus 5 smartphone and instruct it to visit the mobile versions of the 5000 most popular websites (measured by Alexa [Ale16]). We wait for each website to fully load and scroll to the bottom of the page to also trigger subsequent traffic resulting from embedded scripts.

Overall Cloud Usage

In Figure 3.17, we show the *number of cloud services per mobile website*. We observe that 92.8 % of the most popular websites use cloud services and on average each of the websites exposes their visitors to 4.8 cloud services. In the extreme case, fetching the mobile version of `rollingstone.com` leads to connections with 16 cloud services.

Additionally, we study the *resulting cloud traffic of mobile websites* in Figure 3.18. While 11.1 % of mobile websites are almost completely realized using cloud services (cloud traffic $\geq$ 99 %), we observe that the fraction of cloud traffic is nearly evenly distributed among websites, leading to a huge variety in the exposure to cloud services. Hence, which websites a user frequently visits highly influence her *individual* exposure to cloud services.

Most Prevalent Cloud Services. We now identify the cloud services that are responsible for the most cloud usage when visiting popular mobile websites. To this end, we present the 20 *cloud services with the highest traffic from mobile websites* in Table 3.5. In contrast to the most prevalent cloud services on mobile devices in general (cf. Section 3.3.3.1), we observe that Google has a significantly lower traffic share while CDNs play a more important role. Even though most cloud services do not account for large fractions of traffic generated by mobile websites, they are

Service	Traffic	Sites
Akamai	13.74 %	43.24 %
Google	12.02 %	84.50 %
Amazon	10.33 %	76.82 %
Cloudflare	8.97 %	48.76 %
Fastly	2.91 %	41.08 %
Verizon	2.12 %	24.28 %
Facebook	1.56 %	47.86 %
StackPath	1.16 %	13.38 %
Microsoft	0.59 %	13.78 %
Twitter	0.53 %	10.46 %

Service	Traffic	Sites
Incapsula	0.47 %	3.26 %
Alibaba	0.46 %	3.58 %
Yandex	0.36 %	3.18 %
AppNexus	0.33 %	33.02 %
Vimeo	0.15 %	0.48 %
LinkedIn	0.10 %	2.28 %
Oracle	0.09 %	6.36 %
Criteo	0.09 %	9.34 %
GitHub	0.08 %	2.32 %
Rackspace	0.06 %	0.42 %

Table 3.5 Fraction of overall traffic and website penetration for the 20 cloud services (we also treat CDNs as cloud services) that account for the most traffic on mobile websites.

embedded in a large number of websites (e.g., AppNexus accounts for only 0.3 % of traffic but is embedded by 33.0 % of websites). Most notably, Google and Amazon are present on 84.5 % respectively 76.8 % of mobile websites. This high penetration most likely results from small scripts, e.g., for Google Analytics, that are embedded in a large number of mobile websites. As a result, these services have the potential to create detailed tracking profiles of users [RKW12].

3.3.3.3 Cloud Usage of Popular Apps

So far, we have concentrated our efforts on studying cloud exposure caused by interaction with apps. However, to thoroughly compare the cloud exposure caused by different apps and reveal the influence of differing locations on cloud usage, we now test apps under comparable conditions at *large scale*. We analyze the 500 most downloaded free apps in Google Play [Goo16] for the five countries with the highest download numbers (Brazil, India, Mexico, Russia, and USA [App15]).

Measurement Setup

We run our measurements on real hardware to create a realistic environment and prevent apps from changing their behavior due to detected virtualization [MFB+15]. To this end, we connect five Nexus 7 (Model 2013) devices running Android 6.0.1 each to a dedicated wireless router. Each router operates a VPN connection to a server in one of the five countries under study, similar to the setup proposed by MATAdOR [SWZC16]. However, we use commercial VPN endpoints from VPNSecure instead of PlanetLab nodes.

To account for the effect of different VPN speeds, we fix network bandwidth to 2 Mbit/s. We execute each app for 1 minute and provide random user input using Android's Application Exerciser Monkey [And18b], as apps' communication can be based on user input. We repeat our measurements in parallel for all five countries on 10 different days. In total, we study 1475 different apps (one app can be among the 500 most popular apps in more than one country).

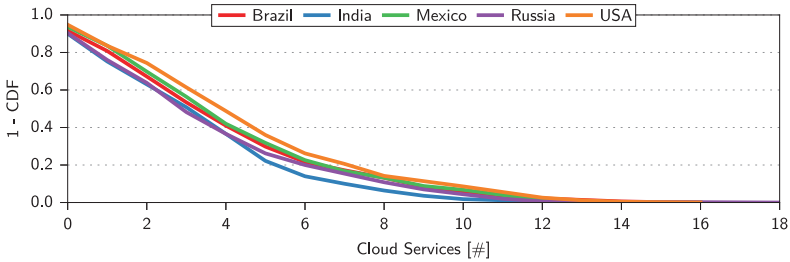

Figure 3.19 On average, each of the most popular apps uses 4.3 cloud services. Apps in the USA contact slightly more cloud services, while apps in India and Russia use less cloud services.

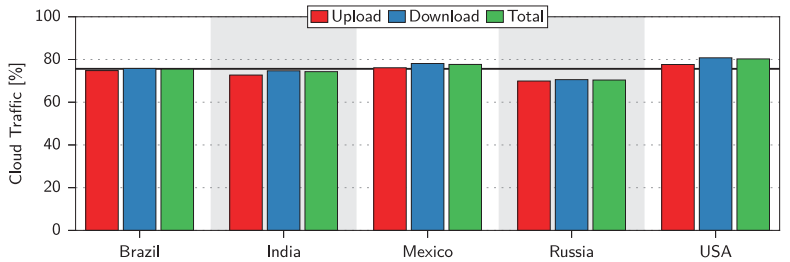

Figure 3.20 Traffic resulting from the most popular apps results in a slightly deviating cloud usage for the different countries (solid line = overall fraction of cloud usage).

Overall Cloud Usage

In Figure 3.19, we show the *number of utilized cloud services per app* for the five different countries (across all 10 days). Notably, 90.0 % (India) to 94.8 % (USA) of the studied apps connect to at least one cloud service. On average, each app establishes a connection to 4.3 cloud services (3.8 in India to 4.9 in the USA). Each of these contacted cloud service constitutes a potential privacy risk (cf. Section 3.3.1.2). The app with the highest number of contacted services, `com.fingersoft.hillclimb`, a game with 7.9 million installs, uses 18 cloud services when launched in Russia.

Given these already high numbers, we now set out to quantify the *fraction of traffic flowing to cloud services*. For each of the five countries, Figure 3.20 contains the average fraction of cloud traffic for upload, download, and total traffic over all apps. The total fraction of cloud traffic ranges from 70.4 % in Russia to 80.3 % in the USA, which is in the order of those numbers we observed for foreground and cellular traffic on real devices in the wild (cf. Section 3.3.3.1). Notably, here we observe a higher fraction of cloud traffic for downloads compared to apps on real devices, likely because a large number of free apps download advertisements from cloud servers. These numbers highlight that our measurement setup is well suited to study the behavior of apps during their interactive usage, as the observed results are sufficiently similar to the results observed on real devices (cf. Section 3.3.3.1).

Service	Traffic	Apps
Google	24.38 %	80.00 %
Amazon	20.90 %	80.27 %
Akamai	13.26 %	56.34 %
Facebook	5.84 %	50.98 %
Verizon	4.76 %	38.58 %
Unity	3.88 %	17.49 %
Chartboost	2.72 %	10.17 %
Fastly	2.12 %	17.69 %
StackPath	1.93 %	16.95 %
AppLovin	1.81 %	7.59 %

Service	Traffic	Apps
Cloudflare	1.38 %	18.58 %
Vungle	1.34 %	5.90 %
Microsoft	0.99 %	9.36 %
AppsFlyer	0.92 %	18.85 %
Yandex	0.71 %	3.86 %
Twitter	0.68 %	12.34 %
Criteo	0.48 %	13.69 %
Tapjoy	0.46 %	4.34 %
StartApp	0.46 %	3.25 %
Supersonic	0.42 %	4.68 %

Table 3.6 Fraction of overall traffic and app penetration for the 20 cloud services that account for the most traffic in our measurements of popular apps.

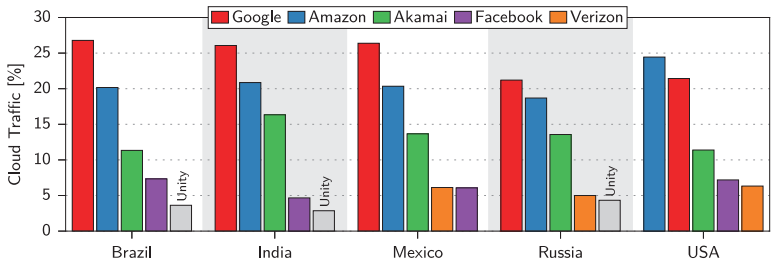

Figure 3.21 Despite a similar trend, we observe notable differences in cloud traffic of popular apps across the different countries in our study.

Most Prevalent Cloud Services

Given the frequent usage of cloud services by the 500 most popular apps per country, we now identify the *most used cloud services* to understand which individual services are particularly responsible for this cloud exposure. To this end, Table 3.6 contains the 20 cloud services with the highest fraction of traffic across the 500 most popular apps in all five countries. Furthermore, we list for each cloud service the fraction of apps that established at least one connection to this service. Here, we observe that the landscape of mobile cloud services is indeed highly centralized, with Google, Amazon, and Akamai each accounting for more than 10 % of an app's network traffic on average. Additionally, four cloud services (Google, Amazon, Akamai, and Facebook) are utilized by more than 50 % of the studied apps, significantly increasing the likelihood that users are exposed to these services. When studying these numbers, it is important to keep in mind that one network packet can be attributed to more than one cloud service when services are realized on top of each other. This situation occurs, e.g., for the audio distribution cloud service SoundCloud, which partly utilizes Amazon EC2 as infrastructure according to our findings.

Given the deviation in overall cloud usage between different countries identified in Figure 3.19, we now focus on what causes this effect by studying the *most-used cloud services in each country* in Figure 3.21. While overall we observe a similar trend

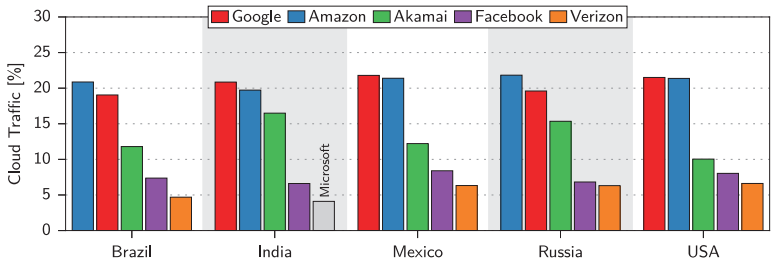

Figure 3.22 Identical apps utilize cloud services differently when operated in different countries.

across the five countries, notable differences exist: Verizon (2.7 % to 6.3 %) and Unity (2.9 % to 4.8 %) are among the five most-used services in only three countries (Mexico, Russia, and the USA for Verizon as well as Brazil, India, and Russia for Unity). Furthermore, Facebook (4.0 %) is not among the five most-used services in Russia. Finally, while Google accounts for the highest cloud usage in Brazil, India, Mexico, and Russia, Amazon (24.4 %) accounts for more traffic than Google (21.4 %) in the USA. Hence, the most popular apps in one country lead to a different cloud exposure and thus different privacy risks compared to other countries.

Influence of Location

To answer the question on whether the observed differences in cloud usage result from different apps used in the five countries or if cloud usage indeed differs based on users' location, we study the influence of location on cloud usage by testing *identical* apps for the five countries. Hence, we tested the 73 apps that are among the 500 most popular apps in *all* of our five countries and synchronized measurements across countries to rule out dependencies on time factors. Again, we ran the experiment on 10 different days.

We first study the cloud usage of the 73 apps by comparing the resulting fraction of cloud traffic for the five *cloud services with the highest traffic in each country* in Figure 3.22. While we observe an overall similar pattern of utilizing cloud services across all countries, we still derive differences between the individual countries: First, India (16.5 %) and Russia (15.3 %) show more traffic for Akamai than the other countries (10.0 % to 12.2 %). Second, Microsoft is among the five cloud services with the highest amount of traffic in India, compared to Verizon for the other countries. Hence, the exposure of users to different cloud services does not only depend on the used apps, but also on the location of the device.

To further study the influence of location, we rely on information on the position of data centers for some, especially larger cloud services that make this information public (cf. Section 3.3.2.2). We use this information to investigate whether the (network) location of a mobile device has an influence on the *geographical distribution of contacted cloud services*. More specifically, we show the fraction of traffic that we were able to assign to a geographic location (aggregated based on continents) in

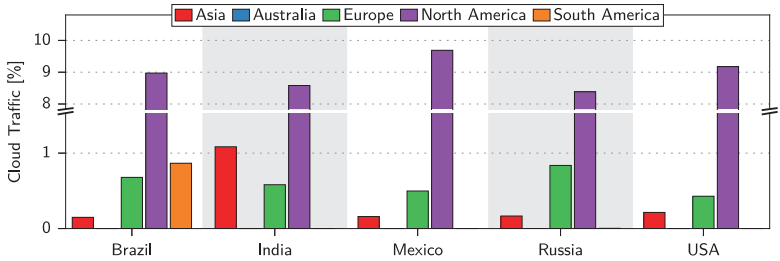

Figure 3.23 Identical apps partly use data centers on different continents when operated in different countries.

Figure 3.23. While the majority of traffic (for which we could derive a location) flows to North America (8.4 % to 9.7 % of overall traffic), we can observe that apps tend to connect to geographically close cloud data centers. This observation is illustrated by an increased fraction of cloud traffic to South America for Brazil, to Asia for India, and to Europe for Russia. Such information on the location of data centers used by apps allows users to execute their right to privacy, e.g., when deciding between apps with similar functionality [ZSW13, LLSH14]. More specifically, a user could prefer those apps that only connect to data centers located in the user's legislation.

3.3.4 Summary and Future Work

Apps on smartphones have access to a growing amount of sensitive information. As apps nowadays increasingly realize their functionality through cloud services, they potentially expose users' private information to a variety of third parties. Even worse, users are often unaware of the resulting possible erosion of their privacy. Starting from these observations, we provide a detailed analysis of the mobile cloud landscape which indicates and concretizes significant privacy risks.

Our problem analysis makes evident that users need to regain control over their privacy. As a first step towards this goal, we have to raise their awareness of their individual exposure to cloud services and the implied privacy risks. To achieve this goal, we present CloudAnalyzer which provides users with detailed statistics of their *individual* cloud exposure caused by their smartphone apps. CloudAnalyzer *locally* monitors network traffic of apps and detects communication with 55 cloud services that represent the mobile cloud computing landscape. As a consequence, we not only reveal the hidden exposure to cloud services caused by smartphone apps but also untangle complex and non-transparent data flows caused by indirection and subcontracting between cloud providers.

To show the applicability of CloudAnalyzer, we deploy CloudAnalyzer to 29 devices to reveal the cloud exposure of actual users over the course of 19 days. Additionally, we analyze the cloud entanglement caused by the 5000 most used mobile websites as well as the 500 most popular apps in five different countries. Our results confirm that smartphone users are indeed exposed to cloud services: About 90 % of

all studied apps contact at least one cloud service and 36 % of apps used by volunteers exclusively communicate with cloud services. One volunteer even reported on uninstalling an app due to excessive cloud usage uncovered by CloudAnalyzer.

We identify three promising directions for future work. First, CloudAnalyzer currently focuses on detecting the *destination* of apps' communication (i.e., used cloud services). To correlate identified communication with specific cloud services to the severity of resulting privacy risks, it is also important to consider the *content* of apps' communication (i.e., which private information is transferred out of the smartphone).

Here, our efforts can be nicely complemented by different streams of related work. Dynamic data flow tracking systems for Android such as TaintDroid [EGH+14] and TaintART [SWL16] monitor data flows during execution of an app to identify actual data leakage that occurs while executing an app. Detected data leakage occurring through system calls to send out data could be combined with CloudAnalyzer's ability to identify cloud services an app is communicating with. This approach, however, requires the modification of the system image of the mobile operating system—a solution likely not feasible for less technically proficient users.

Without modifying the mobile operating system, PrivacyGuard [SH15] and ReCon [RRL+16] detect leakage of personal information purely based on observed network traffic. When targeting leaks of personal information over encrypted connections, this, however, introduces the challenge of securely intercepting TLS connections [NSV+15]. Still and especially in lab settings, combining CloudAnalyzer with approaches to detect and classify the content of leaked personal data is a promising endeavor to further provide users with information on their individual exposure to cloud services and thus raise their awareness of the potential privacy risks resulting from uncontrolled cloud usage.

Second, CloudAnalyzer can be used as a foundation to enable users to compare their personal app-induced cloud exposure to that of their peers to discover potential privacy risks resulting from deviating from normal usage behavior. In the remainder of this chapter, we describe how the underlying concept of comparison-based privacy [ZHHW15] can be realized in a privacy-preserving manner and report on a preliminary feasibility and applicability study based on CloudAnalyzer.

Finally and besides the technical results presented within the scope of this dissertation, the question arises how users perceive the information provided by CloudAnalyzer. For example, users could change their behavior of using smartphone apps to avoid or to reduce the usage of cloud resources. Such aspects are promising subjects of future work, especially targeting social and psychological implications of CloudAnalyzer.

To conclude, CloudAnalyzer empowers users to critically review their individual exposure to cloud services. With a clear view of their exposure and risk, users are encouraged to adapt their app usage behavior or to take more informed decisions when choosing between apps with similar functionality. Notably, CloudAnalyzer also constitutes a valuable tool for researchers interested in understanding the characteristics of users' exposure to cloud services.

Similarly, CloudAnalyzer is beneficial for app developers to ensure compliance with data protection regulations. Using CloudAnalyzer, developers can ensure that their app (and included third party libraries) does not inadvertently contact (certain) cloud services, especially if these are located in countries with weaker data protection regulations [FM12].

3.4 Privacy-preserving Comparison of Cloud Usage

MailAnalyzer and CloudAnalyzer provide users with detailed statistics about their individual cloud exposure when using email respectively mobile apps. However, although having access to such information, less technically proficient users might still wonder how dangerous (or not) their individual own usage behavior is. Hence, we want to enable users to anonymously compare their own cloud usage profile with the profiles of other, "similar" users.

To this end, we adapt the concept of comparison-based privacy that we developed for the similar context of over-sharing in social media [ZHHW15] to enable users to compare themselves along different privacy-relevant metrics to the usage behavior within their peer groups. To this end, we group users based on lifestyle and sociodemographic background and derive a representative cloud usage pattern for each group. Thereby, we enable users to compare themselves to different comparison groups and hence allow them to better assess their individual cloud usage as a basis for making an informed decision on their future usage of cloud resources.

Comparison-based privacy is motivated by the general social observation that comparisons are widely used by humans in their everyday lives to assess their own status, behavior, and decisions. Such comparisons are also effective in influencing a person's behavior, especially with respect to the bounded rationality of individuals and organizations, i.e., situations of limited possibilities for rational decision making (e.g., due to limited information, time, and cognitive resources) [Sim91]. For example, comparing oneself with others might prove particularly helpful in situations in which the actor has little knowledge [GG11]. This group-based comparison provides the user with a starting point for assessing her individual cloud usage risks.

Besides promising benefits, comparing cloud usage with other users poses privacy concerns itself, as the information which cloud services are used to which extent might reveal sensitive information: (i) the operator of the comparison system could learn the peer groups to which a user associates, (ii) the operator of the comparison system could try to infer the identity of a user, (iii) the operator of the comparison system could link together multiple contributions of a user, and (iv) small comparison groups could leak a user's contributions or installed apps.

Thus, from a technical perspective, we need to ensure that an individual's contribution to our group-based comparison is anonymous, i.e., no party may learn who contributed which usage patterns to the comparison. To this end, we employ a crowdsourcing approach with strong differential privacy [Dwo06] guarantees. As the

affiliation to certain peer groups itself may constitute private information worth protecting, we additionally need to unlink the (timely) correlation of contributions of a single user.

In the following, we *securely* realize comparison-based privacy to nudge users on their individual exposure to cloud services. Our system design introduces a privacy proxy that hides users' identities and employs k-anonymity [Swe02] as well as differential privacy [Dwo06] to aggregate and to further protect user contributions from disclosure. To study the feasibility and applicability of our approach, we evaluate it in the context of cloud usage caused by smartphone apps (cf. Section 3.3).

3.4.1 Related Work

Different approaches in related work provide a foundation for our goal of securely realizing comparison-based privacy in the context of cloud usage. The first group of approaches addresses the question of how to release aggregated statistics derived from a central database containing personal information contributed by users in a privacy-preserving manner. Sweeney [Swe02] introduces the notion of k-anonymity which essentially defines that the set of quasi-identifiers (cf. Section 2.2.1) must be identical among at least k users, i.e., there is an anonymity set of size at least k in which users cannot be distinguished based on their quasi-identifiers.

To account for situations where all sensitive database values for a set of quasi-identifiers are identical or chosen from a small known set, Machanavajjhala et al. [MKGV07] extend the notion of k-anonymity with l-diversity. Here, in addition to having the same quasi-identifiers, contributions from different users are required to have different database values. Similarly, Wong et al. [WLFW06] propose (α, k)-anonymity to limit the relative frequency of a specific database value to a user-defined threshold α. To provide even stronger privacy guarantees, Li et al. [LLV07] propose t-closeness that aims at a situation in which the distribution of database value within an anonymity set is close to the distribution of this database value within the complete database.

While the previous approaches aim at anonymizing identifying information, they still report the exact database values. However, these might still be exploitable to retrieve information on users, e.g., if all users in an anonymity set report the same value. In this context, differential privacy [Dwo06] aims at a situation in which it is impossible to tell whether a specific database entry has been included in an aggregated statistic or not.

To achieve differential privacy, the aggregate is typically distorted with specifically crafted noise, e.g., sampled from a Laplacian distribution. The amount of added noise (and hence the level of privacy) is controlled by the differential privacy parameter ε. Although all these approaches in their original form aim at a scenario where one central entity knows all database values in cleartext, they still provide valuable input for our work. Indeed, we apply the concepts of k-anonymity and differential privacy, but we need to realize them in a distributed fashion, where users only contribute encrypted quasi-identifiers and sensitive values.

Working towards this direction, PDDP [CRFG12] realizes a distributed system in which clients locally store their data and apply differential privacy to protect their answers when answering queries posed by analysts. In their subsequent approach SplitX [CAF13], the authors propose an XOR-based encryption scheme (similar to a one-time pad) and publish-subscribe channels to further increase the efficiency of differentially private queries over distributed user data. Following a different approach, Haze [BOT13] realizes a system for collecting road traffic statistics in a privacy-preserving manner. This system is based on a voting protocol, where users upload an encrypted vote for a range of values, e.g., their current speed.

Finally, RAPPOR [EPK14] builds on the notion of randomized responses—again in the setting of crowdsourcing statistics from user devices—and applies it to sets represented as Bloom filters directly on the users' devices. These approaches have in common that they work on some notion of histograms, where—besides the intentionally introduced noise—additional distortion is introduced by assigning values to bins that form the histogram. Working on histograms is typically necessary as these systems aim at supporting a wide range of application scenario. In our setting, we work on a strictly constrained value range and hence can directly operate on integers to achieve less distorted results.

3.4.2 System Design

The underlying idea of our approach is to empower users to compare their individual cloud usage with the cloud usage of other, "similar" users. To this end, we propose to leverage established milieu concepts, which deliver social segmentation indicators that can be used to assign users to peer groups based on social values, mindset, media usage, and consumer behavior[6]. For each of these groups, we derive the average cloud usage (e.g., for a specific app) and hence allow users to compare the exposure to cloud services with their peers. This comparison enables them to take a more informed decision regarding the usage of cloud services or certain apps.

The idea underlying our system design is to collect statistics for peer groups at a central entity and distribute them in aggregated form to all group members. To realize this functionality, it is indispensable that users have to share information about their cloud usage with other parties. However, both individual statistics on the usage of cloud-based services as well as affiliation with peer groups are sensitive information. Hence, our system has to be designed in a way that guarantees the privacy of all user contributions.

To this end, we introduce a *privacy proxy* to ensure that all user contributions are sufficiently anonymized such that even the operator of the system cannot gain access to contributions of individual users. As we show in Figure 3.24, the *smartphone* collects statistics on cloud usage and periodically sends these statistics in encrypted

[6]Creating and evaluating approaches to derive peer groups is an ongoing effort that is mainly driven by our collaborators from the sociology department. Since our system design is agnostic to the approach for creating peer groups, we focus on the technical specifics of realizing comparison-based privacy for comparing cloud usage in a privacy-preserving manner in the following.

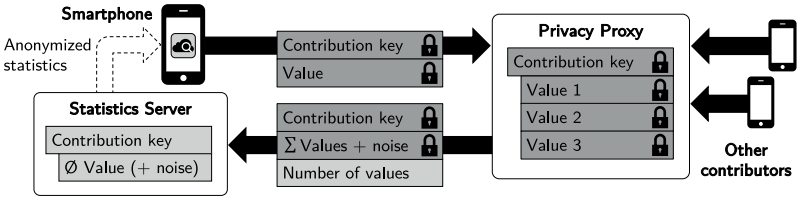

Figure 3.24 To realize comparison-based privacy in a privacy-preserving manner, the privacy proxy creates an anonymity set over different contributions for the same key. The values for one anonymity set are aggregated and distorted before they are sent to the statistics server, which distributes the resulting aggregated noisy cloud usage statistics over an API.

form to the *privacy proxy*. The privacy proxy—without being able to decrypt the statistics—aggregates statistics of different users that belong to the same measurement (e.g., the cloud usage for a specific app within a peer group on a certain day). As soon as the privacy proxy received enough contributions for a measurement (from different users) to guarantee anonymity, it sums up the measurements and adds random noise before releasing the aggregate to the *statistics server*. The statistics server is able to decrypt the aggregated statistics and persists them in a database. To enable comparison within a peer group, each user can then query the statistics server for the aggregated (noisy) result of a specific measurement.

In the following, we first present our security assumptions before we discuss the three entities in our privacy-preserving comparison system in more detail.

Security Assumptions

The underlying assumption of our system design is that the privacy proxy provides its functionality in an honest-but-curious manner (cf. Section 2.3.2). Hence, the privacy proxy operates according to the protocol specification, which includes that its interfaces do not only accept all correctly formatted data but also process and store it as intended. We do not make any assumptions regarding the statistics server, hence it can behave maliciously, e.g., trying to deanonymize users. However, we do assume that all communication between the three entities in our system is secured according to the state-of-the-art, e.g., using TLS, to protect against outside entities. Our system design is secure as long as the privacy proxy and the statistics server do not collude, which can, e.g., be realized and enforced through contracts and auditing of systems (cf. Section 2.1.3).

Smartphone

We use CloudAnalyzer (cf. Section 3.3) to detect cloud usage of mobile apps on Android using IP addresses, DNS names, and TLS information obtained from passive network traces. Based on the information provided by CloudAnalyzer, the smartphone calculates the *contribution value* for each day and app, i.e., the fraction of

traffic that has been sent to cloud services, and encrypts this value with the public key of the statistics server using an additively homomorphic cryptosystem. Furthermore, it creates a *contribution key* identifying the measurement by the app's name, date, and an identifier for the peer group. The smartphone then encrypts the contribution key with the statistics server's public key using a deterministic cryptosystem and sends the encrypted key and value to the privacy proxy.

Periodically, the smartphone queries the statistics server to retrieve the aggregated statistics for all relevant contribution keys (depending on the apps and peer groups of the user). It then presents the resulting average cloud usage statistics for each app and peer group together with the user's own cloud usage statistics to the user. We show an example for this graphical representation in Section 3.4.3.

Privacy Proxy

The core task of the privacy proxy is to separate user contributions from their origin, i.e., any information that can be used to identify an individual user. To achieve this goal, the privacy proxy performs two tasks: (i) creation of a sufficiently large anonymity set and (ii) aggregation and distortion of user contributions. We briefly discuss these two tasks in the following.

Anonymity Set. The privacy proxy employs k-anonymity [Swe02] to prevent that collected statistics can be used to infer information on individual users that contributed statistics on their cloud usage. To this end, the privacy proxy waits until it received at least k contributions of the same contribution key, i.e., a measurement identified by app name, date, and peer group, to create a sufficiently large anonymity set. Only after the privacy proxy received enough contributions, it aggregates these contributions, applies differentially private noise, and forwards the result to the statistics server. To further increase privacy (and the number of usable contributions), the privacy proxy can also first buffer received contributions for a certain period (e.g., a day), before it processes the data for all contribution keys with $\geq k$ contributions.

To create such an anonymity set, the privacy proxy needs to be able to differentiate between different contribution keys. However, since the contribution key itself contains sensitive information, such as the apps installed on a user's smartphone and her affiliation with peer groups, only the statistics server should be able to read this key. Hence, we employ a deterministic cryptosystem to encrypt contribution keys such that only the statistics server can decrypt and hence access this information. Although the privacy proxy cannot decrypt received contribution keys, it can leverage the deterministic property of the cryptosystem to derive which values belong to the same key (as the same plaintext is mapped to identical ciphertext) and hence create an anonymity set of size at least k. In our setting—unlike related work—employing k-anonymity is sufficient because we only release aggregated results that are additionally protected using differential privacy.

Aggregation and Distortion of Contributions. To further protect cloud usage statistics within an anonymity set, the privacy proxy aggregates and distorts them

before forwarding them to the statistics server. Again, since the (unaggregated) cloud usage statistics contain sensitive information, only the statistics server should be able to decrypt them. To still allow the privacy proxy to aggregate and distort the statistics, we employ an additively homomorphic cryptosystem where only the statistics server can decrypt the ciphertext, but everyone can perform additions on the encrypted values. As the statistics server should only have access to the average within the anonymity set, the privacy proxy adds up all contributions within the anonymity set under encryption and only sends the still encrypted sum as well as the size of the anonymity set to the statistics server.

To further restrain possible conclusions about the cloud usage of individual users that contributed their cloud usage statistics, e.g., because all users in an anonymity set have similar cloud usage behavior, the privacy proxy distorts the aggregated result using differential privacy [Dwo06] before forwarding it to the statistics server. More specifically, the privacy proxy randomly samples noise from the Laplace distribution centered around 0:

$$\mathrm{Lap}(x|\lambda) = \frac{1}{2\lambda} \exp\left(-\frac{|x|}{\lambda}\right), \text{ with } \lambda = \frac{\Delta f}{\varepsilon}$$

where Δf is the sensitivity of the aggregation function, i.e., the maximal influence of the contribution of a single user on the overall result (as we consider the average cloud usage of an app across users, Δf is 100 % in our scenario), and ε is the privacy parameter that controls the amount of noise (a smaller ε results in more noise and hence increased privacy but reduced utility). The privacy proxy then adds the sampled noise to the encrypted sum. Finally, the privacy proxy releases the still encrypted noisy sum, the number of values in the anonymity set, and the encrypted contribution key to the statistics server.

Statistics Server

The statistics server receives encrypted, aggregated, and distorted cloud usage statistics for a specific contribution key from the privacy proxy. It then decrypts the received contribution key and the noisy sum and calculates the noisy mean value by dividing the noisy sum by the number of values. Finally, it stores the contribution key and the mean value in a database. Users can query the statistics server for the anonymized mean cloud usage for a particular contribution key (app name, date, and peer group), which enables them to compare their own cloud usage (stored on their smartphone) to a peer group. In this process, the secure combination of privacy proxy and statistics server guarantees the privacy of users and their contributions (under the assumption that privacy proxy and statistics server do not collude).

3.4.3 Feasibility Study

To assess the feasibility and applicability of our approach, we realize a prototype of the smartphone component for Android as well as implement the privacy proxy

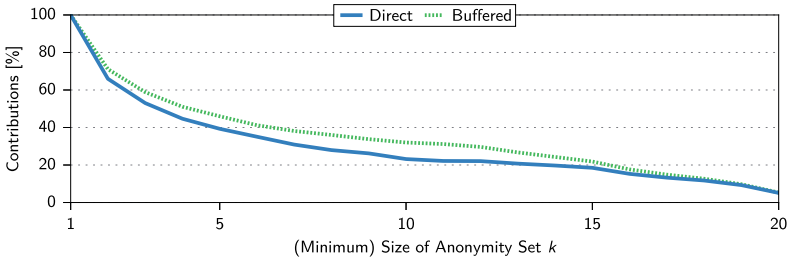

Figure 3.25 The size of the anonymity set k directly influences the amount of contributions that can be utilized for comparison-based privacy. Buffering contributions for a day slightly increases the fraction of contributions that can be utilized.

and the statistics server using Python. We use Paillier [Pai99] as additive homomorphic cryptosystem as well as a combination of salted SHA-256 hashes and a crypto box construction based on Curve25519, Salsa20, and Poly1305 [Ber09] to mimic a deterministic cryptosystem.

For our evaluation, we rely on the measurements we performed to study the cloud usage of actual users on their smartphones in the context of CloudAnalyzer (cf. Section 3.3.3.1). These measurements encompass cloud usage statistics we retrieved from 29 Android devices operated by volunteers over a period of 19 days. In total, these cloud usage statistics cover 383 different apps and 347 days of mobile device usage. We refer to Section 3.3.3.1 for a more detailed discussion of the study design and ethical considerations.

In the following, we first study the influence of the two privacy parameters of our system (size of the anonymity set k and differential privacy parameter ε) before we show an example comparison result for the cloud usage based on our user study.

Influence of Size of Anonymity Set

We study the influence of the size of the anonymity set (k) in Figure 3.25. The choice of k directly influences which contributions can be included in the analysis, as contributions for a specific key (app name, date, peer group) can only be used if at least k users provide their values.

Furthermore, the privacy proxy can either *directly* forward contributions as soon as the threshold k is reached or first *buffer* them (e.g., for a day) before releasing data for all keys with $\geq k$ contributions. For the 29 devices and 19 days we cover in our user study, Figure 3.25 shows that 28.9 % of contributions (difference between $k = 1$ and $k = 2$) are unique and hence cannot be shared without diminishing privacy. By buffering contributions for a complete day, we can slightly increase the fraction of contributions that can be leveraged for the comparison.

For a reasonable choice of $k = 5$ (for our small number of contributors) [WDL13], we can still leverage 39.3 % (direct release) respectively 46.0 % (buffered release) of

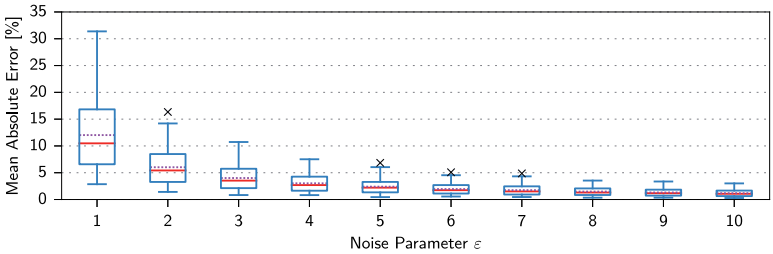

Figure 3.26 Increasing the differential privacy parameter ε reduces the mean absolute error of the aggregated result as less noise (and hence less privacy protection) is added.

contributions. For larger numbers of users—where more usable contributions are expected—increasing k to 10 is advisable [WDL13]. For our small dataset, we fix $k = 5$ and buffer contributions for one day in the following.

Influence of Differential Privacy

To study the impact of differentially private noise, we replay the data collected by our volunteers 30 times using real random seeds [Wal96] to generate Laplacian noise for different privacy parameters ε. Figure 3.26 shows the distribution of the mean absolute error for each app and day (over 30 runs) for different ε. While the majority of approaches in related work uses values of $\varepsilon < 1$, our rather high choices of ε are also reflected in prominent related work [MM10, MS10, CLSX12], especially for comparatively small data sets such as in our feasibility study. The challenge of applying differential privacy in our scenario is to add noise such that privacy is protected and the result is still usable, as the statistics server no longer receives an accurate result due to the distortion. For $\varepsilon = 1$, the mean absolute error on average amounts to 12.0 % (dotted line), which clearly impacts utility. In contrast, $\varepsilon = 5$ with a mean absolute error of on average 2.4 % provides a good trade-off between privacy and utility for our small dataset. We hence use $\varepsilon = 5$ in the following. When considering a real world deployment of our approach with likely hundreds of contributions for a specific key, it is both possible and advisable to choose a smaller ε do offer a higher level of privacy protection.

Exemplary Comparison Result

Figure 3.27 exemplarily shows the comparison result of one of our volunteers to their peer group (all volunteers in our study) for two prominent apps. We selected these two smartphone apps—`Chrome` and `Gmail`—as they are prime candidates to showcase different aspects related to the result of the comparison and potential impact on users. The violet line in Figure 3.27 represents the anonymized mean cloud usage within the user's peer group (in our case all 29 devices) with a 10 % margin (thick lighter violet line). Over a period of two weeks, each dot represents the cloud usage

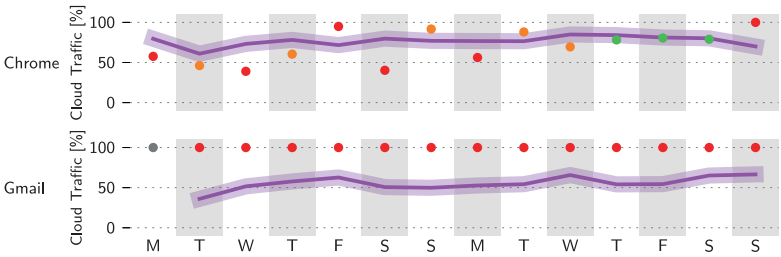

Figure 3.27 In this exemplary comparison result of one user to their peer group, we observe that the user's cloud usage follows the group's average rather closely for the `Chrome` app but clearly deviates from average usage behavior for the `Gmail` app.

of the user on a particular day. Here, colors inform the user how much their cloud usage deviates from those of the peer group. A green dot lies within a 10 % margin of the anonymized mean cloud usage, while orange dots deviate by at least 10 % and red dots by at least 20 % from the mean cloud usage, respectively. The gray dot reported on the first day for the `Gmail` app indicates that not enough contributions for a privacy-preserving comparison where received and hence no comparison is possible. For our volunteer, we observe that the usage pattern is quite similar to the peer group for the `Chrome` app. However, for `Gmail` (the standard email app on Android) the volunteer's cloud usage is significantly higher than the average of the peer group, identifying potential privacy risks as the user apparently is using a cloud-based email service while other users in the peer group (at least partly) use email services not hosted in the cloud.

3.4.4 Summary and Future Work

Relating privacy risks to the app-induced cloud exposure significantly challenges less technically proficient users. We apply the concept of comparison-based privacy to the cloud usage of smartphone apps and present a system design to realize this concept in a privacy-preserving manner. To this end, we introduce a privacy proxy that ensures that all user contributions are sufficiently anonymized based on the concepts of k-anonymity and differential privacy such that even the operator of the system cannot derive contributions of individual users. With our approach, we enable users to anonymously compare their cloud usage with those of the users in their peer groups and hence allow them to better assess their individual cloud usage risk. As a result, we lay the foundation for users to make informed decisions on suitable means for sufficient self-data protection for their future use of cloud services. The results of our feasibility study indicate that anonymously comparing the extent of cloud usage is indeed a feasible and promising approach to nudge users towards exercising their right to privacy.

Given the preliminary status of our feasibility and applicability results, future work is mainly concerned with testing and validating our approach in a larger study. To

this end, we are working with sociologists to create and evaluate different approaches to derive peer groups. Here, we plan to take into account social milieus, social values, as well as attitudes to work, family, leisure, and media consumption. Given that mindsets and value-orientations are somehow stable cognitive orientations underlying lifestyles and consumption patterns, we consider this milieu-based segmentation approach to hold significance for grouping and comparing the users of cloud-based services. Furthermore, we believe that our approach is also valuable to study other privacy aspects beyond cloud exposure, e.g., the privacy risks involved with location sharing on mobile devices, such as smartphones and GPS trackers.

3.5 Conclusion

Based on the observation that everyday technology, such as email, mobile apps, and IoT devices, increasingly relies on cloud services—often without users' awareness of (the extent of) their exposure to these services and resulting privacy risks—we proposed to put users back into control over their privacy by uncovering their cloud usage and thus raise their awareness for the resulting privacy risks. We selected two important deployment domains of cloud services even less technically proficient users regularly interact with to develop approaches that uncover the resulting exposure to cloud services. Additionally, we realized support for users in contextualizing their cloud usage through privacy-preserving comparisons with their peers.

MailAnalyzer targets cloud-based email services as our first deployment domain. To this end, it analyzes information contained in the protocol headers of received emails and correlates this information with data publicly provided by cloud and email providers as well as patterns derived from the Internet infrastructure such as DNS or BGP routing data to detect the usage of cloud resources. We utilized MailAnalyzer both to study email infrastructure that is used when sending email as well as to analyze the cloud usage of 31 million actually received emails. The results we obtained using MailAnalyzer reveal that as of 2016, 13 % to 25 % of the received emails in our study were exposed to cloud services. Notably, 30 % to 70 % of this cloud usage cannot be detected by simply looking at the sender or the receiver.

For our second deployment domain, CloudAnalyzer is concerned with the usage of cloud-based services by mobile apps on smartphones. CloudAnalyzer runs on unmodified off-the-shelf smartphones and passively monitors the network traffic of mobile apps. Similar to MailAnalyzer, it detects cloud usage by comparing DNS, IP, and TLS protocol information to data of a set of 55 representatively selected cloud services. Using CloudAnalyzer, we studied the cloud usage of mobile apps in a user study with 29 volunteers during a period of 19 days, by crawling the 5000 most popular mobile websites, and through automate execution of the 500 most popular apps in five different countries. Our study results show that 90 % of mobile apps connect to cloud services with an average number of 3.2 contacted cloud services per app. Out of the apps installed on the devices of our volunteers, 36 % exclusively communicate with cloud-based services.

Finally, we apply the concept of comparison-based privacy to enable users to put their cloud usage into context through comparison with their peers in a privacy-preserving manner. Our system employs k-anonymity and differential privacy on encrypted cloud usage statistics to retrieve noisy aggregate cloud usage statistics for different peer groups. We performed a preliminary study on the feasibility and applicability of our approach based on cloud usage data obtained from 29 mobile devices during a period of 19 days. The results of this study indicate that the privacy-preserving comparison of cloud exposure is a feasible and promising approach to uncover the potential privacy risks of cloud usage and hence support users in exercising their right to privacy.

In this chapter, we addressed the research question on how cloud *users* can enforce their privacy when using cloud services. To this end, our contributions mainly target the core problem of cloud computing's technical complexity and missing transparency with the ultimate goal to put users back in control over their privacy. Our results presented in this chapter reveal that typical users are exposed to a large number of cloud services and to large extent during everyday Internet activities. Both, by providing users with statistics on their individual exposure to cloud services and by enabling them to contextualize these statistics through comparisons with their peers, we provide users with the transparency over the utilization of cloud-based services that has been missing so far.

It is important to note that the results we obtained in this chapter not only serve as a foundation for users to regain control over their privacy but also serve as a clear motivation for the need to account for privacy in the cloud computing landscape and hence our remaining contributions that comprehensively cover infrastructure providers, service providers, and cloud users to make cloud computing more privacy-friendly by overcoming the core problems for privacy in cloud computing that we identified in Section 1.1.3.

4

Data Handling Requirements-aware Cloud Infrastructure

The results presented in the previous chapter motivate the imminent need to account for privacy in the cloud computing landscape. We now turn our attention to the different actors in the cloud computing landscape and how each of them can contribute to making cloud computing more privacy-aware.

In this chapter, we begin these efforts by addressing our research question on how infrastructure providers can support service providers and cloud users in executing control over privacy. To this end, we first summarize the motivation for and vision of data handling requirements-aware cloud infrastructure [HHW13a, Gro13, Kop13, HGKW13] and derive the necessary components to realize this vision (Section 4.1).

As our first component to realize a data handling requirements-aware cloud infrastructure, we present our compact privacy policy language (CPPL) [Sch15, HHS+16], through which we enable users to express their data handling requirements and turn them into a concise representation that serves as a foundation for respecting data handling requirements when storing data in the cloud (Section 4.2). Based on this policy language, we introduce PRADA [Gie14, Seu15, HMH+17, HMH+18], a general key-value based cloud storage system that offers rich and practical support for data handling requirements to overcome current privacy limitations (Section 4.3). We conclude this chapter with a discussion and summary of our work (Section 4.4).

4.1 Motivation and Vision

When moving sensitive data, e.g., customer records or sensed information, to the cloud, users (both private and corporate) often impose data handling requirements (DHRs) that need to be met by the cloud provider (cf. Section 2.3.1). For example,

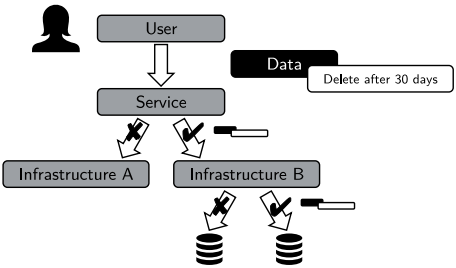

Figure 4.1 A user adds an annotation to her data ("delete after 30 days") before it is passed to the cloud. Based on this annotation, the service chooses an infrastructure which then places the data on a physical device together with other data that should be deleted in 30 days.

a company using a cloud storage service might require that the data of its customers is stored and processed only in a specific legislation to comply with legal requirements. However, in current cloud infrastructure, it is extemely difficult to meet these requirements adequately, as users cannot specify their requirements and cloud providers thus remain completely oblivious of these requirements. More importantly, even if cloud providers were willing to adhere to users' requirements, they often lack the technical means to do so at a fine granularity and instead retreat to static SLAs (cf. Section 2.1.3) that provide users with only little choice. Consequently, the ability to support DHRs would allow cloud providers to enter new markets by addressing customers which want or have to adhere to these requirements.

To achieve support for DHRs in cloud infrastructure, we propose to enrich data in a cloud environment with data handling annotations. Data handling annotations (also known as sticky policies) are a well-established method in the field of data usage management and control [PSM09, ADBK10, PM11, SM12] and we propose to leverage them to signal DHRs across the different entities in the cloud stack (cf. Section 2.1.2.1). We illustrate our vision of annotating data with DHRs using an example in Figure 4.1. In our example, we consider a cloud service that provides storages and synchronization of data across different devices (similar to, e.g., Dropbox). The user wants to upload a file that should be securely deleted after 30 days. To this end, she annotates her data accordingly before she sends it to the cloud service. Subsequently, the cloud service verifies whether or not it can fulfill this obligation and (potentially) chooses between different infrastructure providers it has under contract to select one that is able to fulfill the user's requirement.

The chosen infrastructure provider then has to decide on which part(s) of its storage infrastructure the file should be stored. To ensure the secure deletion after 30 days (cf. Section 2.3.1), the infrastructure provider could put data with similar deletion dates on the same physical device and securely dispose it off once the deadline for deletion has passed. Without the possibility for users to annotate their data with DHRs, the infrastructure provider does not know about these requirements and hence cannot adhere to them. Not only users but also cloud providers benefit from a support of DHRs: Besides enabling cloud providers to tap into the market of

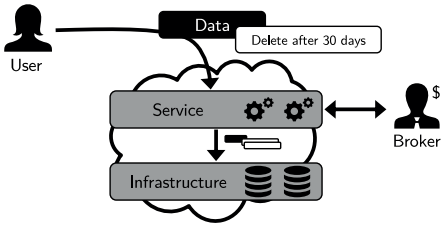

Figure 4.2 Besides the DHRs of the user, each service can add additional requirements. Services use a broker to locate infrastructure providers that comply with the stated DHRs.

customers that are currently unable to utilize the cloud, providing support for DHRs empowers operators to efficiently handle differences in regulations across legislations and industries, similar to the advantages of secure cloud services [CHHD12].

4.1.1 A Data Handling Requirements-aware Cloud Stack

When speaking about data handling annotations as the foundation for a DHRs-aware cloud stack, we consider entities in a layered system, where data is exchanged vertically between entities on adjacent layers as well as horizontally between entities on the same layer. Each entity on the data handling path can add data handling annotations to the data. The resulting data handling obligations are then considered binding for *everyone* on the remaining portion of the data handling path. We argue that this approach is better suited than static SLAs (cf. Section 2.1.3)—prevalent in the cloud computing landscape today—to fulfill privacy requirements in cloud computing, since the dynamic nature of cloud computing and constantly changing and evolving privacy requirements are difficult to handle solely with SLAs [ZB11].

In the following, we further develop the above example to motivate our vision of a DHRs-aware cloud stack. To this end, Figure 4.2 provides a high-level overview of our envisioned architecture. In this setting, a cloud service receives data that is annotated with DHRs from the user. Furthermore, the service provider might itself impose additional requirements. A prominent example for requirements imposed by services results from data protection requirements in the EU, which require certain data on customers to not leave the legislative boundaries of the EU (cf. Section 2.3.1). Hence, the service provider has to select an infrastructure provider that is able and willing to adhere to the resulting combined set of DHRs imposed by user and service provider, e.g., by utilizing existing cloud brokers [GB14] which today determine the best cloud provider based on metrics such as quality of service (QoS), SLAs, and pricing. These brokers have to be extended to also support matching of DHRs against capabilities and policies of cloud providers.

As discussed in Section 2.1.2, cloud services can be realized on top of each other, leading to complex deployment scenarios. In this case, each of the services has to obey to the DHRs imposed by the user and any cloud services on top of it. Furthermore, each service can add additional requirements and rely on a broker to

locate an infrastructure provider that can comply with all stated DHRs. Finally, the infrastructure provider maps data to real hardware and is thus ultimately responsible for fulfilling the stated DHRs, e.g., when assigning data to storage nodes.

Hence, to realize our vision of a DHRs-aware cloud stack, we require two fundamental approaches: (i) the possibility (especially for users) to express DHRs and annotate their data accordingly as well as (ii) a way for cloud providers to comply with DHRs, showcased alongside the selection of storage nodes in a cloud storage system. In the following, we discuss the motivation for these two approaches in more detail.

Expression of Data Handling Requirements

Enabling users to express their DHRs in a machine-readable way is necessary to ensure that cloud services and infrastructure providers can comply with these requirements fully automatized. The widely studied field of privacy policy languages [KCLC07] deals with expressing privacy policies and requirements and hence is a prime candidate to serve as our foundation for expressing DHRs. We can differentiate between three different categories of privacy policy languages: (i) languages for users to specify their privacy requirements, (ii) languages for service providers to specify their privacy policies, i.e., how they will handle and use data, and (iii) languages that combine the two approaches to enable the matching or comparison of user requirements against service provider policies. We consider the third category most promising in our setting, as this allows users to express their DHRs and enables service providers to formalize which requirements they support. Thus, when receiving data annotated with DHRs, the entities in the cloud stack can automatically check whether they can comply with the stated requirements and act accordingly.

As we discuss in more detail in Section 4.2.1.3, a wide range of privacy policy languages has been proposed by related work, either generic or specifically tailored for a specific use case, e.g., to steer access control [GW10, Oas13], to formulate data handling policies [ABP09, TM11], or to support digital rights management [HPB+07]. However, when considering to apply these concepts to the cloud computing landscape, we identify two fundamental conceptual shortcomings: (i) constrained scope and hence limited expressiveness and flexibility and (ii) prohibitive processing, storage, and bandwidth consumption. These shortcomings become even more problematic with the recent proposal to attach DHRs to single network packets, e.g., to facilitate policy-based routing [KPPK11]. To overcome these fundamental conceptual shortcomings in the context of cloud computing, we hence require a privacy policy language that is both flexible and resource efficient.

Complying with Data Handling Requirements in Cloud Storage Systems

Once users are able to express their DHRs, cloud service and infrastructure providers are equipped with the necessary information to comply with these requirements. Here, the main challenge consists in respecting DHRs during the placement of data onto actual hardware. Most fundamentally, this applies to cloud storage systems, i.e., any infrastructure services—ranging from distributed file systems over

distributed key-value stores to distributed databases—that offer the persistent storage of data. However, we observe that cloud storage systems today do not offer support for complying with DHRs. Instead, the decision on which nodes to store data is primarily taken with the intention to optimize reliability, availability, and performance [DHJ+07, LM10, ÖV11, GHTC13], thus mostly ignoring the demand for support of DHRs. Even worse, DHRs are becoming increasingly diverse, detailed, and difficult to check and enforce [PSBE16], while cloud storage systems are becoming more complex, spanning different continents [AB13] or infrastructures [BRC10], and even different second-level providers [BLS+09, GB14].

Although the demand for realizing DHRs in cloud storage systems is widely acknowledged, practical support for them is still severely limited [Int12, WMF13]. Related work primarily focuses on enforcing DHRs while processing data [IKC09, BKDG13, ELL+14], limits itself solely to supporting location requirements [PGB11, WSA+12], or treats the storage system as a black box and tries to enforce DHRs at a coarse granularity from the outside [PP12, WMF13, SMS13]. Hence, a practical solution for enforcing arbitrary DHRs when storing data in cloud storage systems is still missing.

4.1.2 Contributions

With the goal to realize DHRs-aware cloud infrastructure, we present a mechanism for users to express their DHRs and an approach for cloud providers to comply with DHRs when selecting cloud storage nodes as our contributions in this chapter:

1) We present *CPPL*, a compact privacy policy language specifically designed for dynamic and high-frequent interaction patterns as they are prevalent in the cloud computing landscape. CPPL compresses a textual policy specification based on an interchangeable domain specification to enable adaptation of our domain specific compression to any (even yet unknown, future) deployment and network scenario. To illustrate the feasibility of CPPL, we perform synthetic benchmarks and compare CPPL to state-of-the-art privacy policy languages. Furthermore, we showcase the applicability of CPPL in the context of cloud computing, the IoT, and big data. Our results show that CPPL is able to reduce policy sizes by up to two orders of magnitude compared to related work and to process several thousand of policies per second in real-world settings, thus making the expression of DHRs feasible in the scope of cloud infrastructure deployments.
2) We present *PRADA*, a general key-value based cloud storage system that offers rich and practical support for DHRs to overcome current compliance limitations. PRADA adds an indirection layer on top of a cloud storage system to store data annotated with DHRs only on nodes that fulfill these requirements. Our design of PRADA is incremental, i.e., it does not impair data without DHRs. Furthermore, PRADA supports all DHRs that can be expressed as properties of storage nodes. We prove the feasibility of PRADA by implementing it for the distributed database Cassandra and by quantifying the costs of supporting DHRs in cloud storage systems. Additionally, we show PRADA's applicability along two use cases on real-world datasets, a Twitter clone storing two million authentic tweets and a distributed email store handling half a million emails.

4.2 CPPL: A Compact Privacy Policy Language

As the foundation for a DHRs-aware cloud stack, we require a mechanism for users to express their privacy and data handling requirements in a machine-readable way. This becomes necessary, since current state of the art, i.e., legal text dictating privacy policies by providers, can no longer sufficiently address privacy concerns as a massive growth in the amount of data—fundamentally changing data processing by superseding the prevalence of processing data locally by remote processing in the cloud—is accompanied by a significant increase in diversity of data sources [BWHT12] and high granularity of data [HHCW12].

To account for this development, related work proposes per-data item privacy policies (also referred to as sticky policies) [PM11, SSL12, PBSE16]: Instead of having a provider dictate one privacy policy for all users, per-data item policies enable each user to specify her own privacy requirements which then have to be enforced by the cloud provider. Such policies enable the user to express her individual privacy requirements down to the level of specific pieces of data. For example, readings of personal medical devices could be treated differently from much less sensitive readings of personal weather stations. This combination of user-centricity and granularity empowers users to effectively remain in control over their data, even if it leaves their physical control.

With the goal to realize such fine-grained user-centric policies, related work introduced a wide range of policy languages, either generic or specifically tailored to a certain scenario, e.g., in the area of accounting, banking, handling of insurance information, or processing of medical data of patients. However, we find that existing privacy policy languages are either not flexible enough or require excessive processing, storage, or bandwidth resources which prevents their widespread deployment. To overcome these shortcomings and thus to offer support for fine-grained user-centric policies in an interconnected world, we propose to introduce a *domain specific* compression step before sending a policy over the network. To this end, we incorporate flexibly specifiable domain knowledge to realize an efficient bit-level compression.

To provide a foundation for these efforts, we first analyze the deployment and network scenarios in the cloud computing landscape as well as the suitability of privacy policy languages proposed by related work to address emerging requirements in these scenarios. Based on our analysis, we find a mismatch between the communication patterns in such networks and the characteristics of existing privacy policy languages. Consequently, we propose CPPL, a compact privacy policy language which compresses privacy policies by taking advantage of flexibly specifiable domain knowledge to fill this gap.

Notably, CPPL is relevant beyond cloud computing as we show by realizing privacy policies in the context of the IoT and big data to showcase the applicability of CPPL. To this end, we evaluate the performance of CPPL and compare CPPL to state-of-the-art privacy policy languages as proposed by related work. Our evaluation shows that CPPL can reduce policy sizes by up to two orders of magnitude compared to related work and can check several thousand policies per second in real-world scenarios. Hence, CPPL enables individual per-data item policies that serve as the

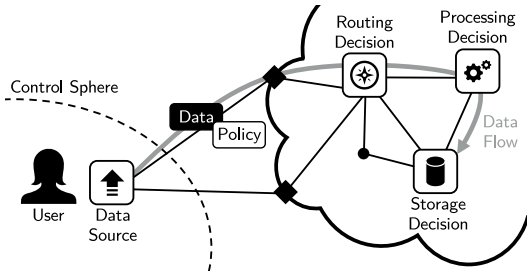

Figure 4.3 When data leaves the control sphere of the user, per-data item policies empower her to still influence routing, processing, and storage decisions.

foundation for a DHRs-aware cloud infrastructure. We provide the source code as well as a library binding of our implementation of CPPL under the open source Apache license (version 2)[7].

4.2.1 Privacy Policies and Cloud Computing

In this section, we outline our targeted scenario and derive requirements that we argue must be addressed by any viable solution to the challenge of realizing per-data item policies for cloud computing. We then rigorously analyze existing policy languages with respect to these requirements and identify different short-comings that render existing work mostly inapplicable in our scenario.

4.2.1.1 Scenario

We consider a scenario in which data is transferred out of the user's control sphere to cloud-based backend infrastructure as shown in Figure 4.3. While already commonplace today, this scenario becomes especially relevant in the context of the IoT (cf. Section 2.4). An IoT home automation system such as Apple Home [App18a], e.g., might transfer raw IoT data to a cloud backend to infer a user's presence and activity for optimal control of heating, ventilation, and air conditioning appliances.

Furthermore, with the upcoming trend of big data, masses of data will be used to derive novel insights. These deployment domains have in common that, while considering huge amounts of data in total, individual data pieces are comparably small. For example, single IoT measurements can be as small as 72 byte (cf. Section 4.2.3.3). When transferring this data out of the control sphere of users, it becomes subject to (overlay) routing, processing, as well as storage operations in the cloud backend. However, performing these operations outside the control sphere of users raises severe privacy concerns, which ultimately results in a complete loss of control of users over their data (cf. Section 1.1.3).

[7] `https://github.com/COMSYS/cppl`

To overcome these concerns, one promising approach in related work is to attach *per-data item privacy policies* (also referred to as *sticky policies*) to data before it leaves the control sphere of the user [PM11,SSL12,PBSE16] as depicted in Figure 4.3. Privacy policies are thus imposed by the user and are binding for all entities involved in handling the data in the cloud outside the control of the user. More specifically, data is only allowed to be routed to, processed on, and stored at nodes in the cloud fulfilling the privacy policy imposed by the user. To this end, the coupling of data and policy ensures continuous availability of the policy.

All entities involved in handling the data can impose additional, more restricting privacy policies. For example, this becomes relevant if a cloud service has to adhere to data protection regulation and wants to pass resulting requirements to the underlying cloud infrastructure. Furthermore, existing data integrity protection mechanisms can be extended to also cover the privacy policy to prevent modifications to privacy policies during transmission or data handling. Alternative approaches such as per-stream policies [NLB13, TLL16], which assign one privacy policy per data stream instead of individual policies per data item, lack support for the emerging concept of federated clouds where data is distributed among several cloud providers.

Our aim in this work is a functional improvement over the status quo by reducing policy sizes to feasible orders of magnitude for an interconnected world. We deliberately do not focus on the orthogonal problem of enforcing policies, i.e., providing (formal) guarantees that cloud nodes indeed adhere to policies. As CPPL does not change the semantics of policy languages, existing solutions that propose cryptographic guarantees [IKC09, HPB+07], tracking data flows [PBSE16], or creating audit logs [PJ12] to enforce policies do still apply.

4.2.1.2 Requirements

We refer to the machine-readable formalization of privacy policies as privacy policy languages. In the following, we derive key requirements for any privacy policy language for the above-described scenario where (potentially small) data leaves the control sphere of the user and is forwarded to cloud-based infrastructures, e.g., in the context of the IoT and big data.

Minimal Storage Footprint: As privacy policies are attached to data and travel with it through the network, they inadvertently result in additional transmission and storage overhead. It is thus paramount that privacy policy languages minimize storage footprint. Minimizing the storage footprint of a privacy policy is a quantitative requirement which can be evaluated by looking at the resulting policy size.

Efficient Policy Checking: Privacy policies are evaluated at numerous times, e.g., whenever data is relocated, replicated, and processed. Hence, the overhead for checking whether a policy matches with the properties offered by a cloud node must be minimized. The efficiency of policy checking can be quantified by measuring the processing runtime required for the necessary operations.

Expressiveness: We identify a large spectrum of expectations for the handling of data: (i) restriction of storage location to a certain country, (ii) deletion of a data

item at a specified point in time, (iii) logging or notification when data is accessed by a third party, or (iv) replication rate of data to ensure availability (cf. Section 2.3.1). Hence, a privacy policy language must provide the ability to express expectations for these various kinds of data handling. This requires the support of *environmental context*, e.g., awareness of storage location or replication rate, *time-based triggers* to specify the point in time for a future action such as deletion, and *event-based triggers* to initiate actions when an event such as data access occurs [ABP09]. Expressiveness of privacy policy languages is a qualitative requirement which can be evaluated by comparing different policy languages.

Extensibility: Enabled by the cloud computing paradigm, new services and application scenarios together with novel privacy and data handling requirements emerge continuously. Thus, a privacy policy language needs to be extensible such that it can be easily adapted to the individual and novel requirements of new deployment domains. The extent of extensibility of a policy language is a qualitative requirement which can be evaluated through analysis of the concept and implementation of a policy language.

Incremental Deployment: A new privacy policy language should be conceptually compatible with already existing privacy policy languages to integrate legacy deployments and ease transition. Whether a policy language supports incremental deployment or not can be qualitatively evaluated by analyzing the underlying design of the policy language.

Matching: A privacy policy language must support the matching between the privacy expectations of a user and the data handling properties cloud providers offer. To this end, cloud providers must be able to specify what their cloud nodes technically provide, which legal principles apply, as well as what the providers' own policies are based on individual business decisions. Also the question of whether a policy language supports the matching of expectations against properties can be qualitatively evaluated by studying the language's specification.

4.2.1.3 Analysis of Privacy Policy Languages

In this section, we analyze (privacy) policy languages from related work with respect to our scenario and requirements. We summarize our analysis in Table 4.1.

XACML [Oas13] is a completely XML-based language for specifying access control policies. XACML is extensible to new requirements and use cases but has an excessive storage footprint which requires applying separate compression [Gee05]. Additionally, XACML has no support for triggers (cf. Section 4.2.1.2). *PPL* [BNP10] and *A-PPL* [AEÖ+14,CDG+13] extend XACML with support of triggers, environmental context, credential-based access, and a matching procedure.

Likewise based on XML, *PERFORM* [DUM10] targets the scenario of pervasive computing. Policies in PERFORM specify actions as request/response pairs limited by constraints. Its awareness of environmental context affords a good basis for expressiveness. However, it does not support triggers thus, e.g., not supporting access notifications or specification of data deletion at a user-defined point in time.

	storage footprint	efficiency	expressiveness	extensibility	deployment	matching
XACML[Oas13] + extensions	◐	-	●	●	◐	●
PERFORM [DUM10]	◐	-	◐	-	○	-
Rei [KFJ03]	○	-	◐	●	○	-
Garcia-Morchon [GW10]	●	-	◐	-	○	○
Ali [ABP09]	◐	-	●	-	○	○
OSL [HPB+07]	◐	-	◐	-	◐	●
C^2L [PJ12]	◐	◐	◐	-	○	○
S4P [BMB10]	◐	-	-	-	-	●
FLAVOR [TM11]	◐	-	●	-	◐	○

Table 4.1 Comparison of existing privacy policy languages. A language fulfills (●), partially fulfills (◐), or does not fulfill (○) a requirement. We use "-" to denote that we cannot judge to which extent a policy language fulfills a requirement based on the information available.

Rei [KFJ03] also targets pervasive computing and supports specification of rights, prohibitions, obligations, and dispensations. The expressiveness of policies in Rei profits from awareness of environmental context but lacks support for triggers thus facing the same limitations as PERFORM in our scenario. Furthermore, Rei does not consider the size of resulting policies as an optimization goal.

Garcia-Morchon and Wehrle [GW10] propose an access control policy language for medical sensor networks. The resource constraints in this environment demand for a concise representation of policies. To this end, they specify policies in Boolean formulas represented as binary trees and efficiently stored in byte-level encoding. However, they explicitly focus on medical contexts which limits the generalizability of their language. Furthermore, matching of user expectations with provider offers is unnecessary for their scenario but paramount in more general scenarios.

Ali et al. [ABP09] describe an obligation language and a framework to enable privacy-aware service-oriented architectures. Their language supports the specification of obligations, can evaluate the environmental context, and supports time- as well as event-based triggers. However, it misses a mechanism to match offers of a node with expectations formulated by a user [ABP09], lacks an efficiency analysis, and does not consider a storage efficient representation of the formal language.

OSL [HPB+07] is a policy language for distributed usage control. In contrast to other languages, OSL partially supports the enforcement of policies by translating them into the digital rights management (DRM) languages ODRL and XrML and then employing existing enforcement mechanisms. However, its performance remains unclear and no attention is paid to the storage footprint.

C^2L [PJ12] is a highly specialized language for restricting the location and migration of virtual machines (VMs) in the context of cloud computing. A typed spatiotemporal logic enables enforcement of policies by rerunning the evaluation engine on the history of placement and migration of VMs. Hence, users are limited to a posteriori checking if a given history contradicts against a policy. Furthermore, the language is limited to the context of VM placement and thus does not provide sufficient expressiveness for the various applications in the complete cloud computing landscape. Finally, C^2L does not consider the matching of user expectations with provider offers.

S4P [BMB10] focuses on matching privacy policies of users to those of service providers. To this end, S4P policies are specified in a first-order language. Policies of users and service providers are then compared using formal methods. This approach aims at realizing functionality and does not consider minimizing storage or processing overheads.

FLAVOR [TM11] focuses on legal rules which define consequences for infringements. To this end, FLAVOR does not only specify which policies a system should adhere to, but also which actions have to be taken if a posteriori verification detects a policy breach. FLAVOR's logic expressions enable specification of obligations with deadlines, triggers for external events, and context information. However, while focusing on a posteriori verification, it does not consider matching of expectations and offerings of a service provider. Moreover, this approach does not address the storage overhead of policies.

To conclude, our analysis shows that no existing policy language supports all requirements for fine-grained privacy protection in the context of cloud computing. Most notably, existing languages either do not achieve a sufficiently small storage size to enable policies on a per-data item level or do not provide the necessary expressiveness and extensibility to cope with future, yet unknown privacy requirements. Furthermore, most existing works do not consider the importance of matching efficiency although this determines applicability for various upcoming scenarios, e.g., in the context of the IoT and big data.

4.2.2 Design of a Compact Privacy Policy Language

In the cloud computing landscape, fine-grained user-centric privacy policies are a practical and much-needed asset to achieve a DHRs-aware cloud stack. The main goal of our work is to fill the identified gap between the requirements for privacy policies (Section 4.2.1.2) and existing approaches (Section 4.2.1.3): We require a high level of expressiveness, the possibility to match users' expectations against data handling properties offered by cloud providers, and a minimal storage footprint at the same time.

To achieve this goal, we present CPPL, a compact privacy policy language which relies on a two-step approach: First, a privacy policy is specified in a human-readable representation (as in related work). Here, we derive a policy representation that is as expressive as related work. In a second, novel step, we compress this policy by taking advantage of flexibly specifiable domain knowledge. Notably, any further processing of the privacy policy, e.g., interpretation at nodes in the cloud, takes place directly on the compressed policy.

We depict an overview of our core design idea behind CPPL in Figure 4.4. Here, a user defines her privacy policy in a human-readable representation (Section 4.2.2.1), possibly using a GUI or an editor. Our *policy compressor* uses this representation and a set of *domain parameters* to derive the compressed policy (Section 4.2.2.2). These domain parameters define a *CPPL dialect* for a specific application scenario or deployment domain and define the variables and values that can be expressed

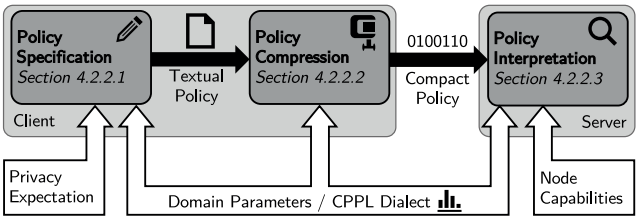

Figure 4.4 The core idea of CPPL is the compression of privacy policies by incorporating and leveraging flexibly specifiable domain knowledge.

in a privacy policy. Each dialect is specified by a central entity, e.g., a standardization organization. When interpreting a policy (Section 4.2.2.3), CPPL uses the compressed policy, the domain parameters, and *node capabilities* of the cloud node in question to evaluate whether the policy can be fulfilled by this node.

Consequently, the design of CPPL has three parts: (i) specification of policies, (ii) compression of policies, and (iii) interpretation of policies. We discuss our design for these three parts in the following and provide a complete example of CPPL's specification, compression, and interpretation of policies in Appendix A.1.

4.2.2.1 Specification of Policies

For our specification of policies, we derived a common pattern from the privacy policy languages in related work: Policies typically specify rules that list allowed (or forbidden) actions and these individual rules can be combined using conjunction or disjunction. Hence, CPPL allows users to express their privacy policies as policy atoms (e.g., `location = "DE"`) which are connected by Boolean algebra. Our use of Boolean algebra is deliberate since it enables even less technically proficient users to determine the semantical meaning of a policy and affords for fast interpretation of a policy. However, CPPL is not inherently bound to this representation of privacy policies as Boolean formulas and can conceptually also work with other policy representations, e.g., XACML [Oas13] and its derivates. Hence, with CPPL we do not propose a completely new policy language (in terms of what can be expressed) but rather show how to combine the concepts of existing policy languages with domain knowledge to achieve large policy size reductions.

We depict an example of CPPL's human-readable policy specification in Listing 4.1. In this example, data must not be stored at `CompanyA`, access to data must be logged, data has to be deleted after a certain point in time, backups have to be kept for one month, and the replication factor must be at least two. Furthermore, data has to be stored in Germany or, alternatively, in encrypted form in the EU.

As the foundation for deriving CPPL's human-readable policies, we provide the complete underlying formal grammar of our policy language in Listing 4.2. In the following, we describe the important parts of this specification which later allow us to derive an efficient policy compression by incorporating domain knowledge.

```
  provider != "CompanyA"
& log_access = true
& deleteAfter(1735693210)
& backupHistory("1M")
& replication >= 2
& ( location = "DE" | (location = "EU" & encryption = true) )
```

Listing 4.1 Example of CPPL's human-readable policy that imposes restrictions on the storage provider, location, and lifetime. It also enforces logging, backups, replication, and, depending on the location, encryption of the corresponding data item.

```
R → variable_bool ; !variable_bool
R → variable_number = value_number ; variable_number ≠ value_number ;
    variable_number < value_number ; variable_number ≤ value_number ;
    variable_number > value_number ; variable_number ≥ value_number
R → variable_string = value_string ; variable_string ≠ value_string
R → variable_enum = value_enum ; variable_enum ≠ value_enum
R → function(parameter_1, ..., parameter_N) ; !function(parameter_1, ..., parameter_N)
F → R ; !F ; (F) ; F & F ; F | F
```

Listing 4.2 In CPPL's policy grammar, relations specify a comparison between variables and values, functions add support for triggers, and Boolean interconnections of these relations and functions (R) create a policy formula (F).

Policy Atoms. Each CPPL policy is constructed out of different atoms, such as variables, relations, and functions. Properly differentiating between the different individual atoms of a policy enables efficient compression later on.

Variable Types: To allow for the expression of a wide range of requirements, CPPL uses variables of different types. We differentiate between Booleans, numeric variables (integers and floats of different sizes), and strings. To ease compression of variables with a predefined set of values, we additionally support enumerations, i.e., a set of values with the same type. For example, `["DE", "FR", "US", "GB", "NL", "EU"]` is an enumeration of type string which encodes a set of country identifiers.

Relations: To express single requirements within a policy, CPPL allows to compare variables using relations. Relations hence afford the comparison of required (as specified in the policy) and actual environmental context, i.e., the properties provided by a cloud node. Which relations can be used to compare variables depends on the variable type. CPPL allows to compare Boolean variables using negation (!), string variables by testing for equality ($=, \neq$), and additionally supports ordering ($>, \geq, <, \leq$) for numeric variables. A relation evaluates to *true* if and only if the comparison evaluates to *true*.

Functions: Requirements with very flexible input, such as event-based triggers (e.g., notification upon data access) and time-based triggers (e.g., performing backups within specific time frames or requiring data deletion at a specific point in time), cannot be expressed using relations in a scalable fashion. Hence, we support the specification of functions that allow general purpose computations to derive whether a node supports the requirements stated by a user. In a CPPL policy, functions consist of a function name and a list of parameters (e.g., `backupHistory("1M")`).

Similar to relations, a function evaluates to *true* if and only if the node supports the expectation given by the function and its parameters.

Policy Formulas. We construct privacy policies out of the above relations and functions by interconnecting them with Boolean operations, i.e., *and* (&), logical *or* (|), and *negation* (!). To allow for concise formulas and increase readability, distinct parts of a policy can be grouped together with (nested) brackets.

Domain Parameters. We, in contrast to related work, incorporate domain knowledge to compress policies. That is, CPPL can be parameterized to the individual use case through *domain parameters*, i.e., which variables are available, which values they can take, and which functions (and parameters for a specific function) can be utilized. Such domain specifics heavily depend on the individual use case. For example, the available variables might differ between a cloud-only and an IoT deployment. Together, domain parameters form a *CPPL dialect* which is provided by a central entity, e.g., a standardization body, for each use case.

For each variable, the domain parameters specification states name and type, e.g., *Boolean*, *string*, or *int32*. Similarly, the specification also lists the available functions together with the types of the functions' parameters. For enumerations, i.e., a set of values with the same type, the specification lists all possible values. CPPL dialects help us to realize three essential properties of policy languages: First, they provide users with a *list of all possible types of DHRs* they can specify in their privacy policies for a certain deployment domain. Second, they enable *verification of a policy*, i.e., that it contains only valid variables and values. Third, they allow to *extend the policy language* to new demands in existing or new use cases. Notably, domain parameters are not defined by individual users and we expect them to stay rather static, occasionally being superseded with an updated version similar to the introduction of new versions of network protocols or other standards.

4.2.2.2 Compression of Policies

The centerpiece of our approach is the compression of privacy policies by taking advantage of specifiable domain parameters. To achieve a high compression ratio, we introduce the domain parameter specification to be able to incorporate domain knowledge into the compression step. The domain parameter specification lists the available variables, functions, and values in a well-defined order. This enables us to replace variable and function names with a numerical identifier for their position in the domain parameter specification. We can employ a similar approach to considerably reduce the size of enumerations.

A compressed CPPL policy consists of four different parts as illustrated in Figure 4.5: (i) the *policy header* stores an identifier for the domain parameter specification the policy relates to, (ii) the *formula stack* stores the Boolean operations connecting the relations of a formula, (iii) the *relation stack* encodes relation information, and (iv) the *variable stack* stores the numerical variable and function identifiers as well as actual values and parameters. With this separation, we can leverage redundancies in privacy policies for compression. Most notably, if a relation, variable identifier,

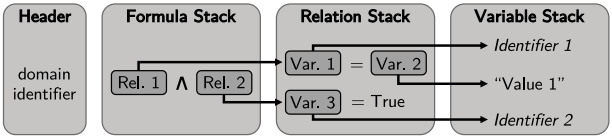

Figure 4.5 A compressed CPPL policy consists of a header and three stacks which reference each other to leverage redundancies for compression.

or value is used more than once in a policy, we need to store it only for the first occurrence and can reference it subsequently. In the following, we describe the encoding and compression of the parts of compressed CPPL policies in more detail.

Policy Header. To achieve a high compression rate, CPPL makes heavy use of information derived from the domain parameters defined in the used CPPL dialect. Hence, it is necessary to know a policy's CPPL dialect when interpreting the resulting compressed policy. As we cannot assume that this is always implicitly given by the context, we explicitly add the CPPL dialect in a 16 bit identifier field. As we strive for space efficiency, CPPL's policy header contains no further information. Especially, we completely waive length fields (which are common for bit level encodings) as they introduce constant space overhead and reduce flexibility by constraining the overall possible policy size. Instead, we encode lengths using special symbols and implicit knowledge derived directly from the encoded policy.

Formula Stack. We introduce a formula stack to encode interconnection of relations, i.e., logical operations, evaluation order as given by brackets, and references to relations. The overall goal of CPPL is to do this as space efficient as possible while still allowing for fast interpretation of the underlying policy. To save the space for an explicit encoding of the evaluation order, we rely on polish notation in the formula stack. While this automatically provides the correct evaluation order, we still require a space-efficient encoding for combining references to relations using logical operations. To achieve this goal, we follow two paths: (i) we reduce the number of logical operations that need to be encoded in the formula and (ii) we minimize the space required for referencing relations.

We reduce the space for encoding logical operations by deferring the handling of negations to the relation stack through De Morgan's theorems [CCM10]. Thus, we only need to differentiate between *and* and *or*, which can be encoded with only one bit. Alternatively, we could employ logic synthesis approaches for hardware circuit design [Mic94] to minimize the size of the Boolean formula or optimize its representation for fast execution. Such an alternative approach, however, would considerably increase the effort required for compressing a privacy policy.

To reduce the space required for referencing individual relations, we order the relation stack according to the position of relations in the formula stack. Hence, we can omit references to relations in the formula stack and simply refer to the *next relation* on the relation stack. While this allows us to reference relations very space efficiently, it prevents referencing one relation more than once (and thus save space by leveraging redundancies). To overcome this limitation, we introduce the concept

of a *redundant relation* which allows referencing a relation that has already been used in the same formula. The address (or offset) of the referenced relation is specified in a fixed-size bit sequence[8].

Based on these optimizations, we only need to encode *and*, *or*, *next relation*, and *redundant relation* in the formula stack, which can be encoded with two bits. However, this does not allow us to signal the end of the formula stack (as discussed above, we refrain from using length fields as this would increase the policy length). To still be able to signal the formula stack's end, we introduce an additional bit to the *redundant relation* symbol to signal the end of the formula stack. Consequently, this adds an overhead of one bit to redundant relation identifiers (which in most scenarios is the least used of all four symbols).

Relation Stack. Similar to the formula stack, the relation stack encodes the interconnection of variable identifiers and variable values through relations. We use three bits to encode the relation types $=, \neq, <, \leq, >, \geq, = True$, and $= False$. Each relation type is followed by two respectively one (for $= True$ and $= False$) *next variable* and/or *redundant variable* symbols encoded in a single bit each. As for redundant relations, we add a fixed-size address to a variable on the variable stack after the redundant variable bit. In contrast to the formula stack, we do not explicitly signal the end of the relation stack as we can directly derive the number of relations on the relation stack from the formula stack.

Variable Stack. The variable stack encodes the variables (including functions) used in a CPPL policy. Each variable is represented by an encoding of its type followed by a type-dependent representation of the variable value. To encode the variable type, we differentiate between variable identifiers (where values are instantiated by the cloud node interpreting a policy later on) and actual values (where values are already defined in the policy). We can derive all possible variable types from the domain parameter specification and hence encode, i.e., enumerate, variables according to their order in the specification. Hence, the number of bits required for encoding variable types depends on the domain parameter specification. A reasonable set for variable types, similar to major programming languages, contains Booleans, integers (8 bits to 64 bits, signed and unsigned), doubles, strings, enumerations, and functions. Additionally, we reserve one encoding for variable identifiers, i.e., a reference to a variable in the domain parameter specification that will be instantiated by a cloud node when interpreting the policy. Hence, four bits suffice to distinguish between a sufficient amount of different variable types and variable identifiers.

The encoding of a variable type is followed by a type-dependent representation of the variable value as described in the following (encoding for new variable types can be easily deduced). First, variable identifiers are encoded as numbers as given by their order of appearance in the domain parameter specification. The number of bits required for this is determined by the number of variables in the specification. Boolean values are encoded as a single bit, integers and floats are encoded with

[8] We chose a fixed-size length to save overhead for a length field. This fixed-size length constitutes a trade-off between the number of relations that can be addressed and the space required for encoding references. As this trade-off is domain specific, we allow configuring the fixed-size length in the domain parameter specification.

their respective bit size, and strings are encoded as null-terminated ASCII values. When encoding numbers, we automatically use the smallest possible representation, e.g., a 32 bit integer will be automatically casted to an 8 bit integer if possible. For enumerations, we derive the encoding from the position in the sorted list of possible values for this enumeration. The variable type of an enumeration can be derived from the identifier of the variable which the value in the enumeration is compared to. Finally, we encode functions by numbering their positions in the specification. Following the identifier for the function, we can directly encode the function's parameters, as their types are already defined in the specification.

Similar to relations in the relation stack, the number of variables in the variable stack can be derived from information in the relation stack. Hence, we do not need to encode the end of the variable stack and, thus, the end of a policy.

4.2.2.3 Interpretation of Policies

Once a CPPL policy has been compressed, it can be attached to data that is sent to cloud nodes. To this end, the policy can either be directly encoded together with the data, e.g., in a JSON-object (cf. Section 5.2.2.3) or transmitted alongside the data, e.g., by including it in individual network packets (cf. Section 4.2.4). Each cloud node that receives the data together with the annotated policy interprets the policy, i.e., compares its own node capabilities to the requirements specified in the policy. We first discuss these *node capabilities* in more detail before we present the actual process of *policy interpretation*.

Node Capabilities. The goal of privacy policy languages is to formulate requirements on the handling of data. This is predominantly achieved by comparing requirements to environmental context and supported triggers, i.e., the capabilities of a specific cloud node [PM11]. Only if the capabilities of a cloud node match the requirements formulated by the user, this node is allowed to process the corresponding data. Essentially, node capabilities denote for each variable name in a domain parameter specification the values supported by this specific node. Furthermore, node capabilities specify for each function defined in the domain parameter specification if it is supported by this node. If a cloud node supports a function in general, the node uses a small script or binary executable to check if it supports the parameters specified for this function as well.

Policy Interpretation. When interpreting a policy, i.e., deciding whether a cloud node supports the requirements expressed in this policy, we replace the variable identifiers in the policy with the values listed in the node capabilities. To check if a node supports the functions in the policy, we extract the parameters and evaluate their support using the corresponding implementation of these functions of this node (see above). Finally, we evaluate the individual relations and then the complete Boolean formula. A node is eligible to process the data if and only if the Boolean formula evaluates to true.

During policy interpretation, we apply logical operations in the order given by the formula stack. This is possible since the polish notation eliminates all brackets.

When iterating over the formula stack to find the start of the relation stack, we sequentially push the operations onto a stack, obtaining reverse polish notation for the actual execution. Furthermore, we cache the result of each relation's interpretation to save processing time for redundant relations. Notably, a policy does not necessarily define an unambiguous handling of data, i.e., there may be more than one satisfying assignment. To cope with this challenge, we additionally employ backtracking based on cached evaluation results to derive the actual variable assignment that has to be adhered to by the cloud node.

4.2.3 Evaluation

To assess the feasibility and applicability of our approach, we implemented CPPL in `C++` based on the `Boost` libraries. We utilize `Flex++` and `Bison` to automatically generate the scanner respectively parser to process CPPL's textual policies. Furthermore, we realize the domain parameters and node capabilities specification using JSON and parse them with `jsoncpp`.

We first perform synthetic benchmarks to get a thorough view of the performance and scalability of CPPL and then realize policies for real-world scenarios which enables us to compare CPPL to related work. Based on this, we study the applicability of CPPL in two cloud-based use cases: (i) storing millions of IoT messages and (ii) matching thousands of policies when performing machine learning in the context of big data.

4.2.3.1 Influence Factors on CPPL's Performance

Performance and scalability of CPPL are influenced by the policy size and the volume of domain parameters in the domain parameter specification. To thoroughly quantify both the influence on performance and scalability, we perform synthetic benchmarks for which we utilize a local test setup that consists of a desktop-grade machine (Intel i7 870, 4 GB RAM, Ubuntu 14.04). For each measurement point, we performed 100 runs and report the mean value with 99 % confidence intervals. We do not consider the overhead for initializing CPPL (in the order of 1.2 ms for compression and matching), e.g., for loading and parsing the domain parameters specification, as this has to be performed only once when the system is started.

Influence of Policy Size

To evaluate the influence of the policy size on the storage footprint, compression runtime, and matching runtime, we perform measurements with fixed domain parameters specifying 100 Boolean, integer, and string variables, each. We construct policies with up to 150 relations of the same variable type, allowing up to 50 (integer, strings) and 2 (Boolean) actual values, respectively. First, we explicitly evaluate a scenario without introducing redundancies for variables or relations (Relations 1 to 50). To study the impact of redundant variables, we then repeat already used variable values without repeating relations (Relations 51 to 100). Finally, to also study

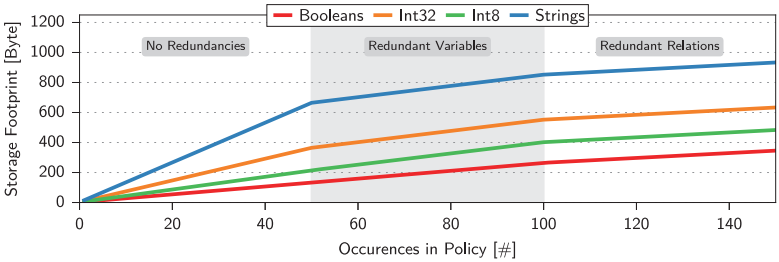

Figure 4.6 When increasing the policy size, redundant variables, relations, and integer optimization improve compression and hence reduce storage footprint.

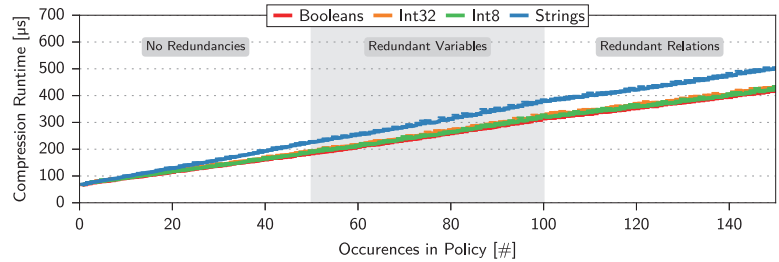

Figure 4.7 Compression runtime scales linearly with increasing policy sizes. Strings have a slightly higher increase in runtime.

the effect of redundant relations, we duplicate the first 50 relations (Relations 101 to 150). We use two integer sizes to evaluate CPPL's effect of automatically downsizing integers. While the domain parameters always specify integers with 32 bit, we use values whose representation either requires 32 bit or only 8 bit in the policy.

We first study the resulting *storage footprint* of a compressed CPPL policy in Figure 4.6. Without the possibility to leverage any redundancies, CPPL's policy size scales linearly, e.g., when considering a 32 bit integer from 9 byte for 1 relation to 364 byte for 50 relations. When introducing redundant variables, we observe a compression gain for strings (ratio 3.53) and 32 bit integers (ratio 1.93). In contrast, 8 bit integers and Booleans do not profit from redundant variables as the identifier for redundant variables would also consume 8 bit. Redundant relations allow for a further compression gain regardless of the variable type, e.g., by a ratio of 2.31 for 32 bit integers. Finally, the smaller storage overhead of 8 bit integers compared to 32 bit integers highlights the advantage of CPPL's automatic integer downsizing.

Next, we show in Figure 4.7 the *compression runtime*, i.e., the time for transforming CPPL's textual policy into its compressed representation, depending on the policy size. Here, we observe that compression runtime scales linearly with increasing policy sizes. More specifically, compression runtime increases from 68 µs for 1 relation to between 418 µs and 431 µs (502 µs for strings) for 150 relations. To put these numbers

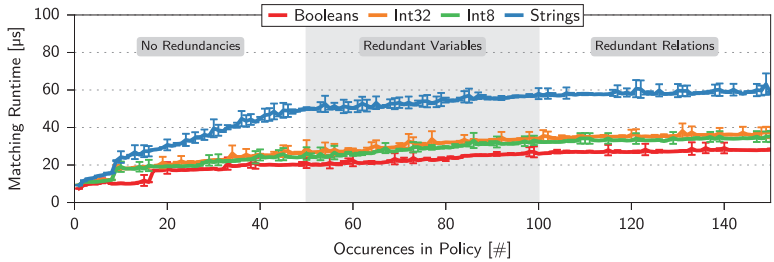

Figure 4.8 Larger policies result in an increased matching runtime. Leveraging redundancies helps to reduce matching runtimes for strings.

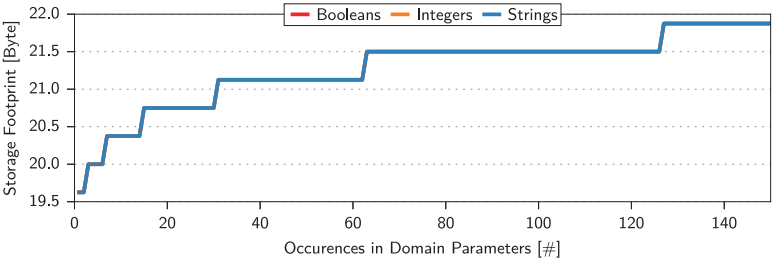

Figure 4.9 Increasing the number of domain parameters (and hence expressiveness) logarithmically increases the storage footprint.

into perspective, CPPL is thus able to compress 1993 to 14 754 policies per second (depending on the policy size). Strings show a slightly higher overhead due to slower encoding and comparison in the redundancy search. Redundant variables or relations do not noticeably influence compression runtime.

Finally, we show the *matching runtime*, i.e., the time for matching a compressed CPPL policy against node properties, in Figure 4.8. Matching (which is performed in the backend and thus typically more often than compression), happens faster than compression with a linear increase in runtime for growing policy sizes. Without the possibility to remove redundancies, the matching time for strings increases from 9 µs for 1 relation to 50 µs for 50 relations. In contrast, the matching time for Booleans increases from 7 µs for 1 relation to only 21 µs for 50 relations. Matching times for integers are slightly higher than for Booleans. Especially for strings, we observe a benefit of removing redundancies, reducing processing for strings for 150 relations to 58 µs. Consequently, CPPL is able to process 17 126 to 134 048 policies per second on a desktop-grade machine.

Influence of More Comprehensive Domain Parameters

We now evaluate the impact of more comprehensive domain parameters, i.e., a larger variety of variables that can be used in a policy, on policy size and processing time.

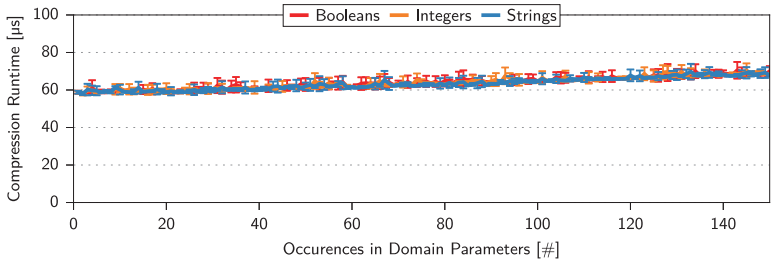

Figure 4.10 CPPL shows a tendency for only a slight linear increase in compression runtime for a growing amount of domain parameters.

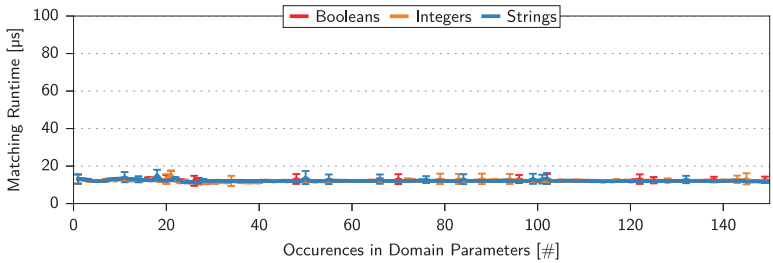

Figure 4.11 CPPL's matching runtime does not depend on the amount of domain parameters and hence stays constant at a very low level, resulting in a large throughput.

To this end, we use a static CPPL policy consisting of one Boolean, one integer, and one string relation. For each of these variable types, we increase the number of domain parameters from 1 up to 150. Here, we do not differentiate between different integer types as the actual values only appear in the (fixed) policy.

We show the resulting *storage footprint* in Figure 4.9, where all three lines lie on top of each other, i.e., only the topmost line is visible, because all variable types exhibit the exact same behavior as only the encoding of variable identifiers is impacted by an increase in the number of available domain parameters. In this case, CPPL requires more bits to encode variable identifiers, which is not affected by the variable type. We observe an increase from 19.63 byte for 1 variable definition to 21.88 byte for 150 variable definitions. More specifically, domain parameters that specify n variables require $\lceil \log_2(n) \rceil$ bits to encode one identifier in the variable stack.

When considering the influence of an increasing amount of domain parameters on the *compression runtime* as shown in Figure 4.10, we observe a tendency for a linear runtime increase. More precisely, the compression runtime increases from 59 µs for 1 variable to 70 µs for 150 variables. This results in approximately 14 288 to 17 004 policy compressions that CPPL can perform per second.

Figure 4.11 shows that the *matching runtime* is not influenced by an increasing set of domain parameters (independent from the variable type) as tens to hundreds

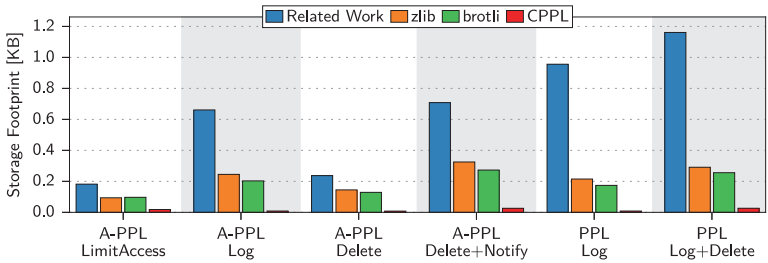

Figure 4.12 When considering real-world privacy policies, CPPL considerably reduces storage footprint compared to related work and generic compression methods.

of domain parameters can efficiently be cached in memory. All observed matching runtimes are in the order of 12 µs to 13 µs. Consequently, CPPL is able to match 76 453 to 85 251 policies per second in this setting.

4.2.3.2 Comparison to Related Work

To prove the feasibility of CPPL, we also evaluate CPPL on real-world policies taken from state-of-the-art related work in the context of cloud computing. To this end, we were able to locate six XML-based policies, namely four A-PPL policies (one limiting access based on location, purpose, and time conditions [AEÖ+14]; one logging access, deletion, and sent operations; one specifying a deletion date; and one defining a deletion date and notification on deletion [CDG+13]) and two PPL policies (one specifying logging of three different actions and one extending the former with a deletion date [PPL14]).

Resulting Policy Sizes

First, we analyze the required storage size for real-world policies both for CPPL and related work. To this end, we compare the original XML representation (without superfluous whitespace characters) of the policy in A-PPL and PPL, respectively, with equivalent CPPL policies. As CPPL uses a compressed format, we also applied `zlib` and `brotli`, two state-of-the-art compression libraries, to the policy representations from related work to also compare against generic compression methods.

We depict the resulting policy sizes in Figure 4.12. Overall, `zlib` and `brotli` achieve a clear compression gain, however, CPPL achieves by far the smallest policy size. For large policies, `zlib` and `brotli` achieve a compression ratio of 2.18 up to 5.49 while CPPL reduces the size of the policy by a ratio of 27.10 up to 112.47. For smaller policies, `zlib` and `brotli` perform worse, achieving a compression ratio of only 1.63 up to 1.94 while CPPL still manages to achieve a reduction by a ratio of 9.97 up to 29.63. In absolute numbers, CPPL is able to reduce *A-PPL LimitAccess* from 182 byte to 18.25 byte and *PPL Log* from 956 byte to only 8.5 byte. As we show in Section 4.2.3.3 in the context of storing IoT data in the cloud, this results

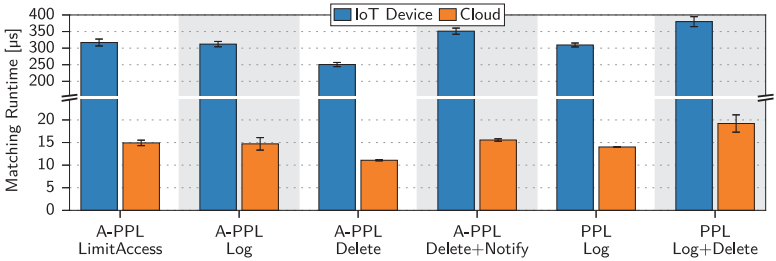

Figure 4.13 The real-world matching runtimes for IoT and cloud class devices show that even on IoT devices, CPPL can perform thousands of matchings per second.

in an enormous reduction of the overall required storage space. Furthermore, and in contrast to generic compression methods such as `zlib` and `brotli`, CPPL affords for policy evaluation directly on the compressed policy representation without the need for decompressing the policy first.

Real-World Performance

Based on these real-world policies, we evaluate CPPL's performance on commodity cloud infrastructure as well as on IoT devices. For the evaluation of CPPL on cloud infrastructure, we use an Amazon Web Services EC2 64-bit instance of type m4.large [AWS18b] running Ubuntu 14.04 as the operating system. To measure the performance of CPPL on IoT devices, we utilize a Raspberry Pi (Model B Revision 2.0) with a 700 MHz ARM11 CPU, 512 MB of RAM, and running Raspbian 8.0 as the operating system.

The evaluation results in Figure 4.13 show that a cloud server can perform more than 52 056 policy matchings per second for our largest real-world policy. For smaller policies, this increases to more than 67 024 matchings per second. To put these numbers into perspective, even Dropbox had on average less than 20 000 insert/update requests per second in June 2015 [Dro15]. For IoT devices, the matching rate still ranges from 2632 up to 3155 matchings per second. This is more than sufficient to process all messages in an actually deployed IoT platform (cf. Section 4.2.3.3), with a largest observed throughput of 149 messages per second. Thus, we enable policy awareness for cloud-based scenarios ranging from the data collection by IoT devices to the processing and storage of data in the cloud.

4.2.3.3 Applicability of CPPL

To demonstrate the applicability of per-data item policies as a foundation for a DHRs-aware cloud stack in general and CPPL in specific, we analyze the policy-induced storage overhead for data measurements in the cloud-based IoT and investigate the impact of policy support for the runtime of machine learning approaches in the context of cloud-backed big data.

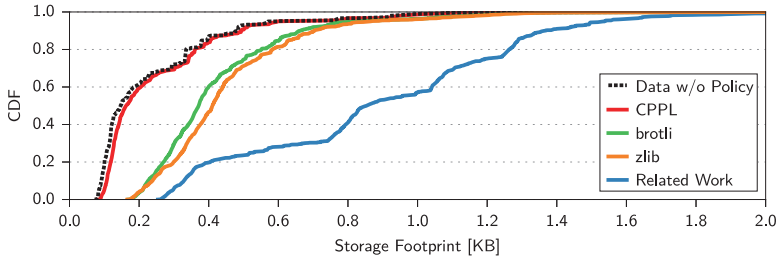

Figure 4.14 CPPL considerably reduces the storage footprint of real IoT data in the cloud compared to privacy policies from related work.

Storage Overhead in the IoT

The IoT not only causes a massive growth in the amount of transferred data in the Internet, e.g., up to expected 40 000 exabytes in 2020 compared to 130 exabytes in 2005 [GR12], but also substantially increases the diversity of data sources [BWHT12] and the granularity of reported data [HHCW12]. Hence, the question arises whether it is also feasible to attach per-data item policies to IoT data when sending it to the cloud, where it is then stored and processed.

To study the impact of per-data item policies on IoT data in the context of cloud computing, we sample frequency and size of real IoT data and analyze the storage overhead of attaching privacy policies to it. We collected real data of IoT devices from the API of `dweet.io` [Bug18], a cloud-based data platform for the IoT. Our dataset, which we collected over a period of 92 hours, consists of 18.41 million IoT messages originating from 7207 distinct devices. The sizes of the IoT messages we collected range from 72 byte to 9.73 KB with a mean size of 394 byte. Although this data is publicly available through `dweet.io`'s API, we took appropriate measures to protect the privacy of people potentially monitored by the IoT devices. To this end, we only stored the identifier of the device and the timestamp of each data message. Furthermore, we sampled only one message per device to derive a representative message size and solely stored the resulting message size (not the payload).

Figure 4.14 shows the cumulative distribution function of IoT message sizes with (solid lines) and without (dashed line) attached per-data item policies. We uniformly randomly assign one of the policies from related work (cf. Section 4.2.3.2) for each IoT message and compare originally uncompressed policies to policies compressed with `zlib` and `brotli` as well as CPPL. These results show that CPPL adds only a negligible storage and transmission overhead compared to data without per-data item policy, while generic compression algorithms and especially uncompressed policies induce considerably higher storage overheads. In total, storing all 18.41 million collected IoT messages without any attached policies requires 4.39 GB of storage space. This increases to only 4.68 GB when attaching CPPL policies, 7.86 GB and 8.42 GB for `brotli` and `zlib`, respectively, and a total of 16.37 GB when using uncompressed policies from related work. As these numbers correspond to less than

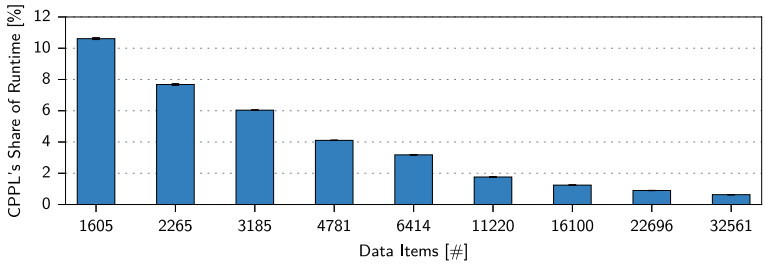

Figure 4.15 Our study of the impact of CPPL on machine learning (UCI Adult dataset [Pla99]) shows that CPPL's share of the runtime becomes negligible for larger data sets.

four days of IoT device usage, this clearly highlights the necessity for space efficient privacy policy languages and the reasonable storage overhead of CPPL.

Policy Matching for Big Data

The large storage space provided by cloud computing is especially interesting for machine learning in big data which benefits from larger datasets for training models to increase their accuracy [Loh12]. Per-data item policies can increase the willingness of individuals to contribute their data as they enable users to stay in control over their data. However, policies lead to additional processing overhead to determine if a policy allows usage of a specific data item for the desired application.

To investigate the performance of CPPL in this application scenario, we measure the overhead of policy matching when it is used to determine if data items can be used for a machine learning-based study. We compare execution times of the training phase of the support vector machine `LIBSVM` [CL11] with the time required to process CPPL policies for this input data (we uniformly randomly assigned one of the policies from related work to each input and considered policy initialization overhead for the first occurrence of each domain parameter specification).

Figure 4.15 shows the share of the runtime that is required for policy processing for the different numbers of input records of the nine different UCI Adult datasets [Pla99]. That is, the remaining share of the runtime is required for the actual training of the support vector machine (SVM). For a very small number of records, processing of policies takes 10.6 % of the runtime that is required for the full process (policy processing and training of the SVM). More specifically, policy processing accounts for 18.9 ms while the training of the SVM requires 178.2 ms. However, with increasing number of data records used for training the SVM, the fraction of the time required for processing of policies considerably decreases. Considering, e.g., 32 561 data records, policy processing is responsible for only 0.6 % (377.7 ms) of the total runtime whereas training of the SVM accounts for the other 99.4 % (59.8 s). Hence, for larger datasets in the context of cloud-based big data, the runtime overhead for CPPL policy processing is negligible. Thus, CPPL enables privacy policy-aware machine learning-based approaches with almost no overhead on processing time.

4.2.4 Summary and Future Work

We presented CPPL, a compact privacy policy language as a foundation for our vision of a DHRs-aware cloud infrastructure. CPPL allows users to specify their privacy requirements regarding routing, processing, and storage of data when it is outsourced to the cloud in a two-step approach: The user first defines a privacy policy in a human-readable representation (as with traditional privacy policy languages). Then, in a second, novel step, CPPL compresses this policy, thereby optimizing the resulting policy size down to the bit level. To this end, CPPL takes into account the specific deployment domain and extensively utilizes domain knowledge to further reduce policy sizes. Our concept of domain parameters allows for easy adaption to new, even yet unforeseen use cases. CPPL further distinguishes itself from related work by its focus on reducing policy sizes and processing overheads.

In the following, we briefly discuss how CPPL achieves the requirements that *any* privacy policy language in the context of cloud computing must fulfill (cf. Section 4.2.1.2). Our benchmarks show that CPPL indeed achieves a *minimal storage footprint*, in which we significantly outperform related work. More specifically, CPPL reduces policy sizes by up to two orders of magnitude. Furthermore, when considering the storage of massive amounts of IoT data, CPPL, in contrast to related work, adds only a marginal storage overhead. At the same time, our measurements illustrate that CPPL allows for *efficient policy checking* and is viable for real-world scenarios at large scales. This is indicated by CPPL's ability to perform tens of thousands of policy matchings on a cloud server and still thousands of matchings on a resource-constrained IoT device. Likewise, CPPL can be used to express permissible data usage, e.g., in the context of big data.

By reformulating existing privacy policies in CPPL, we illustrate support for *incremental deployment*, e.g., by showing that CPPL is compatible with existing policy languages. Through our concept of cloud node capabilities in CPPL, we are able to realize *matching* of users' privacy expectations with the data handling capabilities offered by cloud providers. With our concept of domain parameters, we address the challenges of *expressiveness* and *extensibility* in CPPL. By combining Boolean expressions with run-time interpreted functions, we can cover all privacy requirements that are nowadays supported by related work simply by providing fitting domain parameters. Here, the concept of domain parameters are what makes CPPL extensible: If additional privacy requirements in one of CPPL's application domains emerge, CPPL can easily be extended to support these by merely updating the corresponding domain parameter specification. Similarly, if completely new application domains emerge, CPPL can be effectively adapted to those by creating a new domain parameter specification (CPPL dialect). As we propose to perform this process centrally, e.g., at a standardization body, we do not place any burden on users.

For future work, we mainly identify the application of CPPL in other deployment domains. We have already shown that CPPL is viable and promising in selected aspects of the IoT and big data. Yet, additional effort is required to show that CPPL can be realized on embedded devices with highly constrained processing, memory, and energy resources [HHH+17]. Furthermore, it remains to show that CPPL can

even be integrated with (network) protocols specifically tailored to the requirements of resource-constrained devices that build up the IoT and CPS [HHH+17].

When envisioning the Internet-wide deployment of CPPL, one extremely challenging yet very promising avenue for future work would be the integration of CPPL into network layer protocols to enable policy-based routing [KPPK11]. To this end, CPPL policies could be included in the header of network layer protocols such as IPv4 and IPv6, e.g., using the options field of IPv4 or the extensions concept of IPv6, where reducing the size of encoded policies is important, especially considering the severely limited space available in IPv4's option field. Similarly, the integration of CPPL with DNS would allow to *directly* address resources in a DHRs-compliant manner, e.g., when requesting resources from a CDN. To this end, CPPL policies could be encoded in the 253 characters available for DNS hostnames, providing roughly 36 bytes for encoding CPPL policies, which is sufficient for encoding real-world privacy policies (cf. Section 4.2.3.2).

When taking a legal standpoint, it would be extremely promising to apply CPPL to allow users to express their choices regarding DHRs and privacy in general as guaranteed by legislation such as the GDPR [GDPR16] and hence enable service providers to automatically process and adhere to the requirements. To provide accountability for privacy policies in this setting, e.g., to enable cloud providers to prove which specific policy a user supplied in case of complaints or lawsuits, CPPL could be integrated with approaches for packet authentication [HRGD08].

To conclude, CPPL realizes significant policy size reductions, which allows for per-data item policies and thus fine-grained privacy protection in cloud computing. This lays the foundation for realizing our vision of a DHRs-aware cloud stack. In the following, we present how annotating data with DHRs, e.g., based on CPPL, can be used to make a distributed cloud storage system comply with user-imposed DHRs.

4.3 PRADA: Practical Data Compliance for Cloud Storage

Now that we empowered users to express their DHRs in an efficient manner, we can provide cloud service and infrastructure providers with the knowledge required to respect these requirements while delivering their service. This is especially relevant for cloud storage systems such as distributed file systems, key-value stores, and databases that form the foundation for cloud infrastructure with respect to the handling of data by defining how users' data is stored on physical storage resources.

However, despite their popularity and importance as the underlying infrastructure for more complex cloud services, today's cloud storage systems typically do not account for compliance with regulatory, organizational, or contractual DHRs. Instead, the placement of data on cloud nodes is nowadays optimized with respect to reliability, availability, and performance. To this end, data in cloud storage systems is addressed using a specific key that is used to map data to cloud storage nodes, e.g.,

using a hash function similar to the concept of distributed hash tables in the context of peer-to-peer systems [WGR05, LGW06]. However, the cloud storage node to which data is mapped based on its key will generally not be able to comply with the corresponding DHRs. As a result, users nowadays have little control over compliance with DHRs when their data is outsourced to cloud storage systems.

While the benefits for supporting DHRs in cloud storage systems are widely recognized and highly sought-after by practitioners, support for them is still limited nowadays [Int12, WMF13]. So far, related work mainly considered the challenge of complying with DHRs while processing data in the cloud [IKC09, BKDG13, ELL+14], proposed approaches that solely restrict the storage location of data while ignoring other types of DHRs [PGB11, WSA+12], or considered the cloud storage system as a black box and hence targeted the enforcement of some, coarse-grained DHRs from outside the storage system, e.g., by distributing data between different cloud storage providers [PP12, WMF13, SMS13]. As a result, a practical solution for complying with arbitrary DHRs in cloud storage systems is still missing—a situation that is disadvantageous to both users and providers of cloud storage systems.

To overcome this limitation, we introduce PRADA, a transparent data handling layer which sits on top of legacy cloud storage systems and empowers users to request specific DHRs and provides operators of cloud storage systems with the necessary technical means to comply with stated DHRs. More specifically, our core idea is to augment cloud storage systems with one layer of indirection, which flexibly and efficiently routes data to cloud storage nodes according to the imposed DHRs. We demonstrate the design of our approach along classical key-value stores, while our approach conceptually also generalizes to more advanced storage systems such as Google's Spanner [CDE+13], Clustrix [Clu18], and VoltDB [SW13], which are widely used in real-world deployments. Concretely, we implement PRADA on top of the distributed database Cassandra [LM10, Apa18a] and show in our evaluation that complying with data handling requirements in cloud storage systems is practical in real-world deployments such as microblogging and distributed storage of email.

Our results show that we can realize compliance with DHRs in cloud storage systems at moderate costs. While PRADA results in a moderate increase of query completion times, we are able to keep storage overhead constant and realize a load distribution in the cloud storage cluster that is close to the theoretical optimum even in challenging situations. As we show, data without attached DHRs is not impaired by PRADA. Hence, users can choose for each piece of data whether compliance with DHRs is worth a modest decrease in performance. PRADA realizes compliance with DHRs when assigning data to storage nodes in a cloud storage system and thus provides an important building block for realizing our vision of a DHRs-aware cloud stack.

4.3.1 Data Handling Requirements in Cloud Storage Systems

With the increasing demand for sharing data and storing it with external parties [SV10], complying with DHRs becomes a crucial challenge for cloud storage systems [WMF13]. As a foundation for developing our approach to support compliance with

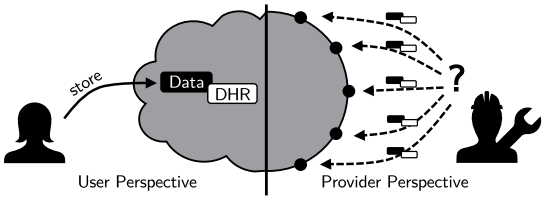

Figure 4.16 When users store data with DHRs in a cloud storage system, the provider is obliged to store it only on those cloud nodes that fulfill the stated DHRs.

DHRs in cloud storage systems, we briefly reiterate our setting and define the scope of our approach. Based on our analysis of DHRs (cf. Section 2.3.1), we derive a formalization of DHRs that allows us to provide support with all possible types of DHRs in our system. This leads us to our definition of a set of goals that must be reached by *any* approach that aims at adequately supporting DHRs in cloud storage systems. Using these goals, we study related work and discuss its relevance for realizing practical support for complying with DHRs in cloud storage systems.

4.3.1.1 Setting

We aim at supporting compliance with DHRs in cloud storage systems. To this end, we consider a cloud storage system that is realized over a set of diverse nodes that are spread over different data centers [GHMP08]. To explain our approach in a simple yet general setting, we assume that data is addressed by a distinct key, i.e., a unique identifier for each data item, similar to the approach of distributed hash tables that serve as a foundation for structured peer-to-peer systems [WGR05]. Key-value based cloud storage systems [DHJ+07, LM10, ÖV11, GHTC13] provide a general, good starting point for our line of research, since they are widely used and their underlying principles have been adopted in other, more advanced cloud storage systems [CDE+13, SW13, Clu18]. We discuss how our approach generalizes to other, more advanced types of cloud storage systems in Section 4.3.4.

As a basis for our discussion, we illustrate our underlying setting in Figure 4.16. Users (private and corporate, cf. Section 2.1.3) insert data into the cloud storage system and annotate it with their desired DHRs—as envisioned in the motivation behind a DHRs-aware cloud stack (cf. Section 4.1.1). These requirements are in machine-readable form, e.g., expressed using CPPL (cf. Section 4.2), and can be parsed and interpreted by the operator of the cloud storage system. Each user of the storage system might impose individual and varying DHRs for each single data item inserted into the storage system.

In this setting, compliance with DHRs then has to be achieved and enforced by the provider of the cloud storage system. Only the provider knows about the characteristics of its cloud storage nodes and only the provider can thus make the ultimate decision on which cloud node a specific data item should be stored. Different works exist that propose cryptographic guarantees [IKC09], accountability mechanisms

[ABF+04], information flow control [BEP+14, PSBE16], or even virtual proofs of physical reality [RMX+15] to relax trust assumptions on the cloud provider, i.e., providing the client with *assurance* that DHRs are (strictly) adhered to. Our goals are different: Our main aim is for *functional* improvements of the status quo. Thus, these works are orthogonal to our approach and possibly can be added on top of PRADA if users' trust in the cloud provider alone is insufficient.

4.3.1.2 Formalizing Data Handling Requirements

We base the design of PRADA on our analysis of existing and potential DHRs (cf. Section 2.3.1). To design for maximum flexibility and thus be able to cope with future requirements and storage architectures, we use our analysis of DHRs as a foundation to derive a formalized understanding of DHRs that also covers future, yet unforeseen requirements. Such a formalization of DHRs can then be realized by different privacy policy languages such as CPPL (cf. Section 4.2).

We distinguish different *types* of DHRs $T_i = (P_i, f_i)$. Here, $P_i = \{p_{i,1}, \ldots, p_{i,n}\}$ defines all possible *properties* which cloud storage nodes can support for a type of DHRs and $f_i(p_{i,l}, p_{i,m}) \rightarrow \{true, false\}$ constitutes a *comparison function* for two properties of the same type. This comparison function enables us to evaluate whether properties demanded by users are supported by cloud storage nodes. Hence, it is possible to compute the set of *eligible nodes* for a specified type of DHRs, i.e., those cloud nodes that can offer the desired properties.

A straightforward example for a type T_i of DHRs is the *storage location*. In this example, the properties p_i consist of all *existing storage locations* and the comparison function f_i tests two storage locations for *equality*. In a more complicated example, we consider as DHR type T_i the *security level of full-disk encryption*. Here, the properties p_i range from 0 bits (no encryption) to different bits of security (e.g., 192 bits or 256 bits), with more bits of security offering a higher security level [Bar15]. In this case, the comparison function implements $\geq$, i.e., all storage nodes that provide at least the requested security level are eligible to store the data.

By combining different types of DHRs and allowing users to specify a set of requested properties (e.g., different storage locations) for each type, we provide them with powerful means to express their DHRs. We provide more detail on how clients can combine different types of DHRs in Section 4.3.2.2 and how we integrate our formalization of DHRs into Cassandra's query language in Section 4.3.3.1.

4.3.1.3 Goals

Our analysis of real-world demands for DHRs based on legislation, business interests, and future trends (cf. Section 2.3.1) emphasizes the importance of supporting DHRs in distributed cloud storage systems. Based on our description of the underlying setting (cf. Section 4.3.1.1), we identify a set of goals that any approach that addresses the challenge of supporting DHRs in cloud storage systems has to fulfill:

Comprehensiveness: To address a wide range of DHRs, the approach should work with any DHRs that can be expressed as properties of a cloud storage node and should support the combination of multiple, different DHRs. In particular, it should support the requirements stated in Section 2.3.1 based on the formalization derived in Section 4.3.1.2 and be able to evolve and adapt whenever new DHRs emerge. Comprehensiveness is a qualitative goal which can be evaluated based on an analysis of the DHRs-aware cloud storage system.

Minimal Performance Impact: Existing cloud storage systems are highly optimized and trimmed for performance. Thus, the impact of offering support for DHRs on the performance of a cloud storage system should be minimized. The performance impact of supporting DHRs can be quantified by measuring the processing runtime for the individual storage system operations.

Cluster Balance: In existing cloud storage systems, the storage load of cloud nodes can easily be balanced to increase performance. Despite having to respect DHRs (and thus limiting the set of possible storage nodes), the storage load of individual cloud nodes should be kept as balanced as possible. Keeping the storage cluster balanced is a quantitative goal which can be assessed by measuring and comparing the load of the different cloud nodes in the storage system.

Coexistence: Likely, not all data will be accompanied by DHRs. Hence, data without DHRs should not be impaired by the availability of support for DHRs, i.e., data without DHRs should be stored and handled in the same way as in a traditional cloud storage system, especially with respect to performance. Ensuring that data without DHRs is not impacted by offering support for DHRs is a quantitative goal which can be evaluated by comparing the processing runtime for data without DHRs against those on an unmodified system.

4.3.1.4 Related Work

We categorize our discussion of related work by the different types of DHRs individual approaches address. In addition, we discuss approaches which provide users with assurance that storage providers adhere to DHRs.

Distributing Storage of Data. To enforce storage location requirements, one class of related work proposes to split data between different storage systems. Wüchner et al. [WMF13] and CloudFilter [PP12] add proxies between users and storage providers to transparently distribute data between different cloud storage providers according to DHRs, while NubiSave [SMS13] enables users to combine resources of different cloud storage providers to fulfill individual redundancy or security requirements. These approaches have in common that they can treat individual cloud storage systems only as black boxes. Consequently, they do not support fine-grained DHRs within the cloud storage system itself and are limited to a small subset of DHRs.

Sticky Policies. Similar to our idea of specifying DHRs, the concept of sticky policies [PM11] proposes to attach usage and obligation policies to data when it is outsourced to third parties. In contrast to our work, sticky policies mainly concern the purpose of data usage, which is primarily realized using access control. One interesting aspect

of sticky policies is their ability to make them "stick" to the corresponding data using cryptographic measures which could also be applied to PRADA. In the context of cloud computing, sticky policies have been proposed to express requirements on the security and geographical location of cloud storage nodes [PSM09]. However, so far it has been unclear how this could be realized efficiently in a large and distributed cloud storage system. With PRADA, we present an approach to achieve this goal.

Policy Enforcement. To enforce privacy policies when accessing data in the cloud, Betgé-Brezetz et al. [BKDG13] monitor access of virtual machines to shared file systems and only allow file access if the requesting virtual machine is fully policy compliant. In contrast, Itani et al. [IKC09] propose to leverage cryptographic coprocessors to realize trusted and isolated execution environments and hence enforce the encryption of data. Espling et al. [ELL+14] aim at allowing cloud service providers to influence the placement of their virtual machines in the cloud to realize specific geographical deployments or to provide redundancy by avoiding colocation of critical components. These approaches are orthogonal to our work, as they primarily focus on enforcing policies when processing data, while PRADA addresses the challenge of supporting DHRs when storing data in cloud storage systems.

Location-based Storage. Focusing exclusively on location requirements, Peterson et al. [PGB11] introduce the concept of data sovereignty with the goal to provide a guarantee that a provider stores data at claimed physical locations, e.g., based on measurements of network delay. Similarly, LoSt [WSA+12] enables verification of storage locations based on a challenge-response protocol. In contrast, PRADA focuses on the broader challenge of realizing support for arbitrary DHRs.

Controlling Placement of Data. Primarily focusing on distributed hash tables, SkipNet [HJS+03] enables control over data placement by organizing data mainly based on string names. Similarly, Zhou et al. [ZGS03] utilize location-based node identifiers to encode physical topology and hence provide control over data placement at a coarse granularity. In contrast to PRADA, these approaches need to modify the identifier of data based on the DHRs, i.e., knowledge about the specific DHRs of data is required to locate it, e.g., when requesting stored data. Targeting distributed object-based storage systems, CRUSH [WBMM06] relies on hierarchies and data distribution policies to control placement of data in a storage cluster. These data distribution policies are bound to a predefined hierarchy and hence cannot offer the same flexibility as PRADA. Similarly, Tenant-Defined Storage [MMV+17] enables clients to store their data according to DHRs. However and in contrast to PRADA, all data of one client needs to have the exact same set of DHRs. Finally, SwiftAnalytics [RZO+17] proposes to control the placement of data to speed up big data analytics. Here, data can only be put directly on specific nodes without the abstraction provided by PRADA's approach of supporting DHRs.

Hippocratic Databases. Hippocratic databases store data together with a purpose specification [AKSX02], which allows them to enforce the purposeful use of data using access control and to realize data retention after a certain period. Using Hippocratic databases, it is furthermore possible to create an auditing framework to check if a database is complying with its data disclosure policies [ABF+04]. However, this concept is limited to a single database node and does not support a distributed

setting, e.g., as required as a foundation for realizing cloud storage systems, where storage nodes have different data handling capabilities.

Assurance. To provide assurance that cloud storage providers indeed adhere to DHRs, de Oliveira et al. [OSGJ13] propose an architecture to automate the monitoring of compliance to DHRs when transferring data within the cloud. Bacon et al. [BEP+14] and Pasquier et al. [PSBE16] show that this can also be achieved using the concept of information flow control. Similarly, Massonet et al. [MNP+11] propose a monitoring and audit logging architecture in which infrastructure provider and service provider collaborate to ensure compliance with data location requirements. These approaches are orthogonal to our approach and could be used to verify that providers of cloud storage systems operate PRADA in an honest way.

Our discussion of related work shows that support for arbitrary DHRs in cloud storage systems is an open challenge. Related work either focuses on respecting DHRs during the processing of data in the cloud, develops specifically tailored solutions for supporting some carefully selected DHRs while storing data (often with respect to storage location), or treats the cloud infrastructure as a black box and hence aims at realizing some DHRs on a coarse granularity from the client side. To overcome these shortcomings of related work, we present the design of PRADA in the following. PRADA empowers users to request compliance with a comprehensive set of fine-grained DHRs when storing their data in cloud storage systems and enables the providers of these systems to efficiently and effectively realize compliance with user-dictated DHRs in a distributed storage cluster.

4.3.2 Supporting Data Handling Requirements

In this section, we describe the design of PRADA, our approach to support data handling requirements (DHRs) in key-value based cloud storage systems that meets the goals we derived in Section 4.3.1.3. The problem that has prevented support for DHRs so far stems from the common pattern used to address data in key-value based cloud storage systems: Data is addressed, and hence also partitioned (i.e., distributed to the cloud nodes in the cluster), using a designated key (i.e., a unique identifier for a piece of data which does not take into account DHRs). Yet, the *responsible node* (according to the key) for storing a piece of data often cannot fulfill the client's DHRs, e.g., because it is located in the "wrong" physical or jurisdictional location. Thus, the challenge faced by our work is how to realize compliance with DHRs and still allow for key-based data access in a distributed cloud storage system.

4.3.2.1 System Overview

To tackle this challenge, our core idea underlying PRADA is to add an indirection layer on top of a cloud storage system. We illustrate how we integrate this layer into existing cloud storage systems in Figure 4.17. The general idea of this indirection layer is to store a data item at a different node, called *target node*, whenever the responsible node cannot comply with the stated DHRs. To still enable the lookup

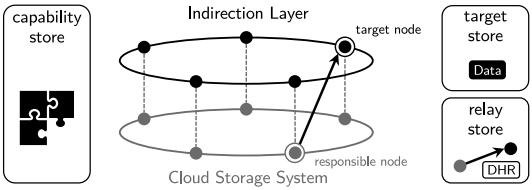

Figure 4.17 PRADA adds an *indirection layer* to support DHRs. The *capability store* records which nodes supports which DHRs, the *relay store* contains references to indirected data, and the *target store* saves indirected data.

of this data item (e.g., when a user wants to access her stored data), the responsible node stores a reference to the target node for this data item. As shown in Figure 4.17, we introduce three new storage components, i.e., (i) capability store, (ii) relay store, and (iii) target store to realize PRADA, as described in the following.

Capability Store: The global *capability store* is used to look up nodes that can comply with a specific DHR, similar to the concept of node capabilities for CPPL (cf. Section 4.2.2). In the context of this work, we consider all DHRs that describe properties of a storage node and range from rather simplistic properties such as storage location to more advanced capabilities such as the support for deleting data at a specified point in time using our formalized notion of DHRs (cf. Section 4.3.1.2). Notably, we focus on providing the possibility to account for such DHRs in our work. Hence, the concrete realization (e.g., the actual deletion of data) has to be realized by the provider of the cloud storage system in a second step and is considered out of scope for PRADA. To speed up lookups in the capability store, each cloud node keeps a local copy of the capability information. Depending on the underlying cloud storage system, the distribution of this information can either be realized by pre-configuring the capability store for all nodes in the cloud storage cluster or by utilizing mechanisms of the cloud storage system itself for creating a globally replicated view of the capabilities of the storage nodes.

Relay Store: Each cloud node operates a local *relay store* containing references to data this node is responsible for (based on the data's key) but stored at other nodes. More precisely, the relay store contains references to data the node itself is responsible for but cannot comply with the DHRs posed during insertion. For each data item, the relay store contains the key of the data, a reference to the target node at which the data is actually stored, and a copy of the DHRs.

Target Store: Each node stores data that has been redirected to it in a *target store*. The target store operates exactly as a traditional data store but enables a node to distinguish data that falls under DHRs from data that does not.

Relying on an indirection layer comes at the cost of increasing the time required for communication within the storage cluster and thus likely increasing query completion times. However, alternatives to adding an indirection layer are likely not viable for scalable key-value based cloud storage systems: Although it is possible to encode very short DHRs in the key used for data access [HGKW13], this requires knowledge

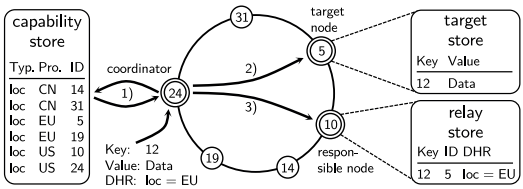

Figure 4.18 When creating data, the coordinator derives nodes that comply with the DHRs from the capability store. It then forwards the data to the target node and stores a reference to the data at the responsible node.

about DHRs of a data item to compute the key for accessing it and disturbs load balancing. Alternatively, replication of all relay information on all nodes of a cluster allows nodes to derive relay information locally. This, however, severely impacts the scalability of the cloud storage system and reduces the total storage amount to the limited storage space of single nodes.

Integrating PRADA into a cloud storage system requires us to adapt storage operations (e.g., creating and updating data) and to reconsider replication, load balancing, and failure recovery strategies in the presence of DHRs. In the following, we describe how we address these tasks.

4.3.2.2 Cloud Storage Operations

The most important modifications and considerations of PRADA involve the create, read, update, and delete (CRUD) operations of cloud storage systems. In the following, we describe how we integrate PRADA into the CRUD operations of our cloud storage model (as introduced in Section 4.3.1.1). To this end, we assume that queries to the storage systems are processed on behalf of the user by one of the cloud nodes in the cluster, which is the prevalent deployment model for cloud storage [LM10]. We refer to the cloud node that processes a query as the query's *coordinator* in the following. Each node of the cluster can act as coordinator for a query and a client application, e.g., a cloud service, will typically select one randomly. To ease presentation, we postpone the discussion of the impact of different replication factors and load balancing decisions to Section 4.3.2.3 and Section 4.3.2.4, respectively.

Creating Data. The coordinator for a query first checks whether a create request is accompanied by DHRs. If no DHRs are specified, the coordinator uses the standard method of the cloud storage system to create data such that the performance of native create requests is not impaired. For all data *with* DHRs, a create request proceeds in three steps as illustrated in Figure 4.18.

In Step 1, the coordinator derives the set of eligible nodes from the received DHRs, relying on the capability store (as introduced in Section 4.3.2.1) to identify nodes that fulfill all requested DHRs. As introduced in our design of CPPL as a privacy policy language in the context of cloud computing, users can combine different types of DHRs, e.g., location and support for deletion (cf. Section 4.2.2). Cloud nodes are

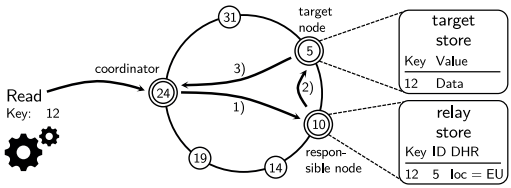

Figure 4.19 When reading data, the coordinator contacts the responsible node to fetch the data. As the data was created with DHRs, the responsible node forwards the query to the target, which directly sends the response to the coordinator.

eligible to store a piece of data if they support at least one of the specified properties for each requested type (e.g., one out of multiple permissible locations). When the coordinator derived the set of nodes in the storage cluster that can comply with all requirements specified by the user, it has to pick the target node that should store the data out of this set of eligible nodes. For this selection, it is important to choose the target such that the overall storage load in the cluster remains balanced (we defer a detailed discussion of this issue to Section 4.3.2.4).

In Step 2, the coordinator forwards the data to the target node, which stores the data in its target store.

Finally, in Step 3, the coordinator instructs the responsible node to store a reference to the actual storage location of the data to enable locating data upon subsequent read, update, and delete requests. The coordinator acknowledges the successful insertion to the client application after all three steps have been completed successfully. To speed up create operations, the second and third step—although logically separated—are performed in parallel. We defer a discussion on the recovery from failures during the creation of data to Section 4.3.2.5.

Reading Data. Processing read requests in PRADA is again performed in three steps as illustrated in Figure 4.19. In Step 1, the coordinator for the read query uses the key supplied in the request to initiate a standard read query at the responsible node for this key. If the responsible node does not store the data itself, it checks its local relay store for a reference to the target node for this data. Should it hold such a reference, the responsible node forwards the read request to the target node that is listed in the relay store in Step 2. To allow the target node to directly send the response back to the coordinator for this request, the forwarded request includes information on how to reach the coordinator node. In Step 3, the target node looks up the requested data in its local target store and directly returns the query result to the coordinator node for this query. Upon receiving the result from the target node, the coordinator processes the result in the same way as any other query result. If the responsible node stores the requested data locally (because it was stored without DHRs or the responsible node can comply with the stated DHRs), it directly answers the request using the default method of the cloud storage system. In contrast, if the responsible node neither stores the data directly nor a reference to it in the relay store, PRADA will correctly report that no data was found using the standard mechanism of the cloud storage system.

Updating Data. An update request for already stored data involves the (potentially partial) update of stored data as well as the possible update of associated DHRs. In the scope of PRADA, we assume that DHRs supplied with the update request supersede DHRs supplied with the initial create request and any potentially earlier updates. Other semantics for handling DHRs supplied with the update request, e.g., combining old and new DHRs, can be realized by slightly adapting the update procedure of PRADA. We process update requests the same way as create requests (as it is often done in cloud storage systems). Whenever an update request results in the necessity to change the target node of already stored data (due to changes in attached DHRs), the responsible node has to update its relay store. Furthermore, the update request needs to be applied to the stored data (currently located at the old target node). To this end, the responsible node instructs the old target node to move the data to the new target node. Upon reception of the data, the new target node applies the update to the data, locally stores the result, and acknowledges the successful update to the coordinator and the responsible node. The responsible node then updates its relay information. As updates for data without DHRs are directly sent from the coordinator to the responsible node, we do not impair the performance of native requests compared to an unmodified cloud storage system.

Deleting Data. In PRADA, delete requests are processed analogously to read requests. To this end, the coordinator sends the delete request to the responsible node for the key that should be deleted. If the responsible node itself stores the data, it deletes the data right away as in an unmodified cloud storage system. In contrast, if it only stores a reference to the data, it deletes the reference in its local relay store and forwards the delete request to the target node. The target node then deletes the stored data and informs the coordinator of the delete request about the successful termination of the query. We defer a discussion of recovering from failures during this process to Section 4.3.2.5.

4.3.2.3 Replication

Cloud storage systems employ replication of data to realize high availability and data durability [LM10]: Instead of storing a data item only on one cloud node, it is stored on r nodes (typically, with a replication factor $2 \leq r \leq 3$). In key-value based storage systems, the r nodes are chosen based on the key that identifies the data (cf. Section 4.3.2.1). When accounting for compliance with DHRs specified by users, we cannot use the same replication strategy as the nodes selected by the key generally do not support the stated DHRs. In the following, we thus detail how PRADA instead realizes replication.

Creating Data. Instead of selecting only one target node, the coordinator of the create query selects r target nodes out of the set of eligible nodes. The coordinator then sends the data to all r target nodes. Furthermore, the coordinator sends the list of all r target nodes to the r responsible nodes according to the unmodified replication strategy of the underlying cloud storage system. Consequently, each of the r responsible nodes knows about all r target nodes and hence can populate its relay store accordingly.

Reading Data. To process a read request, the coordinator of a query forwards the read request to all responsible nodes. Each responsible node that receives a read request for data it does not store locally looks up the target nodes for this data in its local relay store and forwards the read request to all r target nodes. Likewise, each target node that receives a read request sends the requested data back to the coordinator for this request. However, in contrast to the standard behavior, a target node may receive multiple forwarded read requests (from different responsible nodes). In this case, the target node processes only the first request and ignores any subsequent duplicate requests. To enable target nodes to detect duplicate requests, each request contains a unique identifier.

Impact on Reliability. To successfully process a query in PRADA, it suffices if one responsible node and one target node for the requested data are reachable. Thus, PRADA can tolerate the failure of up to $r - 1$ responsible nodes and up to $r - 1$ target nodes for each piece of data. We further discuss the impact of node failures in Section 4.3.2.5.

4.3.2.4 Load Balancing

In cloud storage systems, load balancing aims to minimize (long-term) load disparities in the storage cluster by distributing stored data and read requests equally among the cloud nodes. Since PRADA changes how data is assigned to and retrieved from nodes, existing load balancing schemes must be rethought. In the following, we first describe a formal metric to measure load balance and then explain how PRADA ensures a load-balanced cloud storage system.

Load Balance Metric. Intuitively, a good load balancing aims at all nodes in a cloud storage system being (nearly) equally loaded, i.e., the imbalance between the load of nodes should be minimized. This is important, since underloaded nodes constitute a waste of resources, while overloaded nodes drastically decrease the overall performance of the cloud storage system. We measure the load balance of a cloud storage system by normalizing the global standard deviation of the load of individual nodes with the mean load μ of all nodes [CLZ99]:

$$\mathfrak{L} := \frac{1}{\mu}\sqrt{\frac{\sum_{i=1}^{|N|}(\mathfrak{L}_i - \mu)^2}{|N|}}$$

with $\mathfrak{L}_i$ being the load of node $i \in N$. To achieve a reasonably balanced load across the cloud storage system, we strive to minimize $\mathfrak{L}$. By employing this metric, we especially penalize outliers, i.e., nodes with extremely low or high loads, which follows our intuition of a good load balance.

Load Balancing in PRADA. Traditional key-value based cloud storage systems achieve a reasonably balanced load across the different nodes in the cluster in two steps: (i) Equal distribution of data at insertion time, e.g., by applying a hash function to identifier keys, and (ii) re-balancing the cluster if absolutely necessary, e.g., if huge load imbalances are detected, by moving data between nodes. More advanced cloud storage systems support additional mechanisms, e.g., load balancing

over geographical regions [CDE+13]. Since our focus lies on proving the general feasibility of supporting compliance with DHRs in cloud storage systems, we focus on the properties of key-value based storage for our discussion of load balancing strategies in the scope of this work.

Re-balancing a cluster by moving data between nodes can be handled by PRADA similarly to moving data in case of node failures as we discuss in Section 4.3.2.5. In the following, we thus focus on the challenge of load balancing in PRADA at insertion time, i.e., equally distributing data with DHRs across target nodes. Load balancing of indirection information and data without DHRs is already achieved by the standard mechanisms of key-value based cloud storage systems, e.g., by hashing identifier keys.

In contrast to key-value based cloud storage systems, load balancing for data with DHRs in PRADA is more challenging: When processing a create request, the eligible target nodes are not necessarily equal as they might be able to comply with different DHRs. For example, some eligible target nodes might offer rarely supported but often requested requirements. However, foreseeing future demands is notoriously difficult [RA14]. Thus, we suggest to make the load balancing decision based on the past demand as reflected by the current load of cloud nodes. To this end, the coordinator of a query (which selects the target nodes when processing a create request) needs to be aware of the current load of all other nodes in the cloud cluster. Cloud storage systems typically already exchange this information or can easily be extended to do so, e.g., using efficient gossiping protocols [RDGT08]. We hence utilize this load information in PRADA as follows. To select the target nodes from the set of eligible nodes, the coordinator first checks if any of the responsible nodes are also eligible to become a target node and selects those as target nodes first. This allows us to increase the performance of CRUD requests as we can avoid the indirection layer in this case. For the remaining target nodes, the coordinator selects those with the lowest current storage load.

However, the load information provided by the underlying cloud storage system typically has a certain delay, resulting, e.g., from the employed gossiping scheme [RDGT08]. To cope with this issue and thus have access to more timely load information, each node in PRADA locally keeps track of all create, update, and delete requests it is involved with. Whenever a node itself stores new data or sends data for storage to other nodes, it increments temporary load information for the respective node. Similarly, the node decrements temporary load information when handling delete requests. This temporary and partial load information is used to bridge the time between two updates of the load information, e.g., by the underlying gossiping protocol. As we see in Section 4.3.3.3, this approach enables PRADA to adapt to different usage and load scenarios to quickly achieve a (nearly) optimally (under the constraints posed by users' DHRs) balanced cloud storage cluster.

4.3.2.5 Failure Recovery

When introducing support for DHRs to cloud storage systems, we must ensure not to break the underlying failure recovery mechanisms that, e.g., allow cloud storage

systems to cope with failures of individual cloud nodes resulting from issues such as hardware, software, and network defects. With PRADA, we specifically need to take care of dangling references, i.e., a reference pointing to a target node that does not store the corresponding data (anymore), and unreferenced data, i.e., data stored on a target node without a functioning reference at the corresponding responsible node. These inconsistencies could stem from failures during the (modified) CRUD operations as well as from actions explicitly triggered by DHRs. For example, deletions requested by DHRs require the subsequent deletion of indirection information at the corresponding responsible nodes. In the following, we discuss how PRADA handles failures during these operations in more detail.

Creating Data. For create requests, the coordinator has to transmit data to the target node and inform the responsible node to store the reference. The coordinator can detect errors that occur during these operations by missing acknowledgments. Resolving these errors requires the coordinator to perform a rollback and/or reissue actions, e.g., selecting a new target node and updating the reference at the responsible node. Still, also the coordinator itself can fail during the process of creating data, which potentially can lead to unreachable data. As such failures happen comparably rarely, we suggest refraining from including corresponding consistency checks directly into the processing of create operations [NG15]. Instead, we detect failures of the coordinator directly at the client application, e.g., a cloud service, through missing acknowledgments. In this case, the client application informs all potential target nodes to remove the potentially unreferenced data and subsequently reissues the create operation at another coordinator.

Reading Data. In contrast to all other operations, read requests do not change any state in the cloud storage system. Hence, in case of detected failures during read requests (identified by missing acknowledgments), these requests can simply be reissued and no further error handling is required.

Updating Data. Although update operations are slightly more complex than create operations (cf. Section 4.3.2.2), we can perform failure handling and recovery analogously. As the responsible node updates its reference only upon reception of the acknowledgment from the new target node, the storage state is guaranteed to remain consistent. Hence, the coordinator can simply reissue the update request using the same or a new target node and perform corresponding cleanups if errors occur. Contrary, if the coordinator fails, information on the potentially new target node is lost. Similar to create operations, the client application can resolve this situation by informing all potential target nodes about the failure. Subsequently, the responsible nodes trigger a cleanup to ensure a consistent storage state.

Deleting Data. When deleting data, a responsible node may have already deleted a reference when the communication with the target node to delete the actual data fails. Both coordinator and client application can easily detect this error through the absence of the corresponding acknowledgment. Again, either coordinator or client application can then issue a message to all potential target nodes to delete the corresponding piece of data. We consider this approach to be more reasonable than directly incorporating consistency checks for all delete operations as such failures typically occur only rarely [NG15].

Propagating Target Node Actions. The above CRUD operations are triggered by users or client applications, e.g., cloud services. However, deletion or relocation of data, which may result in dangling references or unreferenced data, can also be triggered by the cloud storage systems itself or by DHRs that, e.g., specify a maximum lifetime for data. To keep the state of the cloud storage system consistent, target nodes perform data deletion and relocation through a coordinator as well, i.e., they randomly select one of the other nodes in the cloud storage system to perform the update and delete operations on their behalf. Thus, the correct execution of deletion and relocation requests can be monitored and potential failures addressed using the above mechanisms for CRUD operations.

4.3.3 Evaluation

For the practical evaluation of our approach, we fully implemented PRADA on top of the widely-deployed distributed database Cassandra [LM10]. Based on our implementation of PRADA, we perform benchmarks to quantify query completion times, storage overhead, and traffic consumption as well as show PRADA's applicability in two real-world use cases. Furthermore, we study PRADA's load behavior based on simulation. Our evaluation shows that PRADA meets our set goals of minimal performance impact, cluster balance, and coexistence (cf. Section 4.3.1.3).

4.3.3.1 Implementation

Our implementation of PRADA is based on Cassandra 2.0.5, but conceptually also works with newer versions. Cassandra is a distributed database that is actively used as a key-value based cloud storage system by more than 1500 companies with deployments of up to 75 000 nodes [Apa18a] and offers high scalability even over multiple data centers [RGS+12], which makes it especially suitable for our scenario.

Cassandra also implements advanced features that go beyond simple key-value storage such as column-orientation and queries over ranges of keys, which allows us to showcase the flexibility and adaptability of our design.

Background on Cassandra

Cassandra realizes a combination of a structured key-value store and the column-oriented paradigm [Cat11]. To this end, data in Cassandra is divided into multiple logically separated databases, called *keyspaces*. A keyspace consists of tables which are called *column families* and contain rows and columns. Each row has a unique key and consists of several columns. Notably, and in contrast to traditional column-oriented databases, rows of the same table do not need to have the same set of columns and columns can be added to one or more rows anytime [Dat17b]. To partition rows based on their key, Cassandra uses a distributed hash table with `murmur3` as the hash function. In contrast to distributed hash tables in peer-to-peer systems [WGR05], each node in the cluster knows about all other nodes and

the ranges of the hash table they are responsible for. Cassandra uses the gossiping protocol Scuttlebutt [RDGT08] to efficiently distribute this knowledge as well as to detect node failures and exchange node state, e.g., the load of individual nodes.

Information Stores

Our design of PRADA relies on three information stores: the global capability store as well as relay and target stores (cf. Section 4.3.2.1). We implement these as individual keyspaces in Cassandra as detailed in the following. First, we realize the *global capability store* as a globally replicated key space initialized at the same time as the cluster. Within this key space, we create a column family for each DHR type (as introduced in Section 4.3.1.2). When a node joins the cluster, it inserts those DHR properties it supports for each DHR type into the corresponding column family. This information is then automatically replicated to all other nodes in the cluster using Cassandra's default mechanism for replicating data within the cluster.

For each regular key space of the database, we additionally create a corresponding *relay store* and *target store* as key spaces. Here, the *relay store* inherits the replication factor and replication strategy from the regular key space to achieve replication for PRADA (cf. Section 4.3.2.3), i.e., the relay store will be replicated in exactly the same way as the regular key store. Hence, for each column family in the corresponding keyspace, we create a column family in the relay keyspace that acts as the relay store. We follow a similar approach for realizing the *target store*, i.e., for each key space we create a corresponding key space to store actual data. However, to ensure that DHRs are adhered to, we implement a DHR-aware replication mechanism to ensure adherence to DHRs. For each column family in the corresponding keyspace, we create an exact copy in the target keyspace to act as the target store.

While the global capability store is created when the cluster is initiated, relay and target stores have to be created whenever a new keyspace or column family is created, respectively. To this end, we hook into Cassandra's `CreateKeyspaceStatement` class for detecting requests for creating keyspaces and column families and subsequently initialize the corresponding relay and target stores.

Creating Data and Load Balancing

To allow clients to specify their DHRs when inserting or updating data, we support the specification of arbitrary DHRs in textual form for `INSERT` requests. To this end, we add an optional postfix `WITH REQUIREMENTS` to `INSERT` statements by extending the grammar from which parser and lexer for CQL3 [Apa18c], the SQL-like query language of Cassandra, are generated using ANTLR [PQ95]. Using the `WITH REQUIREMENTS` statement, arbitrary DHRs can be specified separated by the keyword `AND`, e.g., `INSERT ... WITH REQUIREMENTS location = { 'DE', 'FR', 'UK' } AND encryption = { 'AES-256' }`. In this example, any node located in Germany, France, or the United Kingdom that supports AES-256 encryption is eligible to store the inserted data. This approach enables users to specify any DHRs covered by our formalized model of DHRs (cf. Section 4.3.1.2).

To detect and process DHRs in create requests (cf. Section 4.3.2.2), we extend Cassandra's `QueryProcessor` class, specifically its `getStatement` method for processing `INSERT` requests. When processing requests with DHRs (specified using the `WITH REQUIREMENTS` statement), we base our selection of eligible nodes on the global capability store. Nodes are eligible to store data with a given set of DHRs if they provide at least one of the specified properties for each requested type (e.g., one out of multiple permitted locations). We prioritize nodes that Cassandra would pick without DHRs, as this speeds up reads for the corresponding key later on, and otherwise choose nodes according to our load balancing strategy (cf. Section 4.3.2.4).

The implementation of our load balancing strategy relies on Cassandra's gossiping mechanism [LM10], which maintains a map of all nodes of a cluster and their loads. We access this information using Cassandra's `getLoadInfo` method and extend the load information with local estimators for load changes. Whenever a node stores data or sends a create request, we update the local estimator with the data size. To this end, we hook into the methods that are called when data is modified locally or forwarded to other nodes, i.e., the corresponding methods in Cassandra's `ModificationStatement`, `RowMutationVerbHandler`, and `StorageProxy` classes as well as our methods for processing requests with DHRs.

Reading Data

To allow reading redirected data as described in Section 4.3.2.2, we modify Cassandra's `ReadVerbHandler` class for processing read requests at the responsible node. This handler is called whenever a node receives a read request from the coordinator and hence enables us to check whether the current node holds a reference to another node for the requested key by locally checking the corresponding column family of the relay store. If no reference exists, the node continues with a standard read operation for local data. Otherwise, the node forwards a modified read request to each target node using Cassandra's `sendOneWay` method, in which it directly requests the data from the respective target stores on behalf of the coordinator. Subsequently, the target nodes send the data directly to the coordinator node (as identified in the request). To correctly resolve references to data for which the coordinator of a query is also the responsible node, we additionally add corresponding checks to the `LocalReadRunnable` subclass of the `StorageProxy` class.

4.3.3.2 Benchmarks

We first benchmark PRADA's query completion time, consumed storage space, and bandwidth consumption. In all settings, we compare the performance of PRADA with the performance of an unmodified Cassandra installation as well as a system running PRADA but receiving only data without attached DHRs, denoted by PRADA*. This approach enables us to evaluate whether data without attached DHRs is impaired by PRADA or not.

We set up a cluster of 10 identical nodes (Intel Core 2 Q9400, 4 GB RAM, 160 GB HDD, Ubuntu 14.04) interconnected via a gigabit Ethernet switch. Additionally, we

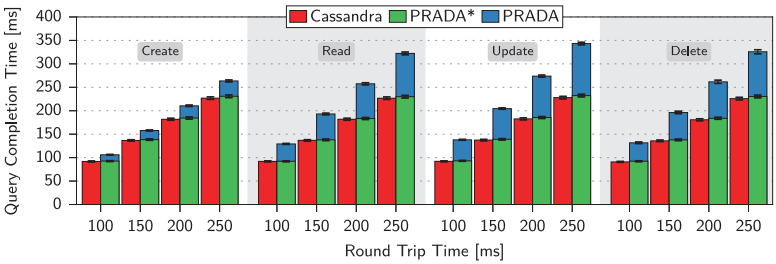

Figure 4.20 When studying query completion times for different RTTs, we observe that PRADA introduces limited overhead for operations on data with DHRs, while data without DHRs is not impacted by PRADA at all.

use one node with the same configuration to interface with the cloud storage system to perform CRUD operations. We assign each node a distinct DHR property. When inserting or updating data, clients request a set of exactly three of the available properties uniformly at random. Each row of data consists of 200 byte (+ 20 byte for the key), spread over 10 columns. These are rather conservative numbers as the relative overhead of PRADA decreases with increasing storage size. For each result, we performed 5 runs with 1000 operations each and depict the mean value for performing one operation with 99 % confidence intervals.

Query Completion Time

The query completion time (QCT) denotes the time the coordinator takes for processing a query, i.e., from receiving it until sending the result back to the client. It is influenced by the round-trip time (RTT) between nodes in the cluster and the replication factor applied to data.

We first study the influence of different RTTs on the QCT for a replication factor $r = 1$. To this end, we artificially add latency to outgoing packets for inter-cluster communication using `netem` [Hem05] to emulate RTTs ranging from 100 to 250 ms in steps of 50 ms. Our choice covers RTTs actually observed in communication between cloud data centers around the world [SMS11] which we independently verified through measurements in the Microsoft Azure cloud. In Figure 4.20, we depict the resulting QCTs for the different CRUD operations and increasing RTTs.

We make two observations: First, QCTs of PRADA* are identical to those of the unmodified Cassandra. Hence, data without DHRs is not impaired by PRADA. Second, the additional overhead of PRADA lies between 15.4 to 16.2 % for create, 40.5 to 42.1 % for read, 48.9 to 50.5 % for update, and 44.3 to 44.8 % for delete. The overheads for read, update, and delete correspond to the additional 0.5 RTT of the indirection layer and is slightly worse for updates as data stored at potentially old target nodes additionally needs to be deleted. This increase in QCTs constitute the costs users have to accept in turn for having support for DHRs in cloud storage

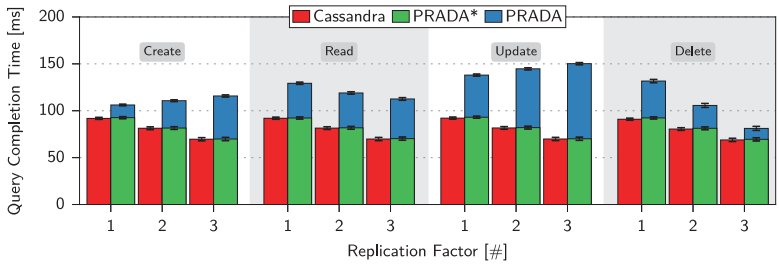

Figure 4.21 When studying the impact of an increasing replication factor on QCTs, create and update in PRADA show modest overhead for increasing replicas due to larger messages.

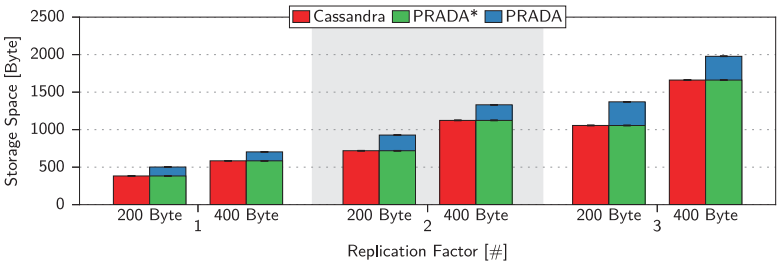

Figure 4.22 PRADA introduces only constant storage overhead per DHR affected replica, while not affecting data without DHRs.

systems. QCTs below the RTT result from corner cases where the coordinator is also responsible for storing data.

We now fix RTTs to 100 ms and study the impact of replication factors $r = 1, 2$, and 3 on QCTs as shown in Figure 4.21. Again, we observe that the QCTs of PRADA* and Cassandra are identical. Consequently, we conclude that data without DHRs is not impacted by PRADA. For increasing replication factors, the QCTs for PRADA* and Cassandra reduce as it becomes more likely that the coordinator also stores the data. In this case, Cassandra optimizes queries.

When considering the overhead of PRADA, we witness that the QCTs for creates (overhead increasing from 14 to 46 ms) and updates (overhead increasing from 46 to 80 ms) cannot benefit from these optimizations, as this would require the coordinator to be responsible and target node at the same time, which happens only rarely. Furthermore, the increase in QCTs for creates and updates results from the overhead of handling r references at r nodes. For reads, PRADA shows an average overhead of 37 to 43 ms due to the additional 0.5 RTT for the indirection layer. For deletes, the overhead decreases from 41 to 12 ms for an increasing replication factor, which results from an increased likelihood that the coordinator node is at least either responsible or target node, which avoids the need for additional communication.

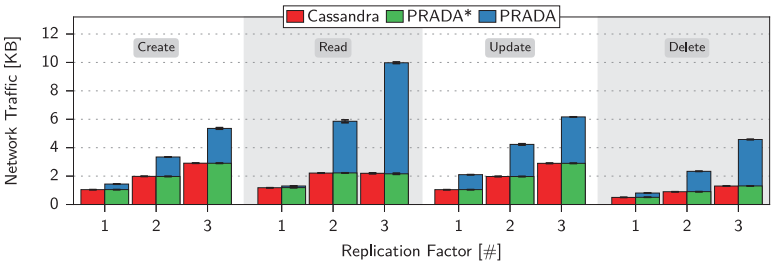

Figure 4.23 When considering the resulting network traffic, we observe that data without DHRs is not affected by PRADA. Furthermore, replicas linearly increase the traffic overhead introduced by DHRs.

Consumed Storage Space

To quantify the additional storage space introduced by PRADA, we measure the consumed storage space after data has been inserted, using the `cfstats` option of Cassandra's `nodetool` utility. To this end, we conduct insertions for payload sizes of 200 and 400 byte (plus 20 byte for the key), i.e., we fill 10 columns with 20 respectively 40 byte payload in each query, with replication factors of $r = 1, 2$, and 3. We divide the total consumed storage space per run by the number of insertions and show the mean consumed storage space per inserted row over all runs in Figure 4.22. Each additional replica increases the required storage space by roughly 90 % for Cassandra. PRADA adds an additional constant overhead of roughly 115 byte per replica. While the precise overhead of PRADA depends on the encoding of DHRs and relay information, the important observation here is that it does not depend on the size of the stored data.

If deemed necessary, the required storage space can be further reduced, e.g., by integrating PRADA with CPPL, our storage space-efficient privacy policy language (cf. Section 4.2). As we show in our evaluation of CPPL's performance (cf. Section 4.2.3), the processing overhead for employing policy compression with CPPL lies well below 1 ms, hence leading only to a marginal impact on the QCTs of PRADA.

Bandwidth Consumption

Furthermore, we measure the network traffic that results from the individual CRUD operations for Cassandra, PRADA*, and PRADA. Figure 4.23 depicts the mean total generated network traffic per individual operation. Our results show that using PRADA comes at the cost of an overhead that scales linearly in the replication factor. When considering Cassandra and PRADA*, we observe that the consumed traffic for read operations does not increase when raising the replication factor from 2 to 3. This results from an optimization in Cassandra that requests the data only from one replica and probabilistically compares only digests of the data held by the other replicas to perform post-request consistency checks. We did not include this optimization in PRADA and hence it is possible to further reduce the bandwidth

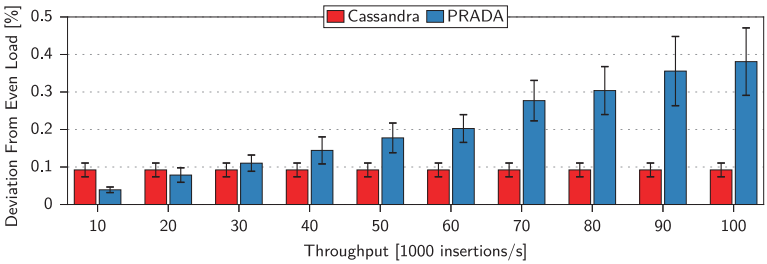

Figure 4.24 Load balance in PRADA depends on the throughput of insert operations. Even for high throughputs, the deviation from an evenly balanced load stays below 0.5 %.

consumed by PRADA by applying the same optimization. For the other operations, the overhead introduced by our indirection layer ranges from 2.4 to 3.3 KB for a replication factor of 3. For a replication factor of 1, the highest overhead introduced by PRADA peaks at 1.1 KB. Thus, we conclude that the traffic overhead of PRADA is manageable for practical operation in cloud storage systems.

4.3.3.3 Load Distribution

To quantify the impact of PRADA on the load distribution of the overall cloud storage system, we rely on simulation as this enables us to perform a large-scale analysis of the load distribution by considering a wide range of scenarios.

Simulation Setup

As we are solely interested in the load behavior, we implemented a custom simulator in Python, which models the characteristics of Cassandra with respect to network topology, data placement, and gossip behavior based on the concept of discrete-event simulation [WGG10]. Using this simulator, we realize a cluster of n nodes, which are equally distributed among the keyspace [Dat17b] and use this cluster to insert m data items with random keys. For reasons of simplicity, we assume that all data items are of the same size. The nodes operate Cassandra's gossip protocol [RDGT08], i.e., synchronize with one random node every second and update their own load information every 60 s. We randomize the initial offset before the first gossip message for each node individually, as, in reality, not all nodes perform the gossip at the same point in time. We repeat each measurement 10 times with different real random seeds [Wal96] and show the mean of the load balance metric $\mathfrak{L}$ (cf. Section 4.3.2.4) over these measurements with 99 % confidence intervals.

Influence of Throughput

We expect the load distribution to be influenced by the freshness of the load information as gossiped by other nodes, which strongly correlates with the throughput of

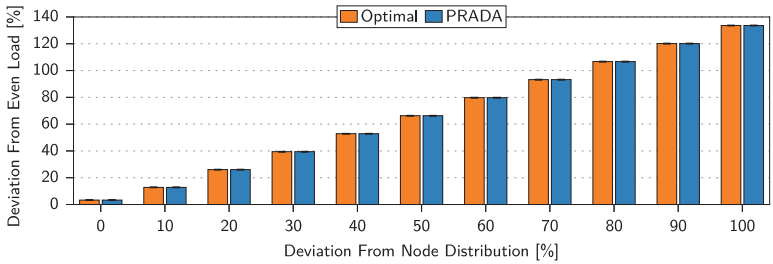

Figure 4.25 When studying the influence of the conformance of DHRs with node properties, we observe that PRADA's load balance shows close to optimal behavior. However, heavy deviation of DHRs from nodes' properties leads to non-even load.

create requests. A lower throughput results in less data being inserted between two load information updates and hence the load information remains relatively fresher compared to a scenario with a higher throughput of create operations. To study this effect, we perform an experiment where we simulate different insertion throughputs and hence vary the gossiping delay. We simulate a cluster with 10 nodes and 10^7 create requests, each accompanied by DHRs. Even for high throughputs, this produces enough data to guarantee at least one gossip round during each simulation run. To challenge the load balancer, we synthetically create two types of DHRs with two properties, each supported by half of the nodes such that each combination of the properties of the two types of DHRs is supported by two or three nodes. For each create request we randomly select one of the resulting possible DHRs, i.e., demanding one of the properties for one or two of the DHRs types.

Figure 4.24 shows the deviation from an even load for increasing throughput compared to the load distribution of a traditional Cassandra cluster. Additionally, we calculated the optimal solution under a posteriori knowledge by formulating the corresponding quadratic program for minimizing the load balance $\mathfrak{L}$ and solving it using CPLEX [IBM17]. In all cases we observe that the resulting optimum leads to a load balance of 0, i.e., all nodes are equally loaded, and hence omit these results in the plot. Seemingly large confidence intervals result from the high resolution of our plot (in all scenarios, PRADA deviates less than 0.5 % from even load). Our results show that PRADA surprisingly even outperforms Cassandra for very small throughputs (the load imbalance of Cassandra results from the randomness of the hash function) and the introduced load imbalance for the other scenarios stays below 0.5 %, even for a high throughput of 100 000 insertions/s. To put these numbers into perspective, Dropbox processed less than 20 000 insertions/s on average in June 2015 [Dro15]. In summary, our results indicate that frequent updates of node state result in a better load balance for PRADA. Still, even for less frequent updates, PRADA still achieves a load balance that is extremely close to the standard load balance realized by an unmodified Cassandra cluster.

Influence of DHR Fit

In PRADA, one of the core influence factors on the load distribution is the accordance of users' DHRs with the properties provided by cloud storage nodes. If the distribution of DHRs in create requests heavily deviates from the distribution of DHRs supported by the storage nodes, it is impossible to achieve an even load distribution. To study this aspect, we consider a scenario where each node has a storage location and users request exactly one of the available storage locations as their DHR. We simulate a cluster of 100 nodes that are geographically distributed according to the IP address ranges of Amazon Web Services [AWS17] (North America: 64 %, Europe: 17 %, Asia-Pacific: 16 %, South America: 2 %, China: 1 %). First, we insert data with DHRs such that the distribution of requested storage locations exactly matches the distribution of nodes. Subsequently, we worsen the accuracy of fit by subtracting 10 % to 100 % from the location with the most nodes (i.e., North America) and proportionally distribute this demand to the other locations (in the extreme setting, North America: 0 %, Europe: 47.61 %, Asia-Pacific: 44.73 %, South America: 5.74 %, and China: 1.91 %). For each of the resulting scenarios, we simulate 10^7 insertions at a throughput of 20 000 insertions/s.

To put our results into perspective, we calculate the optimal load using a posteriori knowledge by equally distributing the data on the nodes of each location. Our results are depicted in Figure 4.25. We derive two insights from this experiment: (i) the deviation from an even cluster load scales linearly with decreasing accordance of users' DHRs with the capabilities of cloud nodes in the storage cluster and (ii) in all considered settings, PRADA manages to achieve a cluster load that is extremely close to the theoretical optimum (increase < 0.03 % in all settings). Hence, we can conclude that PRADA's approach of load balancing perfectly adapts to the challenges imposed by complying with DHRs in cloud storage systems.

4.3.3.4 Applicability

We show the applicability of PRADA by using it to realize two real-world use cases: a microblogging system and a distributed email management system. To this end, we emulate a globally distributed cloud storage using our cluster of 10 nodes (cf. Section 4.3.3.2) by modeling a worldwide distribution of nodes based on measurements we performed in Microsoft's Azure Cloud. We emulate one node in each of the following regions provided by Microsoft Azure [Mic16b]: `asia-east`, `asia-southeast`, `canada-east`, `eu-north`, `eu-west`, `japan-east`, `us-central`, `us-east`, `us-southcentral`, and `us-west`. To this end, we use `netem` to add delay between the cluster nodes according to measurements of this topology we performed using `hping3` [San06] in Microsoft's Azure Cloud. The resulting RTTs between the nodes of our cluster range from 24.3 ms (`eu-north` → `eu-west`) to 286.2 ms (`asia-east` → `eu-west`). We provide the full results of our RTT measurements in Appendix A.2.

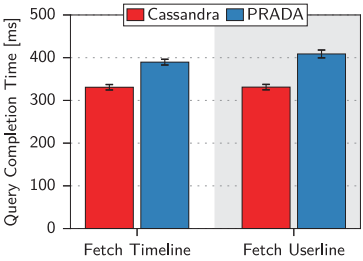

Figure 4.26 In our microblogging use case, adding DHRs to tweets delays query completion by only 18 % to 24 %.

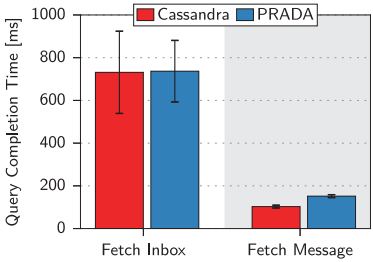

Figure 4.27 For the email storage use case, accounting for compliance with DHRs adds only little overhead to QCTs.

Microblogging

Microblogging services such as Twitter often utilize cloud storage systems to store messages. To evaluate the impact of PRADA on such services, we use the database layout of Twissandra [Twi15], an exemplary implementation of a microblogging service for Cassandra, and real tweets from the twitter7 dataset [YL11]. For each user, we uniformly at random select one of the storage locations and attach it as DHR to all tweets. We perform our measurements with a replication factor of $r = 1$ and measure the QCTs for randomly chosen users for retrieving their userline (most recent messages of this user) and their timeline (most recent messages of all users a user follows). To this end, we insert 2 million tweets from the twitter7 dataset [YL11] and randomly select 1000 users among those users who have at least 50 tweets in our dataset. For the userline measurement, we request 50 consecutive tweets of each selected user. As the twitter7 dataset lacks follower relationships, we request 50 random tweets across all users for the timeline measurements of each selected user.

Our results in Figure 4.26 show that the runtime overhead of supporting DHRs for microblogging in a globally distributed cluster corresponds to an 18 % increase in QCT for fetching the timeline and 24 % for retrieving the userline. Here, PRADA especially benefits from the fact that identifiers are spread along the cluster and thus the unmodified Cassandra also has to contact a large number of nodes. Our results show that PRADA can be applied to offer support for DHRs in microblogging at reasonable costs with respect to query completion time. Especially when considering that likely not each tweet will be accompanied by DHRs, this modest overhead is well worth the additional functionality that supports users in protecting their privacy.

Email Storage

As we have seen by the results uncovered by MailAnalyzer (cf. Section 3.2), email providers increasingly move storage of emails to the cloud. To study the impact of supporting DHRs on emails, we analyzed Cassandra-backed email systems such as Apache James [Apa18b] and ElasticInbox [Ela13] and derived a common database layout consisting of one table for metadata (for creating an overview of a complete

mailbox) and one table for full emails (for fetching individual emails). To create a realistic scenario, we utilize the Enron email dataset [KY04], consisting of about half a million emails of 150 users. For each user, we uniformly at random select one of the available storage locations as DHR for their emails and meta information.

Figure 4.27 compares the mean QCTs per operation of Cassandra and PRADA for fetching the *overview of the mailbox* for all 150 users and fetching 10 000 randomly selected *individual emails*. For fetching of mailboxes, we cannot derive a difference between Cassandra and PRADA as QCTs are dominated by the transfer of metadata (up to 28 465 rows). The large confidence interval result from the small number of operations (only 150 mailboxes) and huge differences in mailbox sizes, ranging from 35 to 28 465 messages for different users. When considering the fetching of individual messages, we observe an overhead of 47 % for PRADA's indirection, increasing query completion times from 103 ms to 152 ms. Hence, we can provide compliance with DHRs for email storage with an increase of 47 % for fetching individual emails, while not increasing the time for generating an overview of a mailbox.

4.3.4 Summary and Future Work

Accounting for compliance with data handling requirements, i.e., offering control over where and how data is stored in the cloud, has become increasingly important due to legislative, organizational, or customer demands. To address this issue, we have proposed PRADA, which empowers users to specify a comprehensive set of fine-grained DHRs and enables cloud storage operators to enforce them. Our results show that we can indeed achieve support for DHRs in cloud storage systems. Of course, the additional protection and flexibility offered by DHRs come at a price: We observe a moderate increase for query completion times and still manageable bandwidth demands while achieving constant storage overhead and upholding a near optimal storage load balance. Notably, data without DHRs is not impaired by PRADA. Hence, users can choose (even at a granularity of individual pieces of data), if DHRs are worth a modest performance decrease.

PRADA's design is built upon a transparent indirection layer, which effectively handles compliance with DHRs and hence realizes the goals that any approach to realize compliance with DHRs in cloud storage systems have to fulfill (cf. Section 4.3.1.3). First, PRADA realizes *comprehensiveness* by supporting any DHRs that can be expressed using our formalized notion of DHRs. Notably, this enables PRADA to also support future and as of now unforeseen requirements. With respect to our goal of *minimal performance impact*, PRADA's indirection introduces an overhead of 0.5 RTTs for read, update, and delete operations. Further reducing this overhead is likely only possible by encoding some DHRs in the key used for accessing data, but this requires everyone accessing the data later on to be in possession of the DHRs, which we consider an unrealistic assumption. A fundamental improvement could be achieved by replicating all relay information to all nodes in the cluster, but this is viable only for small cloud storage systems and does not scale.

We argue that indirection can likely not be avoided, but still pose this as an open research question. Considering *cluster balance*, the overall achievable load balance

highly depends on how well nodes' capabilities to fulfill certain DHRs match the actual DHRs requested by users. However, for a given scenario, PRADA is able to achieve nearly optimal load balance (cf. Section 4.3.3.3). Finally, PRADA realizes *coexistence* by not modifying the processing of data that is inserted without attached DHRs. As seen in Section 4.3.3.2, this indeed ensures that data without attached DHRs is not impaired by PRADA as evidenced by identical results with respect to query completion time, required storage space, and consumed bandwidth.

With respect to future work, we mainly identify two different promising directions. First, PRADA's initial design is centered around key-value based storage systems and we consider it promising to extend our approach to other storage systems that are based on different paradigms. For example, Google's globally distributed database Spanner (rather a multi-version database than a key-value store) allows applications to influence data locality (to increase performance) by carefully choosing keys [CDE+13]. PRADA could be applied to Spanner by modifying Spanner's approach of directory-bucketed key-value mappings. Likewise, PRADA could realize data compliance for distributed main memory databases, e.g., VoltDB, where tables of data are partitioned horizontally into shards [SW13]. Here, the decision on how to distribute shards over the nodes in the cluster could be taken with DHRs in mind. Similar adaptations could be performed for commercial products, such as Clustrix [Clu18], that separate data into slices. From a different perspective, our work on realizing PRADA intentionally focuses on realizing the *functionality* to support compliance with DHRs within cloud storage systems. Orthogonally to this approach stands the question on how users can be provided with *assurance* that a cloud provider indeed enforces their DHRs. On a general level, this question has been widely studied [ABF+04, PGB11, MNP+11, OSGJ13, VEM+15]. However, further work—both on the conceptual and technical level—is required to actually apply the proposed approaches such as audit logging, information flow control, and provable data possession (cf. Section 4.3.1.4) to our design of PRADA.

To conclude, PRADA resolves a situation, i.e., missing support for DHRs, that is disadvantageous to both users *and* providers of cloud storage systems. By offering the enforcement of arbitrary DHRs when storing data in cloud storage systems, PRADA enables the use of cloud storage systems for a wide range of clients who previously had to refrain from outsourcing storage, e.g., due to compliance with applicable data protection legislation. At the same time, we empower cloud storage operators with a practical and efficient solution to handle differences in regulations and offer their services to new clients. Hence, PRADA provides a valuable foundation for our overarching goal of realizing DHRs-aware cloud infrastructure.

4.4 Conclusion

Given the opaque legislation and the lack of control in today's cloud computing infrastructure, we laid out our vision of a data handling requirements-aware cloud stack. To this end, we proposed to annotate data with DHRs before it is sent to the cloud. This empowers users to express their privacy requirements with respect

to the handling of their data in the cloud and at the same time enables cloud providers to incorporate users' requirements while handling their data. To realize this goal, we identified the need to realize two fundamental underlying approaches: (i) a mechanism for users to express their DHRs and hence annotate their data accordingly before it is sent to the cloud and (ii) an approach for providers of cloud infrastructure—specifically cloud storage systems—to incorporate users' DHRs when mapping data to actual storage nodes.

To enable users to express their DHRs, we introduced CPPL, a compact privacy policy language specifically tailored to the characteristics of cloud computing. The core idea of CPPL is to compress a textual, human-readable specification of DHRs using flexibly specifiable domain knowledge into a size and processing efficient compressed representation that is optimized down to the bit-level. Notably and unlike related work, CPPL can directly work on the compressed representation of DHRs when interpreting policies at cloud nodes. Our evaluation not only showed that CPPL indeed achieves huge savings with respect to storage and transmission sizes (up to two orders of magnitude compared to related work) but is also able to process several thousands of compressed policies per second in real-world scenarios. Hence, CPPL constitutes a valuable foundation for our vision of a DHRs-aware cloud stack as it enables users to express their privacy requirements on a per-data item level and thus make cloud providers aware of their users' demands at a fine granularity.

Once users are able to express their DHRs using CPPL, cloud providers have the necessary information to incorporate user demands into their allocation of resources. To this end, PRADA realizes a cloud storage system that offers rich and practical support for users' DHRs by storing a specific data item only on those cloud nodes that fully comply with the attached DHRs, e.g., expressed using CPPL. To showcase the feasibility and applicability of PRADA, we implemented it on top of the widely-deployed distributed database Cassandra. Our evaluation of PRADA shows that the additional offered functionality results in a moderate increase in query completion times as well as a small constant storage overhead while keeping the storage load of the nodes that form the cloud storage system as balanced as possible under the constraints imposed by users. Notably, PRADA does not impair the performance of data that is inserted without attached DHRs. PRADA's ability to store data only on cloud nodes that fulfill users' DHRs hence is a practical foundation for providing the cloud storage infrastructure required in a fully DHRs-aware cloud stack.

To conclude, in this chapter, we addressed the research question on how *infrastructure providers* can support service providers and cloud users in executing control over privacy. Hence, our contributions presented in the chapter mainly tackle the core problems of opaque legislation and missing control, thereby paving the way to a less centralized deployment model by making cloud resources more interchangeable and integrating privacy requirements into the process of cloud brokerage. Our concepts and results presented in this chapter highlight the feasibility of realizing a fully DHRs-aware cloud stack that gives users control over their privacy by enabling cloud infrastructure providers to incorporate user demands while mapping data to their distributed infrastructure. Notably, the results derived in this chapter can serve as an important foundation to realize privacy-preserving cloud services, e.g., in the context of the IoT, as presented in the next chapter. Furthermore, the

concepts underlying both CPPL and PRADA can also be applied to a fully decentralized approach to cloud computing where cloud services are deployed in a secure peer-to-peer manner as we introduce in Chapter 6.

5

Privacy-preserving Cloud Services for the Internet of Things

In this chapter, we target the research question on how service providers can realize privacy-preserving cloud services on top of cloud infrastructure without influence on the underlying resources. To this end, we use the Internet of Things (IoT) as application domain for cloud services with high privacy requirements [ZGW14].

We first motivate the need for privacy-preserving cloud services for the IoT [HHK+14, HHMW14, HHMW16] and deduce the individual components that are required to enable developers of cloud services to account for privacy (Section 5.1). Based on a security architecture for IoT data in the cloud [Mat13, See13, HHM+13, HHMW14], we present SCSlib [Ber14, HBHW14], a security library that enables non-security experts to develop privacy-preserving cloud services that operate on encrypted IoT data in a cryptographically enforced access control system (Section 5.2).

Subsequently, we introduce D-CAM [Wol14, HWM+17], a distributed approach to configuration, authorization, and management of devices and networks in the cloud-based IoT that puts users back in control over their cloud-managed IoT devices as well as networks (Section 5.3). Finally, we wrap up this chapter with a brief summary and discussion (Section 5.4).

5.1 Motivation

As already outlined in Section 2.4, we observe that the increasing deployment of IoT networks—ranging from home networks to industrial automation—leads to a similarly growing demand for storing and processing collected data. To satisfy this demand, the most promising approach is the utilization of the dynamically scalable, on demand resources made available by cloud infrastructure. More specifically,

cloud solutions simplify storage and processing of collected data, utilization of the same data within several services, as well as combining data from several users and supporting user mobility without information fragmentation over different systems.

However, while the integration of IoT networks with cloud computing environments is a striking proposition, the desired interconnection is far from trivial as sensed data often contains sensitive information that third parties may strive to exploit (cf. Section 2.4.2). For example, IoT readings from an industrial deployment can provide competitors with valuable information about the employed equipment and its degree of capacity utilization, thus providing them with a competitive advantage. Likewise, IoT readings from a pervasive healthcare system such as Internet-connected heart rate monitors embedded into smartwatches might prove valuable for health and life insurance companies to increase a person's fee or even deny a new contract [Boy17].

Moreover, sensitive information may not only be contained in the sensed data itself but could also be derived from the corresponding meta information, e.g., location information can be used to accurately track a user [PHW17]. As a result, owners of IoT deployments typically prefer to refrain from unconditionally revealing their sensed data to others, especially in the face of the numerous privacy threats and risks of cloud computing (cf. Section 2.3).

When outsourcing processing and storage of potentially sensitive IoT data to the cloud, we require a practically viable security architecture that enables users to stay in control over their data. To this end, any security architecture that targets this goal has to offer technical measures to (i) protect potentially sensitive IoT data already within the IoT network where it is still under control of the user, (ii) guarantee confidentiality and integrity of IoT data after it has left the IoT network, and (iii) offer user-centric access control of IoT data for trustworthy services. As a foundation for our contributions presented in this chapter, we rely on a trust point-based security architecture that fulfills these goals (we provide an overview of the important components of this security architecture in Section 5.2.2.2). We face two important challenges when applying this security architecture to realize privacy-preserving cloud services for the IoT.

First, the necessary security mechanisms are not transparent to cloud services and implementing them is a labor-intensive and error-prone task [SPP01]. However and especially in the context of the IoT, developers of cloud services are domain experts and typically do not specialize in security [Coo18]. Consequently, they should be relieved from having to realize the required security functionality on their own.

Second, such a cloud-based security architecture considerably increases the configuration, authorization, and management effort of users, especially if they operate multiple IoT networks. While it might seem natural to also offload these tasks to the cloud, we postulate that outsourcing the configuration, authorization, and management of IoT devices and networks to the cloud poses privacy and security threats. These threats are especially relevant since IoT devices often provide safety-critical functionality [Sta14]. Thus, we require additional security mechanisms that enable users to conveniently control their federated IoT networks and still offer protection against malicious entities that may strive to take over control of devices and thus harm privacy and safety.

5.1.1 Contributions

To address these two challenges and thus enable the realization of privacy-preserving cloud services for the IoT, we present the following two contributions.

1) We present *SCSlib*, a security library that transparently handles all security functionality required to access protected IoT data and thereby unburdens service developers from having to implement security functionality such as decryption and signature verification themselves. SCSlib relies on a widely-applicable, standards-based approach to represent and protect IoT data in the cloud. Notably, SCSlib does not require any security expertise from cloud service developers and, at the same time, is sufficiently flexible to satisfy a wide range of performance and security requirements. Our evaluation results obtained on public cloud infrastructure not only show the applicability of SCSlib but also demonstrate a meaningful performance gain for sequential and random access to IoT data in the cloud compared to naïvely implementing the required functionality.
2) With *D-CAM*, we introduce a distributed architecture that enables users to securely configure, authorize, and manage their IoT devices across network borders by using the cloud as a highly available and scalable storage for control messages. Thereby, D-CAM ensures that only authorized parties can configure a user's IoT devices. Most notably, even a malicious cloud provider cannot tamper with the configuration of IoT devices. To illustrate the feasibility of D-CAM, we fully implement a working prototype and extensively quantify the incurred processing and storage overheads. Our results show that D-CAM can easily scale to networks with hundreds of devices. To further increase the privacy of users in the cloud-based IoT, we additionally provide a mechanism for confidentiality of configuration, authorization, and management messages.

5.2 SCSlib: Transparently Accessing Protected IoT Data in the Cloud

We consider a scenario in which operators of IoT networks (i.e., private users, companies, or public institutions) connect their IoT networks to the cloud to benefit from its virtually infinite storage and processing resources [HHCW12, EHH+14]. There, cloud-hosted services selected by the operator of the IoT network operate on the outsourced IoT data. Similar to modern smartphones, these services can be provided by essentially anyone in a cloud service marketplace. Services either exclusively operate on the data from one IoT network or combine the data of several networks and users to realize functionality that would not be possible in an isolated setting, where the individual IoT networks are not interconnected with the cloud.

However, as IoT data often contains sensitive information, privacy concerns become a major challenge when interconnecting IoT networks with the cloud. Importantly, traditional transport security mechanisms between data sources and the cloud do not suffice to protect IoT data in an end-to-end manner as such channel security is typically terminated at the entry point to the cloud, leaving data unprotected

within the cloud. In contrast, *object security*, i.e., protection of individual data objects, between data sources and cloud services affords for the required end-to-end protection of outsourced IoT data.

Applying object security in the context of the cloud-based IoT comes with two inherent challenges. First, IoT data can originate from a wide variety of IoT nodes and thus can be arbitrarily structured, substantially complicating the application of object security mechanisms. Consequently, cloud services have to be informed how data has been protected to successfully decrypt and verify the integrity and the authenticity of the received data. Second, object security operates at the application level. Hence, contrary to transport security, object security is *not* a transparent security mechanism for cloud services. However, implementing the necessary security mechanisms is a laborious and error-prone task. Thus, developers of cloud services should not need to be responsible for realizing security functionality as they often are not experts in security [Coo18].

To address these challenges, we first show how recent progress in standardization can provide the basis for protecting data from different IoT devices when outsourcing data processing and storage to the cloud. These efforts serve as the foundation for the discussion of our trust point-based security architecture that realizes object security between IoT networks and cloud services based on fine-grained, user-centric data access control. Subsequently, we present our Sensor Cloud Security Library (SCSlib), which enables cloud service developers to transparently access cryptographically protected IoT data in the cloud. SCSlib specifically allows domain specialists who are not experts in security to realize privacy-preserving cloud services. To ease the reproducibility of our results and to provide a foundation for other research efforts, we provide the source code of SCSlib under the open source MIT license[9].

5.2.1 The Cloud-based IoT and Privacy

In the following, we concretize our general network scenario for the cloud-based IoT (cf. Section 2.4.1) and introduce the relevant entities involved as well as their interactions. Furthermore, we identify implications of the outlined scenario with respect to privacy as well as security and discuss related work.

5.2.1.1 Scenario and Entities

In our work, we consider a scenario where each IoT network (with an arbitrary number of IoT devices) is connected to the cloud via a dedicated gateway as depicted in Figure 5.1. Each *user* maintains a network consisting of IoT nodes that continuously produce IoT data. The user is in possession of any IoT data that is produced within her IoT network domain and outsources the storage and processing of her IoT data to a cloud computing environment. This cloud computing environment is operated by an *infrastructure provider* and we assume this environment to be public, i.e., the infrastructure provider offers its infrastructure to anyone who is willing to pay for

[9] `https://code.comsys.rwth-aachen.de/redmine/projects/scslib`

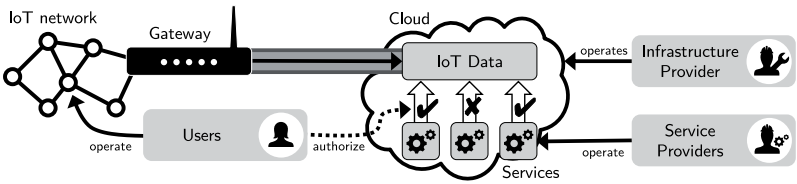

Figure 5.1 Users upload their IoT data to the cloud, which is realized on resources operated by the infrastructure provider. Service providers deploy their services on top of the cloud, which allows users to authorize selected services to access their IoT data.

it. The infrastructure provider is able to monitor any component of its infrastructure, e.g., for maintenance purposes. In addition to the mere infrastructure, the infrastructure provider offers a storage service for IoT data as well as execution environments for cloud-hosted third party services that process outsourced IoT data. The individual cloud services are offered by *service providers*.

As already discussed in Section 2.4, data collected by IoT devices often contains personal information and outsourcing it to the cloud thus raises privacy concerns. Consequently, it is imperative that the user remains in control over who has access to her data. To this end, IoT data stored in the cloud should only be accessible by authorized entities. More specifically, only those cloud services that have explicitly been authorized by the user to access (parts of) her IoT data should be able to retrieve and process said IoT data. Thus, we require a mechanism for users to *authorize* cloud services to access a specified set of their IoT data. Subsequently, any security architecture then has to enforce that indeed only authorized cloud services can gain access to the IoT data of a user.

5.2.1.2 Security and Privacy Considerations

As a foundation for protecting privacy when outsourcing potentially sensitive IoT data to the cloud, we first lay out our underlying security and privacy assumptions. Regarding the local IoT network, we assume that IoT data is adequately protected against local network-level attacks. For example, existing ZigBee security mechanisms [Zig12] could be employed for ZigBee-based IoT networks. For IP-based IoT networks, as currently advocated by standardization bodies [MHCK07, Zig13], traditional IP security solutions could be deployed by modifying the corresponding protocols with respect to the device and network constraints prevalent in the IoT [HHHW13, HHW+13b, HHS+18].

Once IoT data leaves the protected IoT network and is transported via the Internet, we also have to consider external attackers, who may try to manipulate or eavesdrop on the communication with the cloud computing environment. Thus, the confidentiality as well as the integrity of IoT data and of management-related communication between the IoT network and the cloud has to be protected. Furthermore, considering the multi-tenancy characteristics of cloud computing, the user requires her IoT data not to be revealed to an entity that she did not explicitly authorize. Specifically,

a user's IoT data must not be accessible by other users sharing the storage facilities of the cloud. Likewise, neither the infrastructure provider nor unauthorized services must be able to access stored IoT data. Any modification of stored IoT data must be perceivable by either the user or a service processing this data.

To this end, we assume that the infrastructure provider is following an adversary model similar to that of an honest-but-curious adversary (cf. Section 2.3.2). As such, it will operate technology, services, and interfaces as contractually agreed and will not perform active attacks to spy into running services. However, it might try to learn as much as possible about the processed information and it might not guarantee long-term confidentiality of stored information. For example, the infrastructure provider has full control over its hardware and, therefore, can inspect the memory of its physical machines for sensitive information as long as data processing is performed by services on plaintext IoT data. However, contractual obligations and liabilities can render such inspections unattractive for the infrastructure provider as shown by the US government's use of Amazon's AWS GovCloud offer [AWS18c].

Likewise, by strictly restricting access to its monitoring capabilities to a small number of trusted administrators who must be located within the data center, the infrastructure provider can mitigate the risk of exposure due to attacks against its monitoring facilities. The service providers, on the contrary, are generally considered less trustworthy. This is due to the fact that the user cannot control which services are offered by the cloud and those services may actively try to gain unconstrained access to IoT data that is not meant for disclosure to them. Outside entities must be considered malicious adversaries (cf. Section 2.3.2) that may perform arbitrary actions with to break into communication flows and hence gain access to confidential IoT data.

To back up these assumptions, we require the infrastructure provider to separate the execution of its infrastructure and the cloud services running on top of it from other third party cloud services. To this end, the infrastructure provider may, e.g., employ special virtual machine placement policies that do not map other cloud services to the same physical machines that are used to realize privacy-preserving services for the IoT. Likewise, we require that the infrastructure provider enforces strict separation of execution environments for individual services running on the same physical machine, e.g., using containers, to prevent cloud services from interfering with each other. Furthermore, to gain at least a certain level of trust in cloud services, the infrastructure provider or another trusted third party can perform audits of cloud services before they are made available to users. This process is similar to the approach taken by today's app stores on smartphones.

5.2.1.3 Related Work

To address the privacy and security considerations stated above and to mitigate the anticipated loss of control over IoT data once it is stored and processed in the cloud, early approaches in the area of combining the IoT with cloud computing—especially in the area of healthcare and ambient assisted living—identified that providing privacy and security guarantees is a crucial cornerstone. To this end, Rolim

et al. [RKW+10] propose an approach for patient data collection in healthcare institutions based on cloud computing that provides security with respect to confidentiality of transferred data, authentication, and authorization. Likewise, Zhang and Zhang [ZZ11] realize a secure platform for the cloud-based IoT in the context of ambient assisted living and telemedicine that relies on rudimentary security measures such as transport security and authentication using passwords.

To secure health data when it is outsourced to the cloud, Lounis et al. [LHBC12] particularly focus on guaranteeing confidentiality and integrity of outsourced medical data with minimum management and processing overheads. Thilakanathan et al. [TCN+14] propose a platform that realizes mobile telecare by allowing doctors to remotely monitor patients. For this, they rely on the cloud as a central data store which requires them to take special care of security, confidentiality, and access revocation. In a similar context, Li et al. [LYZ+13] and Liu et al. [LHL15] propose approaches for realizing the scalable and secure sharing of personal health records using the cloud. To secure the health records in this setting, they make use of attribute-based encryption respectively signcryption. In contrast to our work, relying on attribute-based encryption introduces non-negligible performance penalties and makes revocation of access rights costly.

On a more general scale, other researchers focus on securely outsourcing general-purpose IoT data to the cloud. Pooja et al. [PPP13] realize the protection of IoT data already within the IoT network. To further increase the security of outsourced data, they make use of two separate clouds for storing the encrypted IoT data respectively the keying material needed for decryption. In different scenarios, architectures utilizing a trusted third party similar to our trust point-based security architecture have been proposed. However, these approaches are typically restricted to securing the transport of data and do not consider the object security that is crucial to our scenario. The Federal Office for Information Security in Germany [Fed14] specifies a trusted gateway to guarantee privacy in intelligent energy networks. Our security architecture shows some similarities to this approach. However, our architecture allows a much more fine-grained access control for data. There are also a number of architectures involving a trusted third party that have been proposed in the context of cloud computing. Kamara and Lauter [KL10] propose an architecture similar to ours with respect to a trusted gateway encrypting outbound data and managing access policies. However, they do not consider the secure processing of data in the cloud. Additionally, they require the requesting of access tokens from the gateway to access data. Thus, in contrast to our approach, data stored in the cloud is only available when the gateway is reachable.

The Twin Clouds architecture proposed by Bugiel et al. [BNSS11] utilizes garbled circuits for encrypting both data and programs in a trusted environment before passing them to the untrusted public cloud. After a costly setup phase, which has to be performed per data item, computations can be executed obliviously in the untrusted cloud. However, the encrypted programs are limited to simple operations and require re-encryption after each execution. Pearson et al. [PMCR11] introduce a cloud design similar to ours that focuses on fine-grained access control for outsourced data. While their approach focuses on sticky policies that have to be enforced by a

trusted third party, our solution introduces a flexible design for object security for IoT data in the cloud.

All these approaches consider security aspects when outsourcing (IoT) data to the cloud. However, they do not consider flexible configuration of security mechanisms and a transparent access to protected IoT data for cloud services. Still, flexible configuration of security mechanisms is required to support different application scenarios, while transparent data access allows also non-security experts to develop cloud services. To enable flexible configuration of security mechanisms, Itani and Kayssi propose SPECSA [IK04], a policy-driven security architecture. Their policy format allows specifying which parts of a message have to be protected. However, they assume messages with a fixed structure and use the same encryption key for all parts of the same security level. Consequently, their approach is not suitable for the cloud-based IoT scenario, as this scenario requires fine-grained, flexible access control. Likewise, several approaches for supporting developers by abstracting from security paradigms have been proposed. GSSAPI [Lin00] provides security services to protect the communication between two entities. However, in our scenario, we require security at the granularity of objects instead of communication channels to protect IoT data also during storage. JSAL [HWZ04] is a security aspect library that requires developers to apply security measures manually, hence requiring expertise in the area of security. On the contrary, our approach does not require service developers to be security experts.

To conclude, there is a need for protecting IoT data and providing a security abstraction layer for accessing protected IoT data in the cloud that allows non-security experts to develop privacy-preserving cloud services.

5.2.2 Protecting IoT Data in the Cloud

To protect IoT data when it is outsourced to the cloud and thus lay the foundation for providing a security abstraction layer to cloud services, we now present our trust point-based security architecture for IoT data in the cloud. To this end, we first have a closer look at the flow of IoT data in the cloud-based IoT scenario. Then, we introduce the trust point, a logical entity for protecting potentially sensitive IoT data already within the local IoT network and thus still under the control of the user. Finally, we discuss measures for representing diverse IoT data and protecting it until it reaches an authorized cloud service.

5.2.2.1 Flow of IoT Data

We now describe a typical flow of IoT data in our scenario *prior* to any secure measures and provide a high-level description of the involved processing steps as shown in Figure 5.2. A data flow commonly starts within an IoT network and consists of periodically generated *data items*. In other words, a data item is an atomic fragment of an IoT data stream and represents the reading of one IoT node at a specific point in time. The payload of a data item consists of one or more

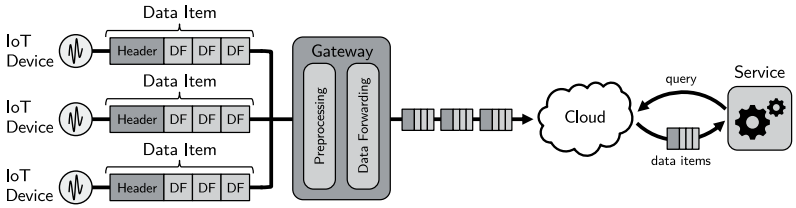

Figure 5.2 Data items consist of meta information (*header*) and data fields (*DF*) that represent individual sensed values. The gateway performs preprocessing steps on data items and uploads them to the cloud. Authorized services can then access the data items.

data fields which contain the measured values from individual sensors deployed on a specific IoT node. Additionally, data items contain a *header* with meta information such as the time and location of measured values.

Within the IoT network, the individual IoT nodes forward data items to a *gateway* device which is used to bridge between IoT and Internet protocols and is responsible for uploading IoT data to the cloud for further storage and processing. Consequently, *all* data items generated inside the IoT network traverse the gateway as the last entity situated inside the IoT network. As a network element, the gateway may be limited with respect to storage and processing resources. Thus, it may neither be able to store large amounts of IoT data nor perform excessive computational tasks.

After an IoT data item is received at the entry point to the cloud, it is stored persistently in the storage backend of the cloud. In addition to this storage service, the cloud also offers a platform for services. This cloud platform hosts third party services that process data items. To start this processing for (a fraction of) her IoT data, the user authorizes individual services to perform the processing of her relevant data items (at the granularity of individual data fields). The authorized services then request the corresponding data items from the cloud and process these data items. Finally, the user can access the results of the processing via an external service interface such as a website or a mobile app or have the results stored in the cloud, possibly again encrypted.

5.2.2.2 Trust Point-based Security Architecture

All devices within an IoT network are under the control of the user who operates this network. However, once IoT data leaves the IoT network, the user loses control over her data. Thus, the gateway marks the border of the user's control or privacy sphere. The central idea of our security architecture is to enhance this user-operated gateway with additional mechanisms that enable the secure outsourcing of IoT data to the cloud. More precisely, our enhanced gateway pre-processes the generated IoT data on behalf of the user and applies confidentiality as well as integrity protection before uploading the protected IoT data to the cloud. As this enhanced gateway device is owned and thus trusted by the user, we call it the *trust point*.

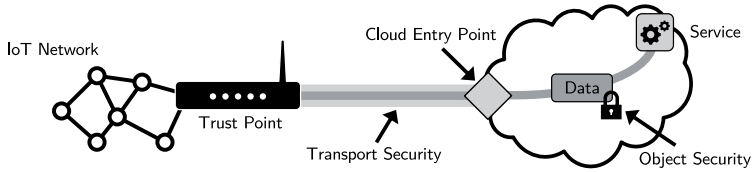

Figure 5.3 Transport security is terminated as soon as data reaches the cloud entry point. To protect IoT data between cloud entry point and service, we additionally employ object security.

As shown in Figure 5.3, the trust point manages the upload of IoT data to the cloud as it is the gateway device of the IoT network. Since the communication between the trust point and the cloud traverses the Internet, the transport channel between the trust point and the cloud has to be secured. This is not only required to provide confidentiality of the transferred information but also to ensure the mutual authentication of the two communication peers. Thus, the trust point can be sure that it is indeed communicating with the cloud. Similarly, the cloud can verify the identity of the trust point and thus the user.

However, mere transport protection does not suffice to protect the transmission and storage of IoT data in the cloud as transport protection is terminated at the cloud entry point (cf. Figure 5.3). At this point, the transport security mechanisms are stripped from the IoT data. Without further protection, plain data would reside unprotected within the cloud environment. To still achieve end-to-end security from the trust point to an authorized service, even during storage, the trust point adds additional object security mechanisms to the IoT data before transmitting it securely to the cloud. To this end, the trust point encrypts individual data fields and signs each data item before sending it towards the cloud. The plain information carried by IoT data can now neither be accessed nor undetectably modified by an unauthorized third party. Furthermore, the additional integrity protection cryptographically guarantees the accountability of IoT data to a specific user, i.e., cloud services can be sure that data indeed originates from this user.

This approach to object security is similar to digital rights management (DRM) [BBGR03] when considering cloud services as end-user devices in the DRM case. However, the main difference is that we do not require enforcement of data access control on the service side. The straightforward and most efficient solution for confidentiality protection is symmetric key encryption, e.g., using AES. While we focus on symmetric key encryption in the following, our security architecture conceptually also supports the use of order-preserving and deterministic encryption [BW07, PRZB11] to allow for search and sort operations on stored (encrypted) IoT data (cf. Section 3.4). The integrity protection of our architecture is based on asymmetric key cryptography. To guarantee integrity of a data item, the trust point signs it with a private key such that integrity protection covers the complete data item.

To access data items and individual data fields, cloud services require access to the symmetric keys used to protect data items, which we call *data protection keys* in the following. This access has to be authorized by the user. To achieve this goal,

the trust point is also responsible for the management of data protection keys. We discuss in the following how the trust point can manage the data protection keys despite its restricted storage capacities. Most importantly, we have to empower users to make an informed decision regarding which cloud services to authorize. Hence, cloud services have to provide a service description (e.g., in a cloud service marketplace) which contains high-level information about the purpose of the service and how the service uses the IoT data provided.

Conformance of the service implementation to the service description must be assured, e.g., through an audit by the infrastructure provider or another trusted third party, similarly to practices applied in today's app stores on smartphones. This conformance is expressed via a cryptographic signature issued by the auditor that covers the service description and the service's public key. After verifying this signature, a user who wants to grant a service access to her IoT data and agrees with the service description provides the data protection keys used for the protection of the data fields to the service. This is achieved by instructing the trust point to encrypt the respective data protection keys with the public key of the service and to transmit this secured information to a key store located in the cloud. The purpose of this key store is twofold. First, it offloads the trust point from the burden of frequent and repeated key requests causing expensive public key operations or the need to store a large number of keys. Second, it relaxes the requirement that the trust point needs to be continuously available. In our architecture, connectivity to the trust point is only necessary to initially grant cloud services access to IoT data. After this one-time authorization, cloud services can retrieve the data protection keys from the key store in the cloud and decrypt them using their private keys even if the trust point is temporarily unavailable.

5.2.2.3 Representation and Protection of IoT Data

Our goal is to store IoT data securely in the cloud such that it can only be processed by authorized cloud services. To this end, the trust point encrypts sensitive IoT readings using a symmetric cipher before uploading it to the cloud. The encryption process is influenced by a user-configurable access control list containing services that are authorized to (partially) obtain and process the user's IoT data. Now, only entities in possession of the data protection key used for encrypting an IoT data item have access to this specific data item. Consequently, to grant a cloud service access to a given data item, the trust point has to provide this cloud service with the corresponding data protection key. To this end, the trust point asymmetrically encrypts the corresponding data protection key with the public key of the cloud service that should gain access to the IoT data and forwards the resulting encrypted data protection key to the respective cloud service(s).

IoT data originating from a single IoT node can contain multiple sensor readings from different sensors. For example, a data item measured by a meteorological sensor might consist of multiple single readings such as humidity and temperature. More specifically, sensed information varies considerably regarding its structure (i.e., the serialization of measured data and meta information), the number of measurements

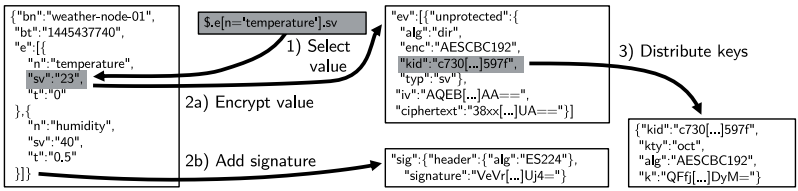

Figure 5.4 Users define data fields in a SenML-encoded data item that should be protected using JSONPath. We use JSON Web Encryption and JSON Web Signature to encode encrypted data fields, signatures, and data protection keys in a standardized manner.

fields (e.g., a single value for a simple temperature sensor or multiple values in case of a complex industrial control unit), and the units of these fields (e.g., degree Celsius or hertz). Likewise, cloud services often only require access to parts of sensor readings. By supporting the encryption of individual data fields, i.e., parts of IoT data, we thus realize fine-grained access control.

However, this requires a unified representation of diverse IoT data items. To this end, we rely on SenML [JSA+17], which has been proposed for standardization at the Internet Engineering Task Force (IETF). SenML supports JSON, XML, and Efficient XML Interchange for serializing IoT data. In the following, we focus on the JSON representation of SenML to showcase our approach. Still, our findings can also be extended to other data models or other serializations, e.g., XML. Independent from the actual representation and serialization of IoT data, we identify the following three essential tasks as depicted in Figure 5.4: 1) identifying those parts in the serialized IoT data item that should be covered by the protection, 2) performing the necessary cryptographic operations and augmenting protected IoT data such that an authorized cloud service can reverse these operations, as well as 3) securely distributing the employed data protection keys to authorized cloud services. We discuss these three tasks in more detail in the following.

Specifying Coverage of IoT Data Protection

We assume that we operate on IoT data items that are readily serialized in SenML. An essential part of protecting such SenML-encoded IoT data items for the cloud then is to encrypt the contained information. However, encrypting IoT data items as a whole is infeasible as certain meta information (e.g., IoT node identifier and timestamp) is required for indexing purposes to afford an efficient retrieval of IoT data in the cloud. Moreover, such holistic protection would restrict service access granting to an all-or-nothing approach as all information would be encrypted with the same data protection key. Especially in industrial settings [JBM+17], however, it is necessary to break down access granting to individual data fields. This way, the manufacturer of an industrial machine can, e.g., access certain data for monitoring the health of the machine operated by one of its clients without getting to know details about the product that is currently being processed on this machine.

Consequently, to provide confidentiality on a fine-grained basis, we require a way to address parts of an IoT data item. We facilitate JSONPath [Gös07] for this purpose, which allows us to address arbitrary fields in a JSON object (similar functionality is offered by XPath for XML). This way, the parts of an IoT data item that should be encrypted can be specified by the user (Step 1 in Figure 5.4). Notably, digital signatures that provide integrity and authenticity of the protected IoT data item cover the entire data item and thus do not require such fine-grained control.

Representation of Protected IoT Data

In the next step, the data fields identified with JSONPath need to be encrypted. To this end, we use standard symmetric encryption that affords efficient protection of bulk data. Still, we design our approach to be flexible regarding the employed symmetric-key primitives and key lengths to allow for scenario-specific security and performance trade-offs and to account for potential future advances in cryptography that may lead to certain primitives no longer being considered secure (as often observed in the past). However, due to this flexibility, simply replacing the plain value in the IoT data item with the encrypted value is insufficient. Instead, the cloud services require additional information, e.g., the used encryption algorithm, an identifier for the used data protection key, or the initialization vector to decrypt the protected data fields. Hence, this information additionally has to be encoded in the IoT data item. To express this information, we employ JSON Web Encryption (JWE) [JH15], which is a standard for representing encrypted content using JSON. Thus, the plain value in the IoT data item is replaced by a JWE object that contains the encrypted value and the additional information needed for decrypting this value (Step 2a in Figure 5.4). Additionally, the integrity and authenticity of the whole IoT data item should be protected. To this end, the common best-practice is to use public key signatures. Thus, we require each data source to be in possession of a public/private key-pair. This can easily be achieved using today's public key infrastructures. Similarly to JWE, we employ JSON Web Signature (JWS) [JBS15] to represent a public-key signature with JSON. To this end, we add a JWS-encoded signature to the protected IoT data items (Step 2b in Figure 5.4).

Distributing Keying Material

In the final step, we have to distribute the data protection keys to the authorized cloud services. These keys are needed by the cloud services to decrypt the individual data fields of an IoT data item. Here, we assume that also each cloud service is in possession of a public/private key-pair. Again, such functionality can be readily supplied by today's public key infrastructures. To grant a cloud service access to (parts of) an IoT data item, the trust point encrypts corresponding data protection keys with the public key of the respective service and uploads the result to the cloud. Here, similar to JWE and JWS, we leverage JSON Web Key (JWK) [Jon15] to represent the encrypted data protection key in JSON format (Step 3 in Figure 5.4). Thus, only an authorized service is able to decrypt the data protection key and thus gain access to the IoT data. Whenever the cloud service requires the data protection

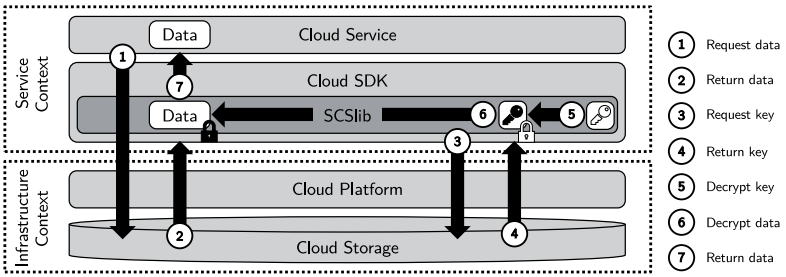

Figure 5.5 When a cloud service requests a protected IoT data item, SCSlib transparently handles all necessary security operations, i.e., requesting necessary data protection keys from the cloud, verifying the digital signature, and decrypting individual data fields.

key for decrypting an IoT data item, it queries the cloud for that key and decrypts it with its own private key. We additionally periodically exchange data protection keys to increase security [Kra96,EMM06] and to offer time-based fine-grained access control with respect to these key change intervals. More precisely, users will be able to provide cloud services access to their IoT data with respect to the time dimension at the granularity of key change intervals.

5.2.3 Transparent Access to IoT Data for Cloud Services

The processing of IoT data by a cloud service requires the verification of the integrity of received data, the decryption of the symmetric data protection key, and finally the decryption of the actual IoT data. However, correctly implementing these necessary security mechanisms is complicated, especially since developers of cloud services often are no security experts [Coo18]. Thus, to allow cloud service developers to access protected IoT data in the cloud without the need to care about decryption, signature verification, and key management, we introduce the Sensor Cloud Security Library (SCSlib) that realizes transparent decryption of IoT data and the verification of data integrity by a cloud service. We realize SCSlib as a C library that builds upon the cryptographic algorithms implemented by the OpenSSL library [VMC02].

We provide an overview on how SCSlib integrates into the process of cloud services querying for IoT data in Figure 5.5. In Step 1, the cloud services use the methods provided by the cloud to request one or multiple IoT data item(s) from the cloud. Subsequently, the cloud platform returns the requested IoT data item(s) from the cloud storage in Step 2. To ensure that only authorized cloud services can access the corresponding data fields, these are encrypted and require decryption before they can be utilized. To this end, SCSlib requests the necessary data protection keys in Step 3 and the cloud platform returns these keys from the cloud storage in Step 4. In Step 5, SCSlib then uses the private key supplied by the cloud service to decrypt the data protection keys. Hence, SCSlib can now decrypt exactly those data fields the cloud service is authorized to access in Step 6. Finally, SCSlib returns the decrypted IoT data item(s) to the cloud service in Step 7.

In the following, we discuss our design and implementation of SCSlib in more detail with respect to the following three main functionalities: (i) interfacing with the cloud, (ii) processing of IoT data items, i.e., verification and decryption, as well as (iii) caching of cryptographic keys for performance improvements.

Interfacing with the Cloud

Our design of SCSlib is driven by the goal to provide flexibility and re-usability on the one hand and transparency for cloud service developers on the other hand. Hence, we decide to develop a library that can be integrated into a cloud service SDK such as Google Cloud SDK and Amazon Web Services SDK or alternatively directly being integrated by the service developer, e.g., if the underlying SDK does not integrate SCSlib (yet). By integrating SCSlib into SDKs or directly into cloud services, all security-critical computations take place in the context of the service and no secrets (e.g., the service's private key) have to be revealed to third parties (cf. Section 5.2.1.2). Additionally, this design still enables cloud service developers to implement (parts) of the necessary cryptographic operations themselves if they do not (fully) trust the open source implementation provided by SCSlib. Consequently, we further reduce security and privacy concerns when outsourcing IoT data to the cloud by increasing transparency over the employed security mechanisms.

For the decryption and verification of IoT data items, SCSlib needs access to the public key of the data source as well as the data protection keys used for encrypting the individual data fields. We designed SCSlib to use callback functions, i.e., functionality provided by the cloud service SDK, for retrieving the necessary keys. This enables each cloud service SDK to implement the communication with the cloud infrastructure specifically tailored to their individual deployment scenario.

Processing of IoT Data Items

To invoke the processing of a IoT data item, SCSlib provides a slim API that consists of three public methods: `sc_verify_data_item()` for verifying the integrity of a data item, `sc_decrypt_data_item()` for decrypting a data item, and `sc_process_data_item()`, which combines the previous two methods. Cloud services pass data items to the library as string-encoded JSON objects, which yields a simple and portable interface. As discussed in Section 5.2.2.3, SCSlib conceptually also supports other methods for serializing IoT data based on SenML such as XML and Efficient XML Interchange [JSA+17].

When processing IoT data items, integrity and authenticity of the IoT data item have to be checked first. To this end, SCSlib looks up the public key of the data source using the corresponding callback function (see above) and then verifies the digital signature of the IoT data item using the retrieved public key. To decrypt the IoT data item, SCSlib iterates recursively over the JSON-serialized object to search for JWE objects representing an encrypted measurement value. For each JWE object within the IoT data item, SCSlib requests the data protection key that is needed to decrypt this data field using the above-described callback function.

Once the data protection key has been received, SCSlib decrypts this key using the private key provided by the cloud service (see above) and subsequently uses the data protection key to decrypt the encrypted measurement value. As a result, we retrieve the original value in the JSON-serialized IoT data item. If a cloud service is not permitted access to a specific data field, the corresponding data protection key is not available to this service. Conceptually, there are two options for handling this exception. Either the still encrypted measurement value can remain in the data item (which allows the cloud service to notice that it cannot access this specific field) or it can be removed (which increases performance when parsing the resulting smaller IoT data item). Our implementation of SCSlib supports both approaches and the actual behavior can be set via a configuration flag.

Caching of Cryptographic Keys

When processing data items, SCSlib operates on different keys for decrypting measurement values and verifying integrity and authenticity of IoT data items. However, which specific keys are actually needed cannot be determined before a given data item is processed, especially when considering random access to IoT data items (i.e., data is not accessed in chronological order). Furthermore, since data protection keys that are needed to access measurement values are encrypted with the public key of the service, they have to be decrypted before they can be used. Likewise, data protection keys are often used more than once during a key change interval (cf. Section 5.2.2.3). To prevent unnecessary overhead, we hence strive to request each data protection key only once and consequently also decrypt each key only once.

To achieve this goal, we introduce internal caches in SCSlib for decrypted data protection keys as well as public keys of data sources used to verify integrity and authenticity of IoT data items. As long as a key is present in the cache, SCSlib does not have to request it from the cloud and, in the case of data protection keys, decrypt it, again. We show in our evaluation that this has a huge impact on the overall performance of processing protected IoT data in a cloud service. The cache size as well as the caching algorithm used by SCSlib can be configured. For our evaluation of SCSlib, we implemented first in first out (FIFO) and least recently used (LRU) as cache management schemes. In the context of operating on encrypted IoT data, FIFO is especially well suited when processing data in isolated batches, while LRU excels in situations where certain data protection keys are used more often than others.

5.2.4 Evaluation

To prove the feasibility of our approach and quantify the performance of SCSlib, we conduct a thorough performance evaluation. As a foundation for this evaluation, we implement a simple cloud service SDK using the C programming language that returns requested IoT data items, data protection keys, and data source's public keys from a static database as well as a cloud service that triggers the decryption and verification of IoT data items. For our evaluation, we use the cryptographic

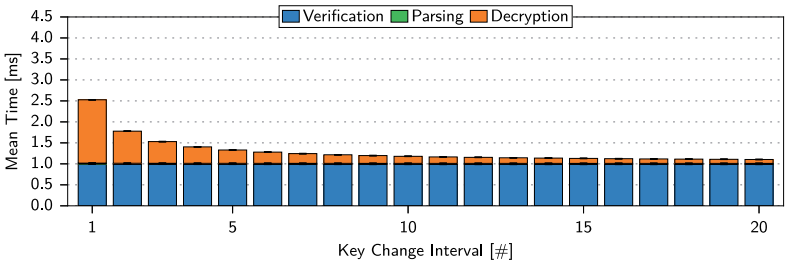

Figure 5.6 The mean time for processing one IoT data item with one data field for an unlimited cache size decreases with increasing key change intervals.

primitives AES with 128 bit keys in CBC mode for encrypting data fields, RSA with 2048 bit keys for encrypting the corresponding data encryption keys, and ECDSA with the NIST curve P-256 for digital signatures. To allow others to reproduce our results, we use Amazon Web Services third generation EC2 64 bit instances of type large (m3.large) [AWS18a] running Ubuntu 12.04 LTS to perform our measurements. For each measurement point, we conduct 50 measurement runs, each consisting of the processing of 1000 IoT data items. We depict the average processing time per data item for these measurements with 99 % confidence intervals in the following.

We first establish a baseline by examining the minimal costs of processing protected IoT data in the cloud when performing only the absolutely necessary operations. To this end, we choose cache sizes (cf. Section 5.2.3) such that each key has to be requested only once. This effectively emulates an infinite cache size. The main influence factors on the time required for processing of protected IoT data in the cloud then are the size of the key change interval, i.e., how often the data protection key for IoT data from the same data source is changed (cf. Section 5.2.2.3), and the number of data fields, i.e., how many encrypted measurement values are contained in one IoT data item.

In Figure 5.6, we show the average time for processing one protected IoT data item with one data field *with respect to the key change interval*. Here, we use an intuitive notion of the key change interval, i.e., after how many items the data protection key is exchanged. The results show that even for a key change interval of 1, the cloud service is able to process 397 IoT data items per second. For more realistic key change intervals, this rate increases to more than 900 data items per second (for a key change interval of 20). Especially for larger key change intervals, the processing time is then dominated by the verification of the digital signature, which requires about 0.99 ms irrespective of the key change interval. In addition, parsing a data item and processing keys only amounts to 0.02 ms. The time needed for decrypting the IoT data item decreases from 1.51 ms to 0.09 ms when increasing the key change interval from 1 to 20. In the following, we fix the key change interval to 10, as this constitutes a good trade-off between flexibility (with respect to time-based fine-grained access control) and performance.

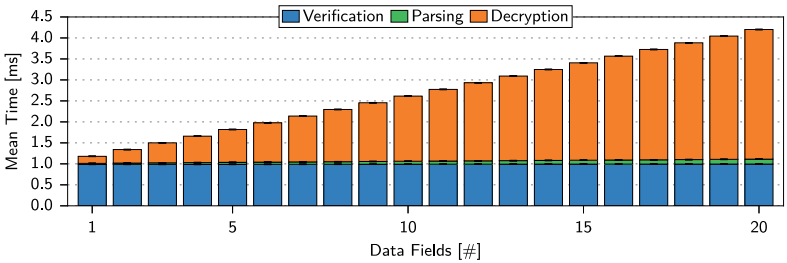

Figure 5.7 The mean time for processing one IoT data item for a key change interval of 10 and an unlimited cache size linearly increases with the number of data fields.

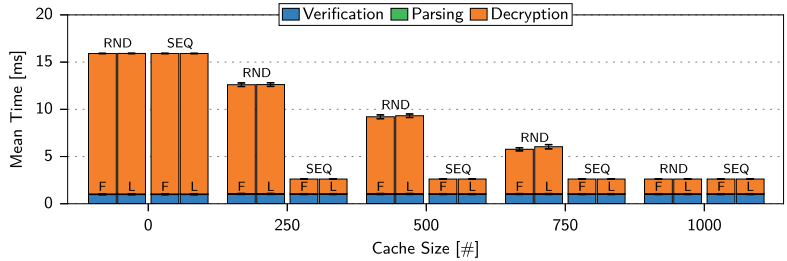

Figure 5.8 SCSlib's caching approach considerably reduces the overhead for processing protected IoT data in the cloud, both for random (RND) and especially sequential (SEQ) access to IoT data when using FIFO (F) and LRU (L) as cache management scheme.

Similar to our measurement setup for increasing key change intervals, Figure 5.7 shows the processing time for one data item with an *increasing number of data fields* for a key change interval of 10 and an infinite cache size. Also in this setting, signature verification constantly accounts for a processing overhead of 0.99 ms. As expected, the time needed for parsing and decrypting the data item increases linearly with the number of data fields from 0.19 ms (0.02 ms for parsing and 0.17 ms for decrypting) for one data field to 3.21 ms (0.12 ms for parsing and 3.09 ms for decrypting) for 20 data fields. These numbers show that SCSlib can handle a throughput of 238 to 846 IoT data items per second depending on the number of data fields in each data item. Based on these results, we use data items with 10 data fields for our remaining evaluation, since these adequately represent the performance of a wide range of realistic IoT data item sizes as we, e.g., observe for the cloud-based IoT platform `dweet.io` [Bug18]. The results obtained so far constitute a lower bound for the processing performance of handling protected IoT data items in the cloud, to which we compare our caching optimizations in the following.

To this end, we report on the performance of SCSlib with respect to different *cache sizes* and *cache management schemes* for IoT data items consisting of 10 data fields for a key change interval of 10 in Figure 5.8. We differentiate between the two cache management schemes FIFO (denoted by "F") and LRU (denoted by "L") and

vary the cache size between 0 keys (no caching) and 1000 keys (all keys cached). Furthermore, we consider both, sequential (denoted by "SEQ") and random (denoted by "RND") processing of IoT data. In the sequential case, IoT data is processed in strict temporal order (which is often observed in real-world scenarios where cloud services operate on streams of IoT data), while in the random case, IoT data is processed in an arbitrary, non-deterministic order (which, while rather artificial, is the most challenging scenario with respect to caching). Our results show that caching indeed has an enormous impact on performance. With an appropriate cache size, we achieve a 6-fold reduction in processing time from 15.91 ms to 2.62 ms, which matches the lower bound we established in our previous measurements (cf. Figure 5.7). As expected, for sequential processing, we are able to achieve the best possible performance as soon as the cache size equals or exceeds the number of simultaneously required data protection keys (in our example, this number is ten, as we have ten data fields per IoT data item). Likewise, the processing time for random processing decreases linearly with the cache size, as the likelihood that a key is still in the cache increases with the cache sizes. Our evaluation also shows that the performance difference between FIFO and LRU is negligible for the two considered scenarios.

To conclude our evaluation of SCSlib, the results of our performance evaluation show that it is feasible to process protected IoT data items in a cloud service. Through SCSlib, we enable interoperability be abstracting from security functionality and thus allow for an open environment and integration with different cloud offers. Furthermore, due to SCSlib's caching, we can considerably improve the per-data item processing time. SCSlib enables non-security experts to develop privacy-preserving cloud services for the cloud-based IoT. Still, these advantages come with higher transmission and storage overheads than tailor-made solutions for individual scenarios. However, these overheads can be minimized by employing optimization techniques such as data item compression, e.g., using Concise Binary Object Representation [BH13] or our compact privacy policy language (cf. Section 4.2).

5.2.5 Summary and Future Work

The cloud-based IoT, i.e., the interconnection of the IoT with the cloud to benefit from the elastically scalable and always available resources provided by the cloud computing paradigm, promises to simplify storage and processing of collected IoT data, enables the utilization of the same IoT by multiple services, and eases the fusion of IoT data across users. To counter resulting security and privacy concerns, we presented a best-practice approach for encoding and protecting IoT data in the context of cloud computing. More specifically, we introduced a trust point-based security architecture that guarantees object security between IoT networks and cloud services by cryptographically enforcing a fine-grained and user-centric access control scheme. The involved security mechanisms, however, are difficult to implement for cloud service developers who do not specialize in security. Hence, we proposed SCSlib, a library that enables cloud service developers to transparently access protected IoT data in the cloud without having to deal with the details of implementing security mechanisms.

SCSlib is a security library that can be included into cloud service SDKs to transparently integrate into the process of querying IoT data from the cloud. Since SCSlib is based on standardized and best-practice approaches for representing and protecting IoT data, i.e., SenML [JSA+17], JWE [JH15], and JWS [JBS15], it is sufficiently flexible to satisfy a wide range of performance and security requirements. For example, SCSlib can be easily extended to also support IoT data items that have been serialized using XML or Efficient XML Interchange instead of JSON if need arises. Likewise, by relying on a standardized expression of security mechanisms, SCSlib conceptually supports a wide range of different security algorithms. Thus, SCSlib can adapt to future security requirements, e.g., by migrating to a symmetric cipher with a higher security level if necessary. Our evaluation of SCSlib performed on commodity public cloud infrastructure confirms the feasibility of SCSlib, especially when considering the performance gains of SCSlib's caching of cryptographic keys for sequential and random access to IoT data stored in the cloud. By employing caches for cryptographic keys, SCSlib achieves a 6-fold increase in processing throughput and is able to process hundreds of encrypted and signed IoT data items per second.

As shown in our performance evaluation, the verification of public key signatures for integrity protection purposes constitutes a major performance bottleneck of our current security architecture. Thus, in the future, we plan to investigate more efficient signature schemes that relieve cloud services from the high computational burdens implied by public key cryptography. Our idea here is to use hash chains [Lam81] to amortize the high computation cost of public key signatures across multiple data items. This is similar to the performance optimization for verifying control messages in D-CAM, our approach for distributed configuration, authorization, and management in the cloud-based IoT that we present in the subsequent section.

From a different perspective, our current trust point-based security architecture does not fully support the revocation of once granted data access rights. In our security architecture, the user can revoke an access policy to prevent a cloud from gaining access to any IoT data produced in the *future*. However, to cryptographically revoke a cloud service's access to IoT data items already stored in the cloud, these data items have to be re-encrypted and the resulting data protection keys distributed to the remaining authorized cloud services. Especially for large amounts of IoT data, performing this re-encryption of data and re-distribution of keys on the trust point quickly becomes infeasible. A promising approach is to utilize proxy re-encryption for performing these steps securely in the cloud [YWRL10]. This concept allows offloading the necessary expensive computations to an untrusted cloud environment without revealing any information about the underlying IoT data.

To conclude, with SCSlib we support the secure incorporation of the two technologies, IoT networks and cloud computing, with respect to the confidentiality of IoT data. However, when moving towards the cloud-based IoT, also the configuration, authorization, and management of IoT devices and networks are typically outsourced to the cloud. This outsourcing does not only raise privacy concerns, but also serious safety concerns. In the following, we hence propose a distributed architecture that enables users to securely configure, authorize, and manage their IoT devices across network borders without having to trust the cloud.

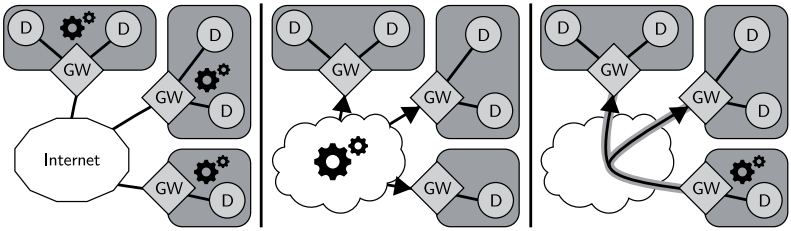

Figure 5.9 Different current IoT deployment models realize configuration, authorization, and management (a) within isolated IoT networks or (b) centrally in the cloud. In contrast, D-CAM (c) enables *distributed* control *across* network borders without having to trust the cloud.

5.3 D-CAM: Distributed Control in the Cloud-based Internet of Things

The IoT enables the worldwide interconnection of "smart things" to enhance important aspects of everyday life, e.g., in pervasive healthcare, assisted living, and smart cities (cf. Section 2.4). With SCSlib (cf. Section 5.2), we proposed an approach to protect the confidentiality of IoT data when sending it to the cloud. However, as IoT devices can directly influence the physical world (e.g., Internet-connected implanted medical devices [SRLO15] or robotic arms in factories [AIM10]), it is additionally important to secure the access to the configuration, authorization, and management of these devices to prevent severe physical damage [SRLO15]. As depicted in Figure 5.9a, in traditional deployments, the control operations of IoT devices and network are securely realized *within* individual networks, e.g., via cryptographically enforced access control lists [LHBC12, PTPS14]. This deployment model allows a user to efficiently manage and secure a *single* network within the IoT.

However, there is an increasing trend of interconnecting previously isolated IoT networks [MSPC12]. This trend ranges from users who want to interconnect their body area network and home network [HHK+16, SHH+18] to companies bridging complete factories via the Internet [HG15]. As discussed in Section 2.4, the predominant approaches to realize such interconnection utilize the high availability and elastic resources of the cloud. In this setting, as shown in Figure 5.9b, the cloud is used to facilitate management of networks and devices as well as to configure and authorize access to devices *across* network borders. This allows users to configure, authorize, and manage access to their devices across different networks. More specifically, a user can manage and configure devices in different networks from a single location without having to take care of the availability and reachability of individual devices that, e.g., reside behind a firewall.

Besides these enormous benefits, outsourcing configuration, authorization, and management of (potentially safety-critical) devices to the cloud poses huge security and privacy threats. These threats range from a curious cloud provider accessing confidential data to a malicious provider gaining physical control over safety-critical

devices. This includes rogue employees of the cloud provider and possible security breaches, jeopardizing the security and privacy of all cloud-controlled devices [CN12].

Hence, we deem it important to tackle the challenge of securely realizing configuration, authorization, and management in the cloud-based IoT. Due to the potential severity of attacks enabled by physical control, our prevalent focus lies in preventing a malicious cloud provider from controlling IoT devices. This focus further extends upon the security assumptions underlying SCSlib (cf. Section 5.2.1), where it is sufficient to protect the confidentiality of IoT data. To this end, we present D-CAM, our approach for achieving distributed configuration, authorization, and management *across* network borders as depicted in Figure 5.9c. D-CAM runs on the user-controlled gateways of individual networks and enables users to configure their *complete* federation of IoT networks from each of these networks. In contrast to entirely configuring IoT networks centrally in the cloud, D-CAM reduces the cloud to act as a highly available and scalable proxy for storing and forwarding tamper-resistant control messages. Thus, we achieve a reasonable trade-off between the advantages of the cloud-based IoT and strong security and privacy guarantees.

5.3.1 Controlling IoT Networks

We begin by providing an overview of our envisioned network scenario. From this scenario, we derive the challenges of securely achieving configuration, authorization, and management for cloud-interconnected IoT networks and discuss related work.

5.3.1.1 Network Scenario and Problem Analysis

In traditional IoT deployments, a network of IoT devices is connected to the Internet (and possibly the cloud) via a gateway controlled by the user (cf. Section 5.2.1.1). In rare cases, an IoT device directly acts as the gateway. We assume that the communication *within* the IoT network is properly secured (cf. Section 5.2.1.2), i.e., the internal IoT network communication provides confidentiality, integrity, and authenticity protection. To allow for interaction with an IoT network over the Internet in this setting, it needs to be properly configured. This involves (i) *configuration* of individual IoT devices, (ii) *authorization* of access to these devices (e.g., for sensing and actuating), and (iii) *management* of the overall IoT network and its structure. In the following, we refer to these operations as *control operations*. Handling control operations is well-studied for traditional single-network deployments. Such networks are typically configured on the single user-controlled *gateway* that connects to the Internet and thus is predestined to enforce all control-related tasks. For example, as the gateway manages connections to the Internet, it will only forward legitimate requests received from Internet hosts to the IoT devices in its network.

However, as the IoT evolves, we observe an increasing trend for bridging several IoT networks over the Internet. Yet, conveniently and consistently managing a *federated* IoT network is challenging. In a naïve approach, SSH or VPNs could be used to remotely control small groups of IoT networks. However, this requires gateways

to be addressable (not behind a firewall or NAT) and available (not offline, e.g., due to an unreliable wireless uplink) at configuration time, which is an unrealistic assumption for dynamic environments such as the IoT. Current state-of-the-art approaches [LVCD13, BDPP16] thus propose to steer control operations from the cloud. Using the cloud as a central hub to manage IoT devices of one user across network borders eliminates the need for managing each network separately and for setting up remote management solutions. In this setting, the user sends control messages to the cloud, which will relay them to all gateways in the user's federated IoT network (if a gateway is offline, it will be updated as soon as it comes back online). Such control messages can be sent in a variety of formats and protocols, e.g., using CoAP, SNMP, or NETCONF [SG16]. Hence, such systems need to be agnostic to the specific format and protocol used for control operations.

5.3.1.2 Security and Privacy Analysis

While the cloud enables the owner of a federated IoT network to perform control operations conveniently and efficiently, this comes at the price of security and privacy risks. In cloud-based systems, the prevalent security assumption is that the cloud provider can be partially, but not fully, trusted. Specifically, the cloud provider is typically considered to be semi-honest or honest-but-curious (cf. Section 2.3.2). That is, it will not disrupt the execution of the protocol and is thus limited to passively gathering information. Most importantly, a cloud provider, under these assumptions, will not tamper with messages it is supposed to relay. This is a widespread and reasonable assumption if the primary goal is to only protect the confidentiality of data. However, as the IoT connects the physical world to the Internet, security in the cloud-based IoT is not only about the privacy of information but additionally requires to guarantee (physical) safety. As a severe example, an adversary could remotely gain control over a pacemaker to modify a patient's heart rate [GZ15] after gaining access to the cloud. Consequently, only assuming an honest-but-curious cloud provider when considering control operations in the cloud-based IoT does not offer adequate protection for safety-critical tasks.

To illustrate this issue, we derive a set of severe attacks a dishonest cloud provider (or rogue employees and entities attacking the cloud) can launch in addition to those of an honest-but-curious cloud provider in the following.

Modification Attack: Changing messages before forwarding them, e.g., to replace parameters in a configuration message.

Insertion Attack: Creating new messages and sending them to devices in the network, e.g., to gain access to a specific device. This class of attacks also includes duplication of legitimate messages (also referred to as replaying) to cause an inconsistent system state.

Reorder Attack: Changing the order of messages before distributing them in the network, e.g., to change the semantics of the requests contained in the messages.

Withhold Attack: Deciding to (temporarily) not pass on certain messages to the network, e.g., to block the deauthorization of access to devices.

These attacks have in common that they can lead to severe consequences, e.g., if the cloud provider (or an employee or someone attacking the cloud provider) uses them to gain control over an actuator in the physical world. To account for these attacks in addition to protecting the privacy of data, we assume a *malicious-but-cautious cloud provider* (cf. Section 2.3.2) and design our system accordingly. In this attacker model, the cloud provider can launch any attack as long as this leaves no evidence. Notably, this does not necessarily imply that the cloud provider indeed behaves maliciously. Rather, it acknowledges that the cloud provider (or an employee) *can* potentially behave maliciously or be subject to attacks. Neglecting the resulting attack vectors would, e.g., enable attackers to gain control over devices in the user's IoT network. This attacker model is especially well-suited for our scenario, as cloud providers face serious consequence if misconduct is detected.

In this work, we do not aim to protect against insider attacks at the user side, e.g., originating from hacked gateways within the IoT network. Still, we show that D-CAM provides accountability, i.e., misbehavior of gateways (through errors or attacks) can be identified, which at least enables users to react to insider attacks.

5.3.1.3 Related Work

Different directions of research offer valuable input for our goal of securely realizing distributed control in the cloud-based IoT. We structure our discussion of related work by the following three main directions of research: (i) controlling access to data in the cloud-based IoT, (ii) secure audit logs, and (iii) blockchain approaches.

Access Control in the Cloud-based IoT. Similar to the efforts underlying our security library SCSlib (cf. Section 5.2), several approaches to control access to data in the cloud-based IoT have been proposed and we briefly recap the most relevant approaches here. These approaches typically encrypt data before uploading it to the cloud and perform access control through selectively releasing decryption keys. In the context of health data, Lounis et al. [LHBC12] leverage attribute-based encryption to distribute decryption keys, where a trusted third party globally defines access rights. Similar approaches for health data in the cloud have been proposed by Li et al. [LYZ+13] and Jahan et al. [JRSJ15] using attribute-based encryption, Thilakanathan et al. [TCN+14] based on a secure data sharing protocol, and Liu et al. [LHL15] by employing attribute-based signcryption.

On a more general scale, our trust point-based security architecture (cf. Section 5.2) constitutes a generic security architecture for outsourcing the storage and processing of IoT data to the cloud. In this setting, the user can grant cloud services fine-grained access to IoT data. To further increase security, Pooja et al. [PPP13] propose to separate storage and processing of IoT data by using two independent cloud infrastructures. In all of these approaches, access control is solely performed to protect the confidentiality of data and does not additionally consider the potentially safety-critical access to actuation capabilities. This problem is addressed by Picazo-Sanchez et al. [PTPS14], who securely implement a publish-subscribe approach for medical body area networks and realize fine-grained access control for commands sent to an IoT device. Their ciphertext-policy attribute-based encryption scheme

induces a processing overhead in the order of seconds compared to D-CAM's overhead in the order of milliseconds.

Furthermore and in contrast to our work, these approaches do not consider the secure federation of IoT networks across network borders. They either require a central trusted entity to perform access control [LHBC12, PPP13, LYZ+13, TCN+14, LHL15] or realize access control completely within isolated networks [PTPS14]. In contrast, D-CAM realizes full configuration, authorization, and management in the cloud-based IoT across networks. Porambage et al. [PBS+15] propose two group key establishment schemes to realize secure multicast in the IoT. Their scheme, however, does not consider many-to-many messages and the management of gateway groups.

Secure Audit Logs. From a different perspective and not specifically focused on the cloud-based IoT, secure audit logs aim at protecting integrity and authenticity of log files produced for auditing purposes [SK99]. Schneier and Kelsey [SK99] present a generic secure logging scheme that affords to detect any deletion or modification attempts even on a compromised host. Improving upon these results, Ma and Tsudik [MT09] introduce the concept of forward-secure stream integrity for secure audit logs specifically created and stored on untrusted hosts. Snodgrass et al. [SYC04] focus on tamper detection for log files of database management systems. Waters et al. [WBDS04] propose an encrypted and searchable audit log that also protects confidentiality of log file entries and still allows searching for log entries under encryption.

Although these approaches do not consider a distributed setting, i.e., multiple entities contributing to a log, they provide us with valuable input. Especially searchable encryption offers insights for future work to decrypt only relevant messages when processing the message log. Considering a distributed setting, Accorsi [Acc10] proposes a black box based on trusted computing to ensure authenticity and confidentiality of log entries. From a different perspective, Chong et al. [CPH03] propose to rely on tamper-resistant hardware to create secure audit logs in the context of DRM.

In contrast, we neither require an additional entity that can become a single point of failure nor have to rely on trusted hardware components. Addressing the issue of trusted third parties for verifying audit logs in distributed systems, BAF [YN09] realizes publicly verifiable forward secure and aggregate signatures. However, BAF still requires an offline trusted third party for guaranteeing audit log integrity. Specifically focusing on cloud computing, SecLaaS [ZDH13] enables the release of cloud users' audit logs, e.g., to aid forensic investigations, while still protecting the privacy of log entries. These approaches, however, are specifically tailored to log files and do not support the management of IoT networks.

Blockchain Approaches. Finally, our approach shares similarities with blockchain approaches, i.e., massively replicated histories of accepted messages, which are cryptographically linked using hash chains. In these systems, a distributed consensus protocol ensures the integrity of messages in the blockchain [CD16].

Bitcoin [Nak08] uses a blockchain to store monetary transactions. It has been extended to implement decentralized lookup stores [ANSF16] and access control systems [ZNP15] by embedding configuration messages into the blockchain. Matzutt

et al. [MHH+16, MHH+18, MHZ+18] show that a blockchain originally designed for financial transactions such as Bitcoin can be used as a general purpose content store. Using one of the mechanisms to insert data into Bitcoin's blockchain, Catena [TD17] realizes a non-equivocation log of application-specific statements.

Lately, blockchain-based systems have also been proposed to manage the IoT. Christidis and Devetsikiotis [CD16] report on a number of different isolated approaches where the blockchain is mainly utilized to clear payments. In contrast, Shafagh et al. [SBHD17] propose a system where the blockchain is used to afford for auditable storage and sharing of IoT data. In contrast to blockchain approaches, D-CAM's inherently strong trust within gateway groups operated by the same user eliminates the need for costly consensus protocols such as block mining in Bitcoin. Performance improvements proposed for Bitcoin, such as block pruning or leader election [EGSR16], are similar to storage optimizations we propose for D-CAM. However, in contrast the optimizations for D-CAM, block pruning still requires verifying the whole blockchain when joining the system.

5.3.2 Distributed Configuration, Authorization and Management

The goal of our work is to overcome the identified security and privacy challenges by realizing distributed configuration, authorization, and management (control operations) in the cloud-based IoT in the presence of a malicious-but-cautious cloud provider. To this end, we present D-CAM, our solution that bases on hash chains [Lam81] to create a distributed administrated log of control messages encoding configuration, authorization, and management operations. This allows us to create a secure timeline [MB02] of these control messages, which can be verified by any gateway in the federated IoT network. To this end, we first focus on achieving integrity as well as availability of control messages. We describe how to additionally achieve message confidentiality in Section 5.3.5.

In the following, we first provide an overview of D-CAM's design. Based on this, we describe how messages are appended to D-CAM's message log and how a federation of IoT networks can be managed based on D-CAM's paradigms. Then, we discuss how the integrity and authenticity of the message log can be verified. Finally, we show how the message log can be compacted to reduce storage space.

5.3.2.1 Design Overview

D-CAM operates in a scenario where multiple IoT networks are interconnected via the cloud to form one larger, virtual IoT network (cf. Figure 5.9c). As each individual IoT network is connected to the cloud via a dedicated user-controlled gateway, their interconnection requires the federation of said gateways, which we refer to as a user's *gateway group*. The task of D-CAM is the reliable distribution of control operations to all gateways in a gateway group in the presence of a malicious-but-cautious cloud provider. We assume that each gateway has a cryptographic identity, i.e., a public/private key-pair, and is controlled by the user owning the IoT network.

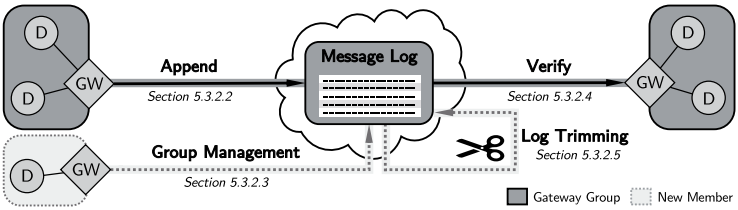

Figure 5.10 D-CAM's design centers around a message log which allows gateway group members to append new messages and verify the contained messages.

At the core of D-CAM resides a distributedly managed, cloud-hosted *message log* for each gateway group to which each gateway in the corresponding gateway group can append messages as illustrated in Figure 5.10. Furthermore, each gateway can verify the integrity and the authenticity of the message log. Messages in the message log immediately reflect control operations, i.e., (i) the *configuration* of devices in the IoT network, (ii) the *authorization* of access to the sensing and actuation capabilities of IoT devices, and (iii) the *management* of the IoT network itself.

As our focus lies on realizing the secure distribution of *arbitrary* control messages in federated IoT networks, we deliberately abstract from specific approaches for configuring individual IoT devices (e.g., CoAP, SNMP, or NETCONF). Furthermore, control messages in D-CAM also include the management of the gateway group itself, i.e., adding and removing gateways. Finally, D-CAM employs log trimming to make the process of joining a gateway group more efficient. Essentially, the message log constitutes a complete history of all control operations that were ever issued to manage one federated IoT network. In summary, D-CAM's message log is maintained in a distributed manner within a gateway group and—for the purpose of configuration, authorization, and management of IoT devices and networks—the cloud is reduced to a highly available message store and relay.

5.3.2.2 Appending to the Message Log

The target of D-CAM is to ensure that only authorized gateways can append control messages to the message log. Furthermore, no unauthorized entity should be able to modify, reorder, or remove messages. To achieve this goal, we protect control messages with a combination of sequence numbers, a hash chain, and digital signatures as shown in Figure 5.11.

In the following, we describe the process of appending one message to the message log and from now on refer to the gateway appending the message as its *initiator*. To avoid message collisions, the initiator first reads the sequence number of the most recent message, increments it by one, and adds it to the new message (dashed line in Figure 5.11). If two gateways simultaneously append a message, they will use the same sequence number and hence D-CAM is able to detect and resolve the collision as follows: The gateways in the gateway group accept only the message of the older group member (any other deterministic tie-breaker works as well due to the strong

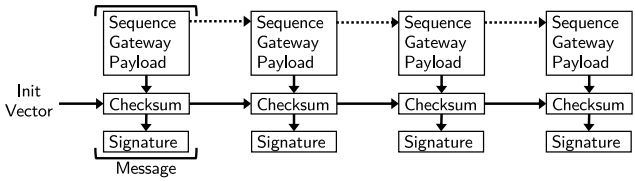

Figure 5.11 Each message in the message log is digitally signed by the originating gateway. All messages in the message log are interlinked via a hash chain.

trust assumptions within gateway groups) and ignore the second message. The unsuccessful gateway retries to append its message after processing the accepted message and updating its sequence number. This way, all collisions are resolved deterministically within the gateway group.

Furthermore, the initiator creates a checksum that covers the message itself as well as the checksum of the directly preceding message using a cryptographic hash function (solid line in Figure 5.11). Thereby, we create a hash chain [Lam81] that cryptographically links all messages in the message log. Due to the preimage resistance of cryptographic hash functions, messages can neither be altered nor reordered without invalidating the hash chain. The first message in the message log contains a random *initialization vector* instead of the previous checksum.

To enable other gateways in the gateway group to verify the integrity and authenticity of a message, the initiator digitally signs each message using its private key. This signature covers the checksum and thus also ensures integrity and authenticity of all previous messages. Subsequently, the initiator sends the message to the cloud, where it is stored and distributed to all gateways in the gateway group. Gateways that are offline or temporarily unavailable will update to the latest version of the message log as soon as they come back online.

Optimization. Creating reasonably secure digital signatures leads to a non-negligible performance overhead, as observed in our evaluation of SCSlib (cf. Section 5.2.4). Hence, with D-CAM we aim to reduce the amount of required digital signatures *without* diminishing the security level. We observe that in IoT deployments control messages often arrive in batches, e.g., if the user configures new devices or changes authorization of device access. If a gateway appends a batch of messages to the message log, it will add a digital signature only to the last message and send the complete batch to the cloud. The integrity and authenticity of the other messages in the batch is still guaranteed by the hash chain. We show in our evaluation that this enables us to considerably reduce D-CAM's processing costs.

5.3.2.3 Management of Gateway Groups

D-CAM uses the message log to secure the management of gateway groups, i.e., to ensure that only authorized gateways participate in a gateway group. Thus, D-CAM provides the same security level for the management of gateway groups as for regular

control operations. We differentiate between four group management operations: (i) *creation* of a gateway group, (ii) *adding* gateways to a group, (iii) *removing* gateways from a gateway group, and (iv) *termination* of a gateway group.

When creating a federated IoT network, the user also *creates* a new gateway group. To do so, she connects to one of her gateways and creates the gateway group, as well as a corresponding message log with a random initialization vector. To announce the creation of this new group and add itself as the first group member, the gateway creates an initial message using the initialization vector as identifier. Once the initial message has been stored in the cloud, the gateway group has been created.

To *add another gateway* to her gateway group, the user connects to the new gateway and creates a join request that is stored in the cloud (outside the message log). Now, she can connect to any gateway in her gateway group to review and accept the pending join request, thereby validating the public key of the joining gateway. To complete the process of adding the gateway to the gateway group, a group member appends a message to the message log that grants the public key of the new group member the right to append messages to the message log. Now, the new gateway is a full member of the gateway group.

Removing gateways from a gateway group in D-CAM works similarly to adding gateways. Any member of the gateway group can append a message to the message log that removes another gateway from the gateway group by revoking its public key. Upon receiving this message, the remaining members will not accept any further messages signed by the removed entity.

Finally, to *terminate* a gateway group once it is no longer needed, any member of the gateway group can append a special tombstone message to the message log. This tombstone message signals to all members of the gateway group that the group has been dissolved and no further messages to the message log will be accepted.

IoT devices themselves can also be managed with D-CAM. Here, D-CAM additionally stores routing information in the message log, i.e., to which gateway a device is connected. Furthermore, D-CAM's design is flexible enough to also store additional configuration parameters regarding the devices in a gateway group if need arises.

Optimization. In certain scenarios, it might not be desirable to allow each gateway group member to perform control operations, e.g., if a gateway is deployed in an untrustworthy or physically exposed environment. Hence, D-CAM also supports *passive gateways*, i.e., gateways that can only be configured using D-CAM but cannot initiate control operations. Gateways suspected to be especially vulnerable thus do not jeopardize the security of the whole network if they are compromised.

5.3.2.4 Verifying the Message Log

Whenever a gateway receives a message batch from the cloud, D-CAM must verify the integrity and authenticity of each individual message in this message batch. To this end, the gateway verifies messages one after the other in sequential order. When processing a message, the gateway first verifies the message's checksum by

computing the hash value over the message and the previous message's checksum. Then, the gateway reads the public key of the message's initiator from a local cache. This cache is updated whenever a non-passive gateway is added to or removed from the gateway group (cf. Section 5.3.2.3). This ensures that only messages originating from currently authorized gateway group members are accepted as valid messages by the gateways in a gateway group. Finally, the gateway verifies the message's digital signature and continues with the next message.

If the verification of a message fails, D-CAM drops this message and stops verifying the message log. We deepen our discussion on how D-CAM deals with verification errors as part of our security discussion in Section 5.3.3.

Optimization. In duality to appending messages, the processing time for verifying a control message is dominated by the effort required for checking the digital signature. Again, our scheme based on hash chains allows us to selectively employ an optimization. D-CAM can verify message batches by iteratively checking the checksum of each message but verifying only the signature of the last message in the message batch. With this optimization, we can guarantee the correctness of all messages in the batch only after verifying the last message. Thus, the batch size constitutes a trade-off between improved verification time and required buffer space as well as more complicated recovery in case of verification failures. Notably, this does not constitute a trade-off between security and performance, as the digital signature in combination with the preimage resistant hash chain commits to the integrity and authenticity of the complete message log.

5.3.2.5 Trimming the Message Log

The cumulated amount of control messages generated by a gateway group steadily increases over time. This becomes problematic as gateways joining a gateway group after a while need to process an excessive amount of messages to catch up with the current network state. At the same time, we observe that older control messages are often obsoleted by new messages, e.g., when overwriting a configuration or revoking an authorization. To leverage this potential for space reduction, D-CAM *trims* the message log by starting a new message log based on the network state at the time of trimming, thereby pruning all obsoleted control messages. This trimming allows for notably shorter bootstrapping times for new gateways, as they now do not have to process all messages that were ever issued to a gateway group anymore.

A dedicated gateway group member (e.g., the oldest) constantly keeps track of the amount of obsolete messages in the message log. To this end, this group member checks for each new message whether it obsoletes, i.e., overwrites, an old message. If the amount of obsolete message exceeds a specific threshold (depending on the group or device), the dedicated gateway trims the message log. To trim the message log, the gateway uploads a complete snapshot of the current network state to the cloud and adds a snapshot message to the message log. The snapshot message contains the snapshot's storage location and the hash value of the snapshot. When a new gateway joins a gateway group, it is provided with the hash over the latest snapshot and thus only has to verify the message log starting from the latest snapshot.

5.3.3 Security Discussion

Based on our description of D-CAM's design, we now briefly discuss how D-CAM protects against the attacks we identified (cf. Section 5.3.1.2) and hence guarantees integrity and authenticity of control operations in the cloud-based IoT.

Modification Attack. Digital signatures ensure that no unauthorized entity, e.g., a malicious cloud provider, can modify a message. Any modification will invalidate the message's signature and is easily detectable by any group member, causing a malicious-but-cautious cloud provider to refrain from launching this attack (cf. Section 2.3.2). Even with our optimization to not sign each message, we can easily detect mismatches in the hash chain even if only non-signed messages are modified.

Insertion and Reorder Attacks. No unauthorized entity can append new messages to the message log as they are unable to create valid digital signatures. Replaying, i.e., duplicating legitimate, signed messages, is prevented as this would imply recurring sequence numbers and checksum mismatches in the hash chain. The same detection strategy can be used for preventing reordering attacks, which would result in a mismatch in sequence numbers and a broken hash chain.

Withhold Attack. In contrast, detecting withholding of messages requires additional effort. We briefly outline two approaches: First, the members of a gateway group can use a side channel (e.g., by directly contacting each other) to periodically exchange status information, i.e., the sequence number and checksum of the latest message. Second, and without a side channel, each gateway can periodically append a heartbeat message to the message log, indicating that currently no updates are to be expected. As gateway group members need to be updated to the latest version to append to the message log, they will detect missing heartbeat messages, which indicates either a gateway failure or a withhold attack. This approach's overhead can be parameterized by adjusting the heartbeat frequency. Furthermore, its storage overhead can be limited by trimming older heartbeats (cf. Section 5.3.2.5).

Further Security Considerations. When adding gateways to a gateway group, the cloud provider might withhold or modify join requests. The user will immediately notice such attacks when reviewing join requests (cf. Section 5.3.2.3). When trimming the message log, the snapshot stored in the cloud cannot be modified as the hash value cryptographically binds the snapshot to the message log (cf. Section 5.3.2.5). Although not specifically designed to protect against insider attacks, D-CAM provides a tamper-resistant log of all control operations. Thus, we can detect misbehavior (e.g., device defects or attacks) and blame the originating gateway. As a consequence, the misbehaving gateway can, e.g., be expelled from the group.

To conclude, D-CAM's approach of a cryptographically protected message log offers protection against the identified attacks, even against powerful adversaries such as a malicious-but-cautious cloud provider. Attack attempts are detected by D-CAM, which prevents, e.g., physical harm. Hence, users can launch countermeasures and collect evidence of attacks. In the following, we show that this strong level of protection comes at modest costs in terms of processing and storage overheads.

5.3.4 Evaluation

To prove the feasibility of D-CAM and quantify its performance, we evaluate its processing, storage, and communication overheads. Based on these results, we compare D-CAM to other remote management approaches such as VPNs or SSH and discuss D-CAM's performance as well as scalability. As a basis for our evaluation, we implemented a prototype for the gateway in the C programming language.

We rely on OpenSSL 1.0.1k for the cryptographic operations, libjansson 2.7 for serializing messages using JSON, and MySQL 5.5 for persistently storing state at the gateways, e.g., the list of gateways in the gateway group. As an exemplary embedded device for the gateway, we chose the Raspberry Pi Model B+ with a 700 MHz ARM11 processor, 512 MB of RAM, and Raspbian Jessie Linux as the operating system.

To properly select the employed cryptographic primitives, we followed the recommendations of NIST [Bar15]. More precisely, we use SHA-256 as hash function and two different digital signature schemes with the same security level (to enable their comparison): RSA with 2048 bit keys and ECDSA with the NIST curve P-256.

5.3.4.1 Processing Overhead

First, we evaluate the processing overhead for appending messages to and verifying messages in the message log. We refer to a *signing interval* of k if a gateway signs on average each k-th message (cf. optimization in Section 5.3.2.2). Analogously, a *verification interval* of k means that a gateway on average checks the digital signature of each k-th message (cf. optimization in Section 5.3.2.4). For each result, we perform 30 runs, each consisting of the processing, i.e., appending or verifying, of 10 000 control messages. In the following, we show the mean processing time for one message with 99 % confidence intervals. We distinguish between the time required for creating, respectively verifying the hash chain and the digital signature, including parsing and serializing messages, and the lookup of public keys.

Appending to the Message Log

The processing time for a gateway to append one control message to D-CAM's message log is influenced by the signing interval and the message length.

First, we vary the signing interval between 1 and 25 and fix the message length to 2500 byte, which allows to encode even larger control messages. Our results in Figure 5.12 (note the logarithmic scale) show that the processing time for creating the hash chain does not depend on the signing interval while the time for creating the digital signatures considerably decreases for an increasing signing interval. Especially for smaller signing intervals, we see that ECDSA strongly outperforms RSA as expected due to their different performance asymmetries. For a signing interval of 20, using RSA allows a gateway to append 202 messages/s compared to

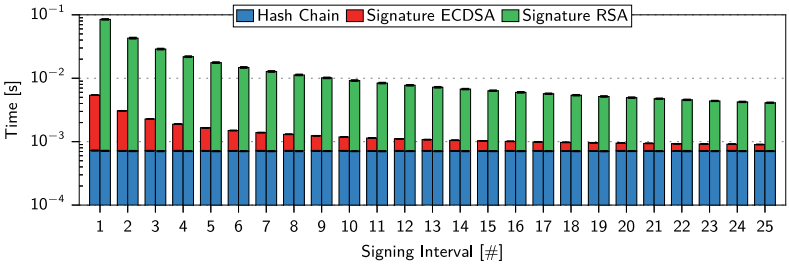

Figure 5.12 The mean processing time for appending a message of length 2500 byte to the message log depends on the signing interval. Increasing the signing interval reduces the average time spend in the predominant signing operation.

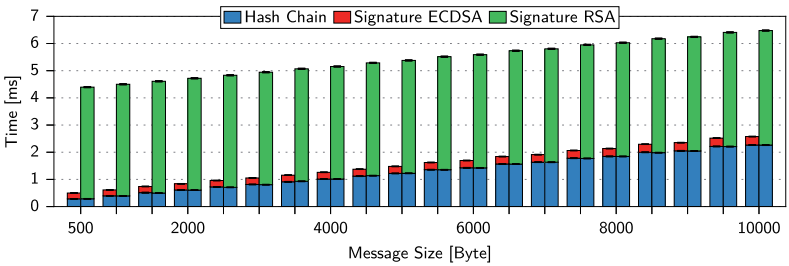

Figure 5.13 The mean processing time for appending a message of varying payload size to the message log with a signing interval of 20 and a group size of 1 increases roughly linearly with the payload size.

1052 messages/s for ECDSA. Furthermore, we observe only little additional savings for increasing the signing interval beyond 20, especially when using ECDSA as signature scheme.

Hence, we now fix the signing interval to 20 and vary the message length between 500 and 10 000 byte in steps of 500 byte. In Figure 5.13, we observe that the processing time increases roughly linearly with increasing message sizes. This is mainly due to an increased time for creating the checksum for longer messages. Again, we observe a superior performance of ECDSA compared to RSA as expected. For a message size of 500 byte, we can process 228 messages/s with RSA compared to 2004 messages/s with ECDSA. This decreases to 154 messages/s using RSA and to 388 messages/s using ECDSA for a larger message size of 10 000 byte.

Verifying the Message Log

The processing time for verifying a control message in D-CAM's message log depends on the verification interval and the message size. Additionally, the processing time might be influenced by the number of gateways from which the control messages in the message log originate, as each gateway uses a distinct public key.

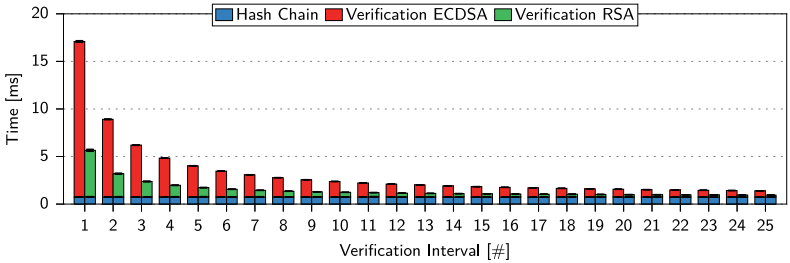

Figure 5.14 The mean processing time for verifying one control message of size 2500 byte in the message log depends on the verification interval. By increasing this interval, the average time spent for the predominant operation for verifying the digital signature is reduced.

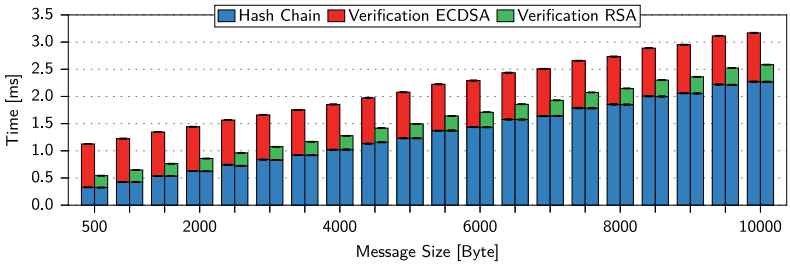

Figure 5.15 The mean processing time required for verifying a control message with a verification interval of 20 and a group size of 1 scales roughly linearly with the payload size. This is mainly due to the effort required for verifying the correctness of the hash chain.

To study the influence of the verification interval on the processing time, we vary the verification interval between 1 and 25 and fix all other parameters. More specifically, we fix the message size to 2500 byte and the gateway group size to 1. As shown in Figure 5.14, the processing time required for verifying the hash chain does not depend on the verification interval while the processing time required for verifying digital signatures decreases with an increasing verification interval as expected. Here, RSA benefits from the performance asymmetry and outperforms ECDSA. For a verification interval of 20, RSA enables us to verify 1007 messages/s compared to only 641 messages/s for ECDSA. Increasing the verification interval beyond 20 offers only little additional performance gains.

Thus, we now set the verification interval to 20 while keeping the group size at 1 and evaluate the impact of varying the message size between 500 and 10 000 byte in steps of 500 byte. We depict the resulting processing time for verifying control messages in Figure 5.15. The processing time required for verifying one control message increases approximately linearly with an increasing message size. This stems from an increase in the processing time required for verifying the hash chain checksums and validating the digital signature. We again notice a superior performance of RSA over ECDSA. Using RSA allows a gateway to verify 1840 messages/s compared to

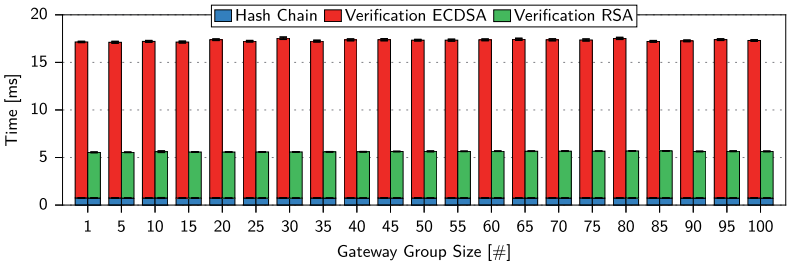

Figure 5.16 The number of gateways that append control messages to the message log has a negligible impact on the mean processing time required for verifying one message of length 2500 byte in the message log with a verification interval of 20.

888 messages/s using ECDSA for control messages with a size of 500 byte. For a control message size of 10 000 byte, these numbers decrease to 387 messages/s for RSA and 315 messages/s for ECDSA, respectively.

Next, we analyze the impact of the number of gateways from which the control messages in the message log originate on the processing time required for verifying control messages. We fix the verification interval at 20, the message size to 2500 byte, and increase the number of gateways that append control messages to the message log from 1 to 100. In this setting, gateways append messages on a rotating basis, i.e., the first gateway will append its second message only after all other gateways have appended a message. Our results in Figure 5.16 show that the verification time does only negligibly depend on the number of gateways from which the control messages in the message log originate. More specifically, ECDSA is able to verify 58 messages/s both for 1 gateway and 100 gateways, while the processing time for verifying control messages with RSA shows a subtile decrease from 180 messages/s for 1 gateway to 177 messages/s for 100 gateways.

Remarks

Setting both signing and verification interval to 20 constitutes a reasonable trade-off between processing time and required buffer space for verification. Furthermore, if the goal is to optimize performance of appending messages in D-CAM, ECDSA is preferable over RSA. However, RSA shows a superior performance for verifying messages. Here, it is important to note that for a gateway group of size n, a control message has to be verified by n gateways while it is appended only once. Thus, especially for larger gateway groups, selecting RSA as digital signature scheme is recommended. Notably, the number of gateways from which the control messages in the message log originate does not perceivably influence the processing time required for verifying messages. This behavior is expected as long as we can keep the public keys of all these gateways in memory. Even on a resource-constrained Raspberry Pi, we can easily cache the public keys of hundreds of gateways.

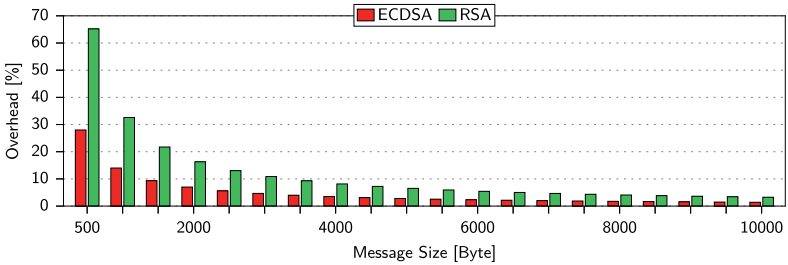

Figure 5.17 The relative per-message storage and communication overhead of D-CAM reduces with increasing message size.

5.3.4.2 Storage and Communication Overhead

To analyze the storage and communication overhead of D-CAM as well as the influence of trimming the message log, we rely on analytical methods and simulations.

Per-Message Storage and Communication Overhead

The per-message storage and communication overhead of D-CAM stems from the space required for encoding header fields (e.g., sequence number and gateway identifier), the checksum that realizes the hash chain, and the digital signature. More precisely, for our choice of cryptographic primitives, the storage and communication overhead of D-CAM consists of 36 byte for encoding the header, 32 byte for the checksum, plus 258 byte for encoding an RSA digital signature respectively 72 byte for encoding an ECDSA digital signature. We show the resulting storage overhead for increasing message sizes in Figure 5.17. As the sizes of the header, checksum, and digital signature stay constant for varying message sizes, this overhead decreases from 65.2 % for messages of size 500 byte to 3.3 % for messages of size 10 000 byte when using RSA and from 28 % for messages of size 500 byte to 1.4 % for messages of size 10 000 byte when using ECDSA.

Influence of Trimming the Message Log

The behavior of D-CAM's trimming approach depends on the number of obsolete messages in the message log. We study this behavior with a simulation approach where we consider control message logs of size up to 100 000 messages and let D-CAM trim the message log whenever it observes at least 5000 obsolete messages (the specific amount is one of D-CAM's parameters). We iteratively append messages, where each inserted message may obsolete a previous one with a probability of $p = 0, 0.2, ..., 1$. In Figure 5.18, we compare the number of messages a joining gateway has to process to the optimal number, i.e., only non-obsoleted messages. Each experiment was conducted 1000 times with real random seeds [Wal96] and we depict the mean amount of messages that have to be verified by the joining gateway.

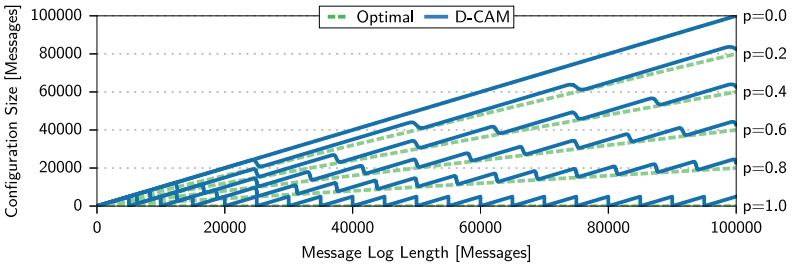

Figure 5.18 The influence of trimming the message log depends on the probability of messages being obsoleted (p = 0, 0.2, ..., 1). D-CAM at most incurs a fixed overhead, whose precise value is a configurable parameter of D-CAM (here: trimming after 5000 obsolete messages).

We omit confidence intervals to ease readability, as the 99 % confidence intervals for the mean amount of messages that have to be verified are below 204 messages for all values. Indeed, our results show that D-CAM at most incurs a fixed overhead of 5000 messages. Furthermore, the number of trimming operations required (indicated by the drops of the blue line in Figure 5.18) directly scales with the probability of obsolete messages, ranging from 20 when messages are always directly superseded ($p = 1$) to 0 if new messages never replace old messages ($p = 0$).

Remarks

Our evaluation of D-CAM's storage and communication overhead leads to two observations. First, if reducing the storage and communication overhead of D-CAM is the primary optimization goal, using ECDSA as signature scheme is the preferable choice. Notably, the resulting storage and communication overhead can further be reduced by increasing the signing interval (cf. Section 5.3.2.2). Second, when considering the amount of messages that need to be processed by a new gateway joining a gateway group, D-CAM's optimization to periodically trim obsolete messages results in at most a constant overhead compared to an optimal solution that directly deletes any obsoleted messages.

5.3.4.3 Comparison to Remote Management Approaches

Although D-CAM provides more functionality, e.g., group management and a verifiable audit log, than established remote management approaches such as VPNs or SSH, it is still interesting to see how D-CAM performs compared to said approaches. As our goal is to achieve a *consistent* configuration of the whole federated IoT network, a comparable solution based on VPNs or SSH requires one connection from each gateway to each other gateway to reliably and consistently communicate all control operations. In a network of N gateways, this results in sending N messages for each control operation and adding as well as maintaining N new connections for each new gateway. Considering bandwidth constraints of gateways, e.g., mobile

uplinks, this becomes infeasible already for small networks. Contrarily, D-CAM only sends one message per control operation from a gateway to the cloud, irrespective of the network size. Thus, D-CAM's scalability is not bound by bandwidth. Furthermore, D-CAM's design reduces setup and management costs and is less susceptible to misconfiguration.

To quantitatively compare D-CAM to VPNs and SSH, we perform additional measurements using our evaluation setup. To this end, we use OpenVPN 2.3.4 (for comparison with VPNs) as well as OpenSSH 6.7 (for comparison with SSH) both with RSA 2048 bit keys and AES-256 in CBC mode for encryption. These parameter choices provide the same security level as D-CAM. For a gateway group size of N, the transmission of a message of size 2500 byte results in $2925 \times N$ byte application layer payload for OpenVPN and $2766 \times N$ byte application layer payload for OpenSSH. This stands in stark contrast to an application layer payload of 2826 byte irrespective of the gateway group size for the transmission of a message of size 2500 byte when using D-CAM.

Hence, already for federated IoT networks of 3 gateways, D-CAM reduces the communication overhead compared to utilizing VPNs or SSH. We observe similar trends for the processing time required for creating and verifying control messages (for a signing respectively verification interval of 20 in D-CAM).

5.3.4.4 Concluding Observations

We specifically designed D-CAM to scale to large federated networks. Our evaluation results confirm that the processing time for appending to the message log as well as the processing time for verifying individual messages in D-CAM's message log are not noticeably impacted by the size of the gateway group, i.e., the number of gateways participating in a federated IoT network. Likewise, neither storage nor communication overhead of D-CAM depend on the gateway group size. D-CAM scales linearly in the size of the message log, being bound only by the amount of available storage space.

Our message log trimming approach further helps in reducing the required storage space and the verification time for gateways joining an already established IoT network. Additionally, D-CAM does not constitute a trade-off between security and performance. We provide the same level of security as digital signatures and additionally protect against modification, insertion, reordering, and withholding of control messages.

Increasing D-CAM's signing and verification intervals allows us to reduce the processing overhead when creating respectively processing individual control messages. The trade-off here is that messages must be buffered at a receiving gateway before they can be verified. Furthermore, in the unlikely event of signature mismatches, D-CAM might have to drop more control messages than actually necessary.

To conclude, D-CAM provides a high level of security against powerful adversaries such as a malicious-but-cautious cloud provider at reasonable costs with respect to processing and storage overhead.

5.3.5 Achieving Message Confidentiality

So far, we have concentrated on achieving integrity and authenticity in D-CAM. However, certain scenarios also require the confidentiality of control messages to achieve privacy as the information on the configuration, authorization, and management of IoT devices and networks may in itself contain private information.

For example, such control messages could reveal that a company operates specific equipment and its precise configuration, thus providing competitors with a strategic advantage. In the private setting, control messages show which medical sensors a user operates, thus hinting at certain medical conditions. To preserve users' and corporations' privacy when using the cloud to control and federate their IoT network, we thus encrypt all control messages in D-CAM to only allow authorized gateways within the federated IoT network to access their content. Similar to the trust point-based security architecture underlying SCSlib (cf. Section 5.2), we efficiently encrypt control messages using a symmetric group key (e.g., using AES-256) that is shared among the members of a gateway group.

However, users can add or remove gateways arbitrarily. This flexibility renders the distribution of the group key challenging, as a gateway must only be able to read control messages in the message log that were appended during the time span of the gateway's membership in the corresponding gateway group. To achieve this goal, we change and redistribute the group key whenever changes to the group membership occur, i.e., gateways are added or removed. Furthermore and similar to SCSlib, we periodically exchange the group key to strengthen security (cf. Section 5.2.2.3).

For distributing the group key, we rely on the public keys of the gateways in the gateway group as these are known to all other members of a gateway group by design. Each time a gateway appends a control message to add or remove a gateway from the gateway group, it also has to change the group key. To this end, the gateway encrypts the group key for each gateway that (still) is a member of the gateway group after the addition or removal operation by using the respective public keys. To distribute these keys, the gateway that initiated the operation leading to the key exchange then appends the encrypted group keys for each gateway to the message log. Thus, only the current gateway group members can decrypt the new group key and as a result the following messages in the message log. As gateways join or leave a gateway group rather sporadically and periodic key exchanges happen in larger time intervals, this introduces only a modest overhead (cf. Section 5.2.4) that is worth the additional protection of the confidentiality of control messages and hence users' privacy with respect to the configuration, authorization, and management of their IoT devices and networks.

5.3.6 Summary and Future Work

When steering the configuration, authorization, and management of federated IoT networks from the cloud, severe privacy, security, and safety concerns arise. To overcome these concerns, we presented D-CAM to realize *distributed* configuration,

authorization, and management in the cloud-based IoT *across* network borders. D-CAM runs directly on the user-controlled gateways in a federated IoT network and allows users to control their *complete* federated IoT network from each of their gateways without having to care about the reachability and availability of individual devices. To this end, D-CAM utilizes the concepts of hash chains and digital signatures to create a secure and distributed administrated log of control messages stored in the cloud. We deliberately restrict the cloud to act as a highly available and scalable proxy for relaying and storing secured control messages. This allows us to ensure the integrity, authenticity, and confidentiality of control messages, even in the presence of a powerful attacker such as a malicious-but-cautious cloud provider. D-CAM's tamper-resistant log of all control operations additionally allows to detect and pinpoint internal attackers. Thus and in contrast to related work, D-CAM is especially well-suited for controlling access to actuating capabilities of safety-critical devices as they are prevalent in today's IoT deployments.

As our evaluation results show, D-CAM's high level of security, especially against powerful adversaries such as a malicious-but-cautious cloud provider comes at modest costs. Even on a resource-constrained gateway (such as a Raspberry Pi), D-CAM is able to process more than 640 messages per second for a reasonable choice of system parameters. Notably, D-CAM's processing overhead depends only on the number of control messages to be processed and does not per se increase with the number of gateways in the gateway group. Furthermore, D-CAM's message log trimming scheme results in at most a fixed storage overhead compared to a system realizing configuration, authorization, and management centralized in the cloud without the extra level of security provided by D-CAM. Compared to other remote management approaches (e.g., VPNs and SSH), D-CAM does not only show comparable performance for small networks but considerably scales better for larger networks.

We are convinced that the benefits of D-CAM can be valuable also beyond securing configuration, authorization, and management in the cloud-based IoT. To this end, promising future work would be concerned with deploying and adapting D-CAM for other application domains. In the context of software-defined networking (SDN), D-CAM could be evolved to handle the distribution of SDN rules, e.g., expressed using OpenFlow [MAB+08], to bridge isolated individual networks over untrusted communication infrastructure such as the Internet to create a federated SDN-enabled network. In this setting, integrating the different architectural components of SDN with their different roles and varying rights into D-CAM might prove challenging, especially when multiple SDN controllers need to be synchronized in an extremely timely fashion [TGG+12].

Indeed, further research is required to enhance D-CAM such that it is able to reach the controller responsiveness requirements, e.g., a controller response time in the order of 100 ms, as they are prevalent in SDN deployments today [TGG+12]. Likewise, D-CAM could be applied to ease the configuration, authorization, and management of devices in community networks [BBB+13]. A community network is a distributed and decentralized system that typically operates at comparable large scales to deliver a wide range of applications and services, most importantly Internet access [BBB+13]. Examples include the Freifunk movement in Germany or the Guifi.net network in Spain. As community networks constitute a less trustworthy environment

than the federation of the IoT networks of one user, D-CAM needs to be enhanced with consensus protocols comparable to Bitcoin's proof-of-work [Nak08, CD16] to be applicable in this scenario.

Finally, by coupling D-CAM even tighter with the concept of blockchains and especially smart contracts, we could realize deployment scenarios where access to IoT devices or the data they produce is automatically granted to anyone who pays a certain user-defined fee, e.g., using a micropayment scheme, or is dependent upon sufficient anonymization schemes such as k-anonymity or differential privacy [MMZ+17, SBHD17]. Here, the main challenge lies in technically ensuring that access to IoT devices and their data is indeed only granted if the user-imposed conditions for this access have been met.

In conclusion, D-CAM allows users to securely realize distributed configuration, authorization, and management in cloud-connected IoT networks even in the presence of powerful attackers at modest costs in terms of processing and storage overhead. By doing so, we enable users to conveniently and reliably interconnect their previously isolated IoT networks without raising privacy, security, and safety concerns that otherwise would prevent the federation of IoT networks based on the cloud as a highly available and scalable underlying infrastructure.

5.4 Conclusion

While cloud computing is a promising solution for handling the growing demand for storing and processing large amounts of data collected by an increasing number of IoT deployments, integrating the IoT with cloud computing raises severe privacy concerns (cf. Section 2.4.2). When realizing cloud services for the IoT, the providers of these services are in a diametral position as they do not control the underlying cloud infrastructure but still have to account for the privacy of their users. To support cloud service providers in developing and deploying cloud services in a privacy-preserving manner, we proposed two approaches, (i) to transparently realize the protection of IoT data stored in the cloud and (ii) to secure the configuration, authorization, and management of IoT devices and networks in the cloud.

To unburden service developers from having to implement the necessary security functionality and hence enable domain specialists who are not security experts to realize privacy-preserving cloud services, we introduced SCSlib, a security library that transparently handles the security functionality required for accessing protected IoT data in the cloud. SCSlib is based on our trust point-based security architecture for IoT data in the cloud [HHCW12, HHM+13, HHMW14] that essentially realizes a user-centric and cryptographically enforced access control system. To this end, SCSlib relies on a widely applicable, standards-based approach to represent and protect IoT data in the cloud and, as a result, can support different performance and security requirements. Our evaluation performed on public cloud infrastructure confirmed the feasibility of abstracting from processing protected IoT data in cloud services. Notably, SCSlib's caching scheme clearly improves processing times for sequential and random access to IoT data in the cloud.

Moving onwards from solely protecting the access to IoT *data*, we presented D-CAM, a distributed architecture that enables users to additionally secure the *configuration, authorization, and management* of their IoT devices and networks across network borders. To this end, D-CAM effectively ensures that only authorized parties can issue and access configuration commands. Notably, D-CAM limits the cloud to act as a highly available and scalable storage for control messages and thus realizes reliable and secure network control across IoT networks. D-CAM provides strong security guarantees such that even a dishonest cloud provider cannot control IoT devices without permission of the owner of these devices. In our evaluation of D-CAM, we have seen that the introduced processing, storage, and communication overheads are reasonable and worth the additional level of protection. Furthermore, our evaluation results indicate that D-CAM can easily scale to secure large federated IoT networks. To also protect private information potentially contained in configuration, authorization, and management messages, D-CAM additionally supports a mechanism to protect the confidentiality of these messages.

In this chapter, we mainly addressed the research question on how *service providers* can build privacy-preserving cloud services on top of cloud infrastructure. Consequently, our contributions in this chapter primarily tackle the core problem of users' missing control. We tackle this problem by cryptographically protecting the access to IoT data as well as the configuration of IoT devices and networks. Through these efforts, we additionally provide users with transparency over who has access to their data and who can control their IoT devices and networks. Furthermore, by unifying interfaces and hence realizing interoperability with different cloud services, our contributions pave the way towards breaking up the inherently centralized cloud computing landscape.

The results presented in this chapter highlight the importance of addressing the role of cloud service providers and developers to protect users' privacy when using cloud-based services. By integrating our contributions presented in this chapter with data handling requirements-aware cloud infrastructure as proposed in Chapter 4, we can further increase the level of privacy offered to users, e.g., by allowing them to specify requirements such as the security level of SCSlib's cryptographic primitives. Furthermore, the concepts underlying SCSlib and D-CAM can serve as an important foundation for realizing cloud services in a fully decentralized peer-to-peer system of trusted resources as presented in the subsequent chapter. Here, SCSlib can be adapted to afford confidentiality of data at rest and during transport. Likewise, D-CAM could be applied to secure the management of resources in such a decentralized setting.

6

Decentralizing Individual Cloud Services

So far, we focused on approaches where different actors in the cloud computing landscape cooperate to provide privacy. This cooperation, however, requires users to put a certain level of trust into infrastructure and service providers which might not always be justified. In this chapter, we explore how a decentralized deployment model for a certain class of cloud services which do not require massive scalability can enable users to completely refrain from using cloud services by cooperating with other users. To this end, we first motivate our idea of decentralizing individual cloud services (Section 6.1). We then present PriverCloud [Hil14,HHHW16], a secure peer-to-peer cloud platform based on social trust. PriverCloud builds on top of devices operated by users' close friends and family to realize a trusted, secure, and decentralized execution environment for individual cloud services (Section 6.2). Finally, we conclude this chapter with a discussion and summary of our results (Section 6.3).

6.1 Motivation

One of the fundamental challenges with respect to privacy in cloud computing is the centrality of the cloud computing market (cf. Section 1.1.3). This centralization is inherent to the current deployment model of cloud services, where cloud services are realized on top of cloud infrastructure operated by a small number of providers that jointly dominate the market (cf. Section 1.1.3). Some of these challenges result from the key characteristics of cloud computing (cf. Section 2.1.1), e.g., infrastructure providers have to rely on a large amount of computing and storage resources which require huge upfront investments to provide rapid elasticity as well as failover and resilience. Thus, the cloud computing landscape naturally evolves around a comparably small number of players.

Yet, we observe that not all types of cloud services necessarily require the massive scalability promised by cloud computing. This includes individual services, i.e., cloud services where users interact only with their own data, such as calendar and contact synchronization, which often do not require the full massive scalability offered by cloud services. Hence, for this class of cloud services, it would be sufficient to deliver the remaining advantages of the cloud computing paradigm such as availability and reliability. As a result, we could break up the centrality of cloud computing and empower users with exceptionally strong privacy expectations and mistrust into cloud providers to completely refrain from using cloud services and still benefit from selected advantages realized by the cloud computing paradigm.

State-of-the-art approaches to overcome the centrality of public cloud services can be classified into two categories. First, approaches that shift cloud services to devices controlled by an individual user such as ownCloud [Own18] or Seafile [Sea18] typically trade-in availability and scalability for increased privacy. This is mainly due to the use of only a single or very few devices, often hosted at the user's home and connected only via one residential access line to the Internet. Second, when solely considering the confidential storage of data in the cloud, the centrality of cloud computing can partly be countered using encryption [Box18] or splitting of data between different cloud providers [BKTM11, JZV+12]. However, in such a setting, data is merely stored in the cloud. Decryption and any processing have to happen on the users' devices without the possibility to benefit from the scalable resources of the cloud. Still, even in this restricted scenario, the cloud provider can derive valuable meta information, e.g., time and location of data access. Hence, state-of-the-art approaches (partly) break up the centrality of cloud computing and put users back in control over their data at the cost of diminishing the benefits of cloud computing to a large extent. Thus, the question of how we can realize individual cloud services in a decentralized manner without having to give up the advantages of cloud computing is an open and pressing challenge.

6.1.1 Contributions

To address the challenge of overcoming the centrality of cloud computing, we propose to decentralize individual cloud services to allow users to protect their privacy and still benefit from the advantages of cloud computing. More specifically, in this chapter, we present *PriverCloud*, a secure peer-to-peer cloud platform that utilizes idle resources of devices of friends and family to realize a trusted, decentralized system in which cloud services can be operated securely and privacy-preserving. Notably, our approach solely relies on cooperation between users and hence eliminates any trust assumptions for service or infrastructure providers. Furthermore, to alleviate trust assumptions between different users, PriverCloud optionally supports the use of trusted platform modules (TPMs) to technically guarantee the privacy of user data. To ease the migration from public cloud services, PriverCloud affords for the execution of existing cloud services developed for Google App Engine [Goo18a]. As our evaluation shows, PriverCloud achieves high availability by securely distributing data storage over trustworthy devices as well as by monitoring the reachability of cloud services and automatically recovering from any detected failures.

6.2 PriverCloud: A Secure Peer-to-Peer Cloud Platform

Despite the privacy challenges resulting from cloud computing, cloud services provide very desirable features that cannot be neglected. Specifically, they offer high availability, easy accessibility, extreme scalability, and simple deployment. Most notably, cloud services provide a high ease of use due to their integration into many devices and applications, e.g., smartphones and web browsers. Still, besides all our efforts for cooperative approaches to privacy in cloud computing (cf. Chapters 3 to 5), the privacy expectations and mistrust into cloud providers of some users might be so strong that they decide not to use any cloud services at all. In this setting, the question of how we can provide these users with (a subset of) the advantages of cloud computing—at least for certain types of cloud services—naturally arises.

Hence, to realize privacy-sensitive cloud services and keep the advantages of cloud computing, we propose an architecture called PriverCloud. We motivate our approach based on two core observations: (i) moving away from the centrality of cloud computing is key to account for exceptionally strong privacy requirements and (ii) users posses unused processing resources in their home networks (e.g., home routers or network attached storage and set-top boxes) that become increasingly more powerful (e.g., modern home routers have multiple CPU cores and 1 GB of RAM). Hence, we advocate for moving privacy-sensitive services from public clouds to an individual PriverCloud for each user which consists solely of trusted infrastructure contributed by close friends and family. This approach allows us to break up the inherent centrality of cloud computing but still leverage decentralized resources to realize most of its prominent features.

To turn this vision into an actually deployable technical system, we identify the following challenges: (i) coping with the inherent resource constraints with respect to processing, storage, and networking of devices typically available in home networks, (ii) achieving the advantages of cloud computing in a highly decentralized system built on heterogeneous devices, (iii) extending trust from individual device owners to a whole PriverCloud deployment, and (iv) achieving deployability, most importantly by easing the migration from public cloud services to a PriverCloud deployment.

In the following, we discuss how to solve these technical challenges for PriverCloud deployments spanning over resource-constrained devices in home networks. We substantiate the feasibility of our proposed approach by evaluating the performance of our implementation of PriverCloud. Our results show that PriverCloud can be deployed to devices with constrained resources with modest overhead introduced by our security measures. Furthermore, PriverCloud reliably detects and recovers from failures of devices in the order of seconds.

6.2.1 Problem Analysis and Trust Model

The motivation of users to refrain from using cloud services mainly results from the inherent centrality of cloud computing and the resulting loss of control of users

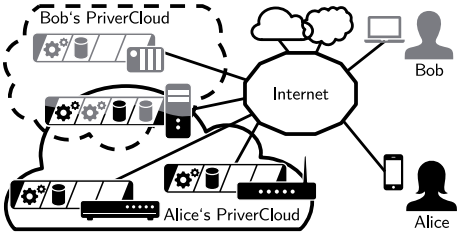

Figure 6.1 In our envisioned scenario, each user builds her individual PriverCloud instance over trusted devices contributed by friends and family. A user can use processing (⚙) and storage (🛢) resources on these devices to realize Internet-accessible privacy-sensitive cloud services.

over their data. This loss of control is mainly due to three threats. First, the cloud provider (or one of its employees) might be interested in the data and access it without authorization [PCB15]. Second, certain countries access and intercept data within their legislation for safety, security, economic, or scientific purposes [Gel13, PP15]. Finally, it is common for cloud service providers to subcontract other cloud providers [PP15], e.g., to mitigate load peaks, as demonstrated to the user by our awareness approaches (cf. Chapter 3). Hence, the previous two threats amplify significantly, as the user does not only have to trust one cloud provider (and the responsible jurisdiction) but a potentially unknown number of additional cloud providers and the jurisdictions they operate in (cf. Section 1.1.3).

To overcome these severe threats to privacy, it is thus inevitable to break up their two root causes: centrality and loss of control. We do so by introducing our PriverCloud architecture with individual instances that run only on devices a specific user explicitly trusts and are deployed in a location with acceptable legislation. Thus, our approach allows for a new calibration of the trade-off between privacy and advantages of cloud computing such as availability and accessibility. This stands in stark contrast to today's approaches for strictly preserving privacy for cloud services which come at the cost of diminishing many benefits of the cloud computing paradigm. In the following, we first discuss our underlying scenario and trust assumptions. From this scenario, we derive the challenges that any approach to decentralizing individual cloud services needs to address before we discuss and analyze related work.

6.2.1.1 Scenario

We present our envisioned scenario in Figure 6.1 by exemplarily focusing on the viewpoint of the user Alice and her PriverCloud instance. Before Alice can start using PriverCloud, she first has to gain access to infrastructure that she trusts and that can provide her with the required processing and storage resources. For this purpose, we envision to leverage the idle resources on devices of close friends and family. These devices range from less powerful, embedded devices (e.g., Raspberry Pis or NAS and set-top boxes) to more powerful devices such as desktop computers. Typically, these devices are located within home networks and connected to the Internet using residential access lines and as such suffer from connectivity disruptions.

Once Alice has built-up her PriverCloud instance, she can begin to run cloud services on it. We specifically target cloud services that are especially susceptible to privacy threats. Here, our focus lies on individual cloud services, i.e., services targeting a small closed target audience (e.g., only Alice herself or selected friends). These applications can range from a calendar service offering synchronization, scheduling, and notifications up to a fully-fledged document storage service able to store several GBs of data and offering functionality such as file sharing, image editing, or multimedia streaming. In contrast, cloud services which can be accessed by anyone are in our opinion better off with public cloud services as their information is publicly available anyways. To operate a cloud service, Alice selects a service from a service marketplace, similar to those available for smartphones (of course, Alice can also develop her own custom cloud service). The cloud service is then deployed on one or multiple of the devices in Alice's PriverCloud instance. Should the cloud service require persistent storage of data, this data is distributed to the available storage provided by these devices.

As with public cloud services, Alice should be able to access her services independent of her location via the Internet at any time. She should neither have to care about the actual device a specific service is running on nor which device stores her data. Notably, no modifications should be required on the client side to allow Alice to continue to use her web browser or other applications (e.g., an app on her smartphone) to access the services deployed in her PriverCloud instance as with today's cloud services. In our approach, each user has her own PriverCloud instance spanning over resources she trusts, e.g., provided by friends and family. However, as we utilize resources based on social relationships, the PriverCloud instances of different users are likely to overlap (gray/black device in Figure 6.1). In this example, Alice and Bob trust the same device and hence can both utilize its resources. Importantly, this does not imply that Alice and Bob have to trust each other. In the following, we discuss the trust assumptions in our scenario in more detail.

6.2.1.2 Trust Assumptions

In traditional cloud deployments, we differentiate between different actors that provide the necessary processing and storage infrastructure, offer services on top of this infrastructure, and consume these services (cf. Section 2.1.3). Contrary, in our envisioned scenario underlying PriverCloud, all these tasks have to be performed by the participants of the peer-to-peer system themselves. To ease presentation in the following, we refer to participants who make their storage and processing resources available to other participants as *resource providers* and denote those participants that consume resources to operate their services as *users*. Typically, participants in PriverCloud will take both roles, i.e., act as a resource provider for other users and at the same time use resources offered by other resource providers.

As a foundation for our design of PriverCloud, we first discuss our trust assumptions for the underlying scenario as illustrated in Figure 6.2. Most importantly, we assume a scenario that leverages social trust, i.e., that users trust resource providers. Consequently, it is safe to assume that resource providers in general refrain from accessing

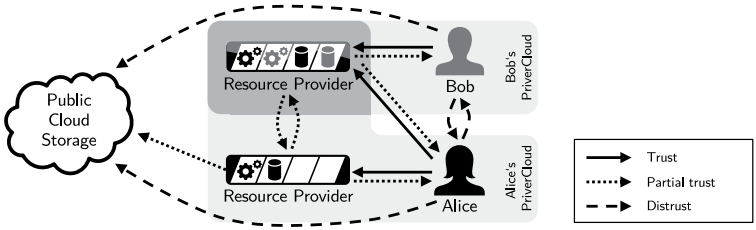

Figure 6.2 Our underlying scenario considers different levels of trust between the different users, resource providers, and optional public cloud storage involved with PriverCloud.

potentially sensitive data of other users or tamper with services deployed by users on their resources. In Section 6.2.2.3, we additionally provide technical measures based on TPMs that further strengthen users' trust into resource providers.

As shown in Figure 6.2, resource providers typically offer their resources to more than one user. Here, our assumption is that these users do not necessarily trust each other. This is a reasonable assumption since often users do not even know which other users rely on the same resource provider. Furthermore, resource providers might themselves use cloud services to increase their available storage space and hence also use this additional cloud storage to provide resources to other users. Naturally, users do not trust the providers of these cloud services.

When considering the role of resource providers, we assume that resource providers partially trust the users of their resources. More specifically, resource providers need to trust users to behave responsibly with respect to the provided resources, e.g., only requesting the amount of resources they actually need and not wasting resources. When resource providers leverage cloud services to increase their storage space, they trust the providers of these cloud services to honestly store the data. However, resource providers do not trust providers of cloud services to respect the confidentiality of outsourced data. Finally, neither users nor resource providers trust any entities on the network path to other participants of a PriverCloud instance. Most notably, these entities include networks operators and ISPs.

6.2.1.3 Challenges

Based on our envisioned scenario and the above trust assumptions, we identify the following four main challenges any approach to decentralizing individual cloud services needs to address.

Respecting Resource Constraints: As we target devices in home networks, we have to cope with limited storage and processing resources as well as limiting network conditions. Additionally, as we envision to utilize a wide range of different devices, we have to account for heterogeneity of resources. When considering *storage resources*, a home router might provide up to few GB of storage space, while a NAS box can supply up to a few TB of disk space. Since these resources need to be

shared with other users, restrictions and quotas apply (e.g., 100 MB of storage per user VM in the Seattle testbed [CBKA09]). A similar situation applies to *processing resources*. Cloud services, formerly executed on powerful server CPUs provided by public cloud infrastructures, have to be operated on comparatively limited CPUs provided by desktop PCs or even embedded devices. Furthermore, when decentralizing individual cloud services, we face *network conditions* of residential access links which provide limited availability and capacity. Specifically, devices connected via residential access lines might not be always connected to the Internet and the available bandwidths are typically orders of magnitude smaller than those of data centers. Further complicating this issue is the asymmetry in bandwidth home networks often suffer from, i.e., a higher ratio of downlink than uplink. Especially the limited uplink makes the operation of bandwidth-hungry services challenging.

Preserving Cloud Advantages: Preserving advantages of cloud computing when decentralizing individual cloud services is a challenging task. Specifically, from a usability perspective, a user should not even notice that she is not using traditional cloud services (although the usage of decentralized individual cloud services should be evident for transparency purposes). First of all, the *availability* of data and services in public clouds has to be achieved using decentralized devices with residential access links as sole connection to the Internet. Although the distributed nature of our envisioned deployment scenario makes this challenging, it also opens up new opportunities. In contrast to public clouds, decentralized individual cloud services are not susceptible to outages of complete data centers [KKLL09]. Similar to availability, the *accessibility* of data and services should not be harmed compared to public clouds. Most importantly, decentralized individual cloud services should be accessible from any device and anywhere, just as public cloud services. Hence, clients (e.g., smartphone apps or web browsers) should not need to implement application logic or decryption operations. Additionally, users should be able to transparently access their services without having to care about on which resources these are currently deployed. From another perspective, decentralized individual cloud services should provide *scalability* with respect to a service's varying processing and storage demands at least to a certain extent (as required by individual services).

Extending Trust: Our envisioned scenario for decentralizing individual cloud services builds on social trust. However, we have to provide measures to extend this initial trust in individual persons to the whole system. First of all, decentralized individual cloud services span over the untrustworthy Internet (cf. Figure 6.1) and hence are susceptible to several attack vectors. Secondly, not only do users have to trust the devices their services run on, but also the resource providers need to trust users to not abuse the resources of their devices. Thirdly, we have to account for multi-tenancy in resource usage. More specifically, two users that do not necessarily trust each other might end up utilizing resources on the same device (cf. Section 6.2.1.2). Finally, no untrusted entity should have access to private information, which includes meta information such as file names or access patterns.

Achieving Deployability: With our scenario of decentralized individual cloud services, we aim for a drop-in replacement of today's public cloud services. Hence, *deployability of decentralized individual cloud services* becomes an important challenge to facilitate the seamless migration from public clouds. Most importantly, a

sufficient amount of different cloud services has to be available to replace today's public cloud services. Furthermore, we have to provide a simple *deployment of services*. As for public clouds, users need to be able to deploy decentralized individual cloud services themselves, without requiring interaction with other parties.

6.2.1.4 Related Work

One prominent stream of related work targets the delivery of cloud-like resources in a peer-to-peer manner, similar to the vision underlying our approach. As a first approach, P2PCS [BMT12] targets the peer-to-peer delivery of cloud infrastructure resources from a large, unreliable, and uncoordinated pool of devices. Likewise, Mayer et al. [MKH+13] propose an autonomic cloud system in which PaaS resources (cf. Section 2.1.2.1) are voluntarily provided by heterogeneous devices using a peer-to-peer system. Khan et al. [KNSV13] as well as Baig et al. [BFN16, BFN18] propose to extend community networks (cf. Section 5.3.6) to provide cloud resources in a peer-to-peer manner, e.g., to deploy tailored services at the edge of the network. These approaches have in common that they strive to offer publicly accessible computing resources in a distributed manner. They do not, however, explicitly address resulting privacy challenges, e.g., by ensuring that services are operated only on trustworthy infrastructure.

In contrast to these generic approaches, Cutillo and Lioy [CL13a, CL13b], similar to our motivation, specifically target the goal of preserving privacy by deploying cloud resources using a peer-to-peer overlay based on social trust. In their approach, users' cloud services can also be deployed to untrusted resources and, hence, the main objective of their approach is to leverage social trust to hide users' participation and interaction with cloud services. Our objective is different. We strive to solve the technical challenges of realizing decentralized individual cloud services over resource constrained devices in home networks in a secure manner. Still, the work of Cutillo and Lioy could enhance our approach by additionally providing anonymity for resource usages, i.e., by ensuring that resource providers do not learn who interacts with which cloud services.

To break up the inherent centrality of cloud computing, another stream of related work proposes to split up the *storage* of data over different cloud providers. RAIN [JZV+12] aims at splitting data into very small segments which are distributed among a multitude of storage providers. In contrast, MetaStorage [BKTM11] allows users to distribute data on a per-file basis over several existing cloud offers and has been extended to preserve compliance with privacy requirements [WMF13]. Likewise, CloudFilter [PP12] introduces a transparent proxy between users and their storage providers to automatically split data between different cloud storage providers based on users privacy requirements. Following a similar approach, NubiSave [SMS13] combines resources from multiple cloud storage providers to realize user-specific redundancy and security requirements. Yeo et al. [YPLL14] specifically target the use of multiple cloud storage providers on resource-constrained mobile devices. While these approaches still target traditional cloud infrastructures, FriendBox [GSMG12] builds up a storage cloud over resources contributed by friends. This

approach, however, trades in most of the advantages of cloud computing to achieve privacy. Specifically and in contrast to our work, users cannot benefit from scalability and accessibility, as the client used to access data has to realize any application logic and decryption or reassembling of data.

From a different motivation than ours, several approaches aim to utilize idle resources of home network devices to provide cloud-like services. For example, Seattle [CBKA09], a community cloud (cf. Section 2.1.2.2) built over commodity devices, aims at providing a learning platform. Caton et al. [CHC+14] extract trust levels from social networks to extend Seattle with trust-based resource allocation. CuteCloud [CZFK12] employs virtual machines to manage idle resources in a community cloud. CWC [ASS+12] strives to build a cloud over processing resources of charging smartphones. Similarly, ParaDrop [WDB14] aims at realizing edge computing by offloading processing tasks from the cloud back to home gateways. From a different perspective, different approaches propose to utilize idle resources of set-top boxes [JNC12] or mobile devices such as smartphones [ESM09, DKG+10] to build MapReduce clusters. Although these approaches, in contrast to our work, do not aim at preserving privacy when using *arbitrary* cloud services, they provide valuable input for addressing parts of our challenges, especially with respect to realizing cloud characteristics on resource-constrained devices.

With the goal to safeguard access to personal data, several approaches from related work [CCH+15, MGM+10, MSWP14] propose to create personal containers or data boxes. The core idea of these approaches is to store all personal data of a user in a secure location and selectively make this data available for specific purposes. Decentralizing individual cloud services, as proposed in our work, could provide a solid and secure foundation for realizing such approaches. From a different perspective, Sealed Cloud [JMR+14] employs TPMs to prevent insider attacks in traditional, data center clouds, where different security assumptions have to be considered. Still, their insights can partly be applied to our work where we use TPMs to further strengthen users' trust into resource providers.

Finally, from a more technical perspective, we have to consider approaches that either allow users to set up their own cloud environment or encrypt all data before it is sent to traditional cloud services. When shifting cloud services from traditional cloud infrastructure to devices controlled by the individual users (e.g., using ownCloud [Own18] or Seafile [Sea18]), often only one or very few devices are available to execute these cloud services on, which severely jeopardizes availability and scalability. This issue further exacerbates for private "clouds" hosted at the users' homes as home networks are typically connected via a single residential access line to the Internet and thus constitute a single point of failure. Similarly, when still utilizing public cloud services but encrypting data prior to upload (e.g., using Boxcryptor [Box18]), the inability of clouds to efficiently process encrypted data requires application logic and decryption on the client when accessing data. This diminishes many advantages of the cloud with respect to processing and accessibility. Additionally, cloud providers can still obtain valuable meta information, e.g., time and location of data access. Hence, current approaches put the user in the dilemma of having to choose between either her privacy or the advantages of cloud computing.

6.2.2 Decentralizing Individual Cloud Services with PriverCloud

To overcome users' dilemma of having to choose between preserving their privacy and benefiting from the advantages of cloud computing, we propose to decentralize individual cloud services. More specifically, we propose to create an individual PriverCloud instance for each user which is built upon trusted resources contributed by close friends and family. While this approach certainly is not suited for all kinds of cloud applications, it offers the user an additional choice for certain applications which target a small user group and have strong requirements for privacy, especially in fear of privacy threats originating from the tracking and surveillance by corporations and governments. In the following, we show how our decentralized architecture can be realized in a technical system and utilized to improve users' privacy.

When decentralizing individual cloud services with PriverCloud, we have to perform three core operations (cf. Section 6.2.1.1): (i) building-up an individual PriverCloud instance, i.e., acquiring the necessary resources as well as selecting and deploying cloud services, (ii) operating an individual PriverCloud instance, i.e., realizing the advantages of cloud computing in a peer-to-peer system over constrained resources, and (iii) securing operations within a PriverCloud instance, i.e., securing communication and authentication, separating cloud services, and extending social trust through technical measures.

In the following, we discuss how we realize these three core operations of PriverCloud and address the underlying challenges of respecting resource constraints, preserving cloud advantages, extending trust, and achieving deployability (cf. Section 6.2.1.3). Thereby, we arrange our presentation according to the typical usage pattern of our exemplary user Alice (cf. Section 6.2.1.1).

6.2.2.1 Building-up a PriverCloud

Before Alice can start to use cloud services in her PriverCloud instance, she first has to acquire the necessary storage and processing resources as well as to select and deploy services on top of these resources.

Acquiring Decentralized Resources

Initially, Alice has to acquire the storage and processing resources which are required to build her individual PriverCloud instance. To amplify privacy in contrast to public clouds, she must trust the resource providers to respect her privacy. In the context of our proposed PriverCloud architecture, we derive this required trust from existing social trust (e.g., close friends or family). More specifically, we propose to utilize idle processing and storage resources on devices ranging from less powerful, embedded devices (e.g., Raspberry Pis or NAS and set-top boxes) to more powerful devices such as off-the-shelf desktop computers owned and operated by trusted persons such as family members and friends. As discussed in Section 6.2.1.2, this underlying trust has to hold in both ways, i.e., resource providers also have to trust Alice to not misuse their resources (e.g., by requesting excessive amounts of

resources). Notably, existing social trust not only provides a foundation for realizing more privacy-friendly services but also provides incentives for contributing resources [GSMG12], as resource providers can trust Alice to also provide them access to her resources in a tit-for-tat manner (we further deepen this discussion in Section 6.2.4).

To build up Alice's PriverCloud instance over the storage and processing resources provided by her family and friends, we employ the concept of peer-to-peer computing. More specifically, each PriverCloud instance, i.e., exactly those resources that are available to one user to deploy her services and data on, constitutes one peer-to-peer network in which exactly those devices the user specifically selected participate. Consequently, as resource providers typically provide access to their devices to more than one user, devices can participate in more than one PriverCloud instance. We deepen our discussion on how we protect access to a PriverCloud instance, i.e., peer-to-peer network, of one user and how we separate services of different, potentially mistrusting, users running on the same device in Section 6.2.2.3.

Acquiring Cloud Services

After Alice has acquired the necessary processing and storage resources to build up her PriverCloud instance, she has to acquire cloud services that she can deploy on top of these resources. To achieve a seamless migration from today's public cloud offers to Alice's PriverCloud instance, we provide support for running existing cloud services developed for the AppScale platform [App18b], an open source implementation of the widely-used Google App Engine framework [Goo18a]. A wide range of cloud services that base on AppScale and/or Google App Engine is readily available today. Additionally, many developers are familiar with the employed programming model, which facilitates a steady development of new services. PriverCloud mimics the socket and storage APIs of AppScale and even introduces additional functionality, e.g., a transparent transport security mechanisms. Existing AppScale and Google App Engine services can be run in PriverCloud with negligible modifications to their source code (at most eight lines of additional boilerplate code).

To allow users to actually obtain services, we envision a service marketplace, similar to those for mobile apps on smartphones, that lists all available PriverCloud services and allows users to conveniently deploy a selected cloud service to their individual PriverCloud instance. In this marketplace, each service provides a description of its functionality and users can rate cloud services, which allows users to take a more informed decision. Additionally, we require source code availability, such that the functionality of a service can be audited by the operator of the marketplace or a trusted third party.

Deploying Cloud Services

Once Alice has selected a cloud service from the PriverCloud service marketplace, she has to deploy it in her PriverCloud instance. To this end, each PriverCloud instance is initiated with a special PriverCloud service, the *ControlCenter*. The ControlCenter provides a web interface that allows users to manage their cloud services,

e.g., deploying new cloud services in their PriverCloud instance. When deploying a new cloud service, the ControlCenter identifies the device of Alice's PriverCloud instance that fulfills the cloud service's resource requirements best (based on the current load) and deploys the cloud service on this device. Still, not every device of Alice's PriverCloud instance might be able to fulfill high resource demands of certain cloud services, e.g., bandwidth-intensive cloud services may face network limitations when deployed on the wrong device. To address this issue, we classify cloud services and devices according to their resource demands and availability, respectively. This enables the ControlCenter to place cloud services on devices that provide sufficient resources, e.g., an application with high uplink demand will be deployed on a device with sufficiently good Internet connectivity. By operating the ControlCenter within a PriverCloud instance, PriverCloud does not require additional client software to deploy cloud services and hence preserves one of the important advantages of cloud computing (cf. Section 6.2.1.3). Likewise, all operations necessary to deploy cloud services can be controlled from a web interface, similar to the marketplaces for mobile apps on smartphones, without the need for technical expertise and hence easing the migration from today's public cloud offers (cf. Section 6.2.1.3).

6.2.2.2 Operating a PriverCloud

Stepping away from the perspective of individual users, we now primarily focus on how to realize the advantages of cloud computing in the face of constrained resources often prevalent in home networks (cf. Section 6.2.1.3) when executing services in a PriverCloud instance. To this end, we provide measures for accessing cloud services, achieving service and data reliability, amplifying data redundancy, and realizing scalability. As for public cloud services, these technical measures are mostly hidden from users but are important to justify trust into the underlying system. In the following, we first describe how users interact with cloud services in the absence of failures and then detail how PriverCloud achieves reliability and scalability.

Accessing Cloud Services

Users need to be able to access their cloud services (similar to those deployed in public clouds), without having to know on which specific device they have been deployed. To achieve this goal, we assign each cloud service a DNS hostname under which this service can be accessed using Dynamic DNS. The ControlCenter updates the corresponding Dynamic DNS entry whenever the cloud service is migrated to another device or if the IP address of the device to which the cloud service was deployed changes.

However, devices in home networks typically should run more than one cloud service despite having only one public IP address. Hence, PriverCloud has to demultiplex incoming request to individual cloud services. As we specifically target privacy-sensitive services and communication traverses the untrustworthy Internet (cf. Section 6.2.1.1), it is reasonable to assume that all communication will be protected using transport layer security (TLS). Thus, we can utilize the server name

indication (SNI) extension of the TLS protocol [Eas11] for demultiplexing between different cloud services. More specifically, modern clients such as web browsers and smartphones will automatically include the DNS hostname of the cloud service they want to contact in the initial handshake process. Since the hostname is transmitted in plaintext (cf. Section 3.3.2.2), PriverCloud can use this information to demultiplex received requests to the correct individual cloud services without diminishing security guarantees.

Service Reliability

Cloud services that are currently being executed in a PriverCloud instance may abort at any time due to device or network failures. As in public clouds, such failures must be handled transparently for the user, i.e., cloud services must automatically recover from device or network failures without requiring user interaction. To this end, we extend the ControlCenter that manages service deployment for each individual PriverCloud instance (cf. Section 6.2.2.1) to also monitor the status of each deployed cloud service using the TLS heartbeat extension [STW12]. More specifically, after starting a cloud service, the ControlCenter establishes a TLS connection to this service and continuously sends heartbeat messages. When the ControlCenter does not receive a heartbeat response within a specified timeframe (in our implementation five consecutive heartbeats), it assumes a service malfunction. A detected malfunction of a cloud service then triggers a recovery of this service by deploying it to another device in the user's PriverCloud instance. In this process, a sensible selection of the heartbeat frequency is crucial as it configures the trade-off between detection delay and bandwidth consumed for monitoring. Additionally, a grace period before initiating the recovery of a service can avoid unnecessary overhead in case of temporary failures. Notably, also the ControlCenter can fail, hence, we operate multiple instances that monitor each other. Here, only one of the ControlCenters is actually used to deploy and monitor services, while the other ControlCenters only act as a stand-in for the actually used ControlCenter.

Data Reliability

While the above approach allows us to restart cloud services in case of errors, this does not hold for the data persistently stored by these services. To make service recovery transparent for users, a service requires access to all previously stored data after recovery. Hence, we decouple the storage location of data from the processing location of services and provide redundant storage using a distributed hash table (DHT) [WGR05] that spans across the devices within a PriverCloud instance. This allows cloud services to store and later retrieve data independently from their processing location and, hence, also after a recovery. We further increase reliability by storing data on more than one device to create redundancy. Additionally, using a DHT enables us to address the resource heterogeneity of devices (cf. Section 6.2.1.3): We dynamically adjust the value range of the DHT for which a specific device is responsible for and assign devices with a large amount of storage resources more

than one value range. As a result, we can balance the storage load of the devices in a user's PriverCloud instance according to the available resources.

Securing and Amplifying Data Redundancy

Achieving data reliability comes at the cost of additional storage space needed for creating the required storage redundancy. To offer another approach to create storage redundancy and hence increase the reliability of stored data, we propose to utilize the virtually infinite storage resources of public cloud storage services [BKTM11] with the goal to extend the available storage space of a device with access to the cloud storage account of the provider of this device. However, especially when using public cloud storage, we have to guarantee data confidentiality and prevent unauthorized modifications. Thus, we transparently apply encryption and integrity protection to data before storing it in the DHT, similar to the object security mechanism applied by SCSlib to protect IoT data when it is stored in the cloud (cf. Section 5.2.2), such that only the user-controlled service can decrypt and thus access the data.

Nevertheless, we have to take care that untrusted parties, especially the providers of utilized public cloud storage services, do not learn meta information such as time and location of data access (cf. Section 6.2.1.3). Hence, instead of using Alice's public cloud storage account (e.g., Dropbox or Google Drive) for storing the data, we extend the storage resources of devices by using the public cloud accounts of the providers of these devices. With this approach, we can amplify the redundancy of data storage and not only protect confidentiality and integrity of outsourced data but also successfully hide the origin of data stored in public clouds.

Scalability

Finally, more advanced or frequently used cloud services may require more processing resources than even powerful devices in a PriverCloud deployment can provide. In this case, we follow the scale-out approach prevalent in cloud deployments today and distribute one cloud service over multiple devices in a PriverCloud instance. This deployment model becomes especially feasible if the processing load is induced by user requests and hence request level parallelization can be employed to split a cloud service into independent components that require only little synchronization.

In contrast, if a service requires operating on large amounts of data, we can employ a paradigm similar to MapReduce [DG04] to perform operations on data as close to its storage location as possible. With respect to increasing storage demands, our DHT approach for providing reliable storage of data (see above) is inherently scalable. Alice has to simply acquire more storage resources, e.g., by adding more devices or additionally utilizing public cloud storage, if the need arises.

6.2.2.3 Securing a PriverCloud

Finally, to ensure user's privacy in a PriverCloud instance, we have to ensure that only trusted entities can participate in a PriverCloud instance and that data and

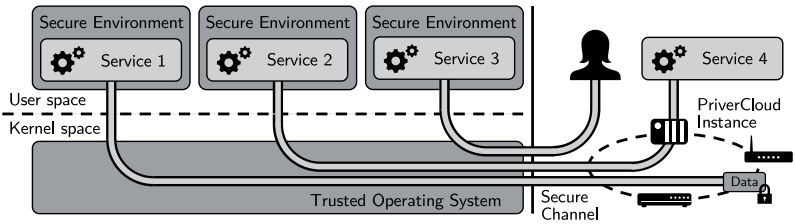

Figure 6.3 PriverCloud realizes secure end-to-end communication and authentication, separates different cloud services executed on the same device, and extends social trust into resource providers through technical measures.

communication are protected against unauthorized access. To this end, PriverCloud enforces secure communication and authentication between the devices within a PriverCloud instance as well as between users and their cloud services, separates different cloud services that are executed on the same device, and augments the social trust into resource providers (cf. Section 6.2.2.1) with technical measures. We provide an overview of the security measures of PriverCloud and how they integrate into our deployment scenario in Figure 6.3. All communication in PriverCloud takes place over secure channels and is authenticated such that only authorized devices and services can participate in a PriverCloud instance. Furthermore, all cloud services are separated using secure execution environments to prevent interference between cloud services of different users. Finally, a trusted operating system ensures that resource providers do not exploit users' trust and protects against security breaches from outside entities. In the following, we discuss these operations in more detail.

Secure Communication and Authentication

Both, the communication between the different devices in Alice's PriverCloud instance as well as the communication of Alice with her cloud services traverse the untrusted Internet and hence have to be protected. Besides integrity and confidentiality, this includes the authentication of communication peers, e.g., to prevent that untrusted and hence unauthorized devices join Alice's PriverCloud instance.

To protect the confidentiality of communication in Alice's PriverCloud as well as to authenticate devices, cloud services, and users, we rely on mutually-authenticated TLS channels for the communication between devices as well as between users and cloud services. Alice relies on or operates a certificate authority to issue TLS certificates for the access to her PriverCloud instance. Specifically, Alice deploys TLS certificates that grant access to her PriverCloud instance to all devices she trusts, e.g., operated by family and close friends. Devices in the DHT underlying Alice's PriverCloud will only accept connections from other devices if these authenticate using a valid TLS certificate issued by the corresponding certificate authority. Hence, only devices that are authorized (and hence trusted) by Alice can participate in her PriverCloud instance. Likewise, the users of Alice's cloud services will only establish

connections with those cloud services that provide a valid TLS certificate from the respective certificate authority.

Finally, a special class of certificates is issued by Alice to her ControlCenters to authenticate at devices in Alice's PriverCloud instance, e.g., when deploying cloud services on Alice's behalf. At the same time, ControlCenters verify the identity of devices before deploying cloud services based on the issued TLS certificates.

Efficient Separation of Cloud Services

Different cloud services, deployed in different PriverCloud instances and hence under control of different users, can run on the same device in parallel (cf. Figure 6.1). To ensure Alice's privacy in this situation, we require a strict separation of different cloud services deployed on the same device. Furthermore, the resource provider may wish to reserve a certain amount of resources for own local services, i.e., ensure that cloud services deployed to the own device can use those resources explicitly assigned to them.

To address these requirements for efficiently (in terms of processing and memory overhead) separating cloud services, PriverCloud employs virtualization to sandbox different cloud services running on the same device. Sandboxing cloud services using virtualization comes with two benefits. First, virtualization allows us to protect a cloud service against other cloud services running on the same device. Second, we can use virtualization to closely restrict access to resources, e.g., to prevent direct, unrestricted access of cloud services to the Internet or file system, and to enforce the usage of dedicated APIs to access resources. For example, cloud services can only use dedicated PriverCloud sockets that automatically enforce security and authentication for network communication. Likewise, access to file storage (realized using PriverCloud's underlying DHT) is only possible through API endpoints that automatically and transparently handle the encryption and decryption of data, similar to SCSlib in the context of the IoT (cf. Section 5.2).

However, one of the inevitable challenges of PriverCloud results from the limited processing resources, especially when considering resource-constrained, cheap devices (cf. Section 6.2.1.3), which prohibits virtualization using fully-fledged virtual machines. To account for this challenge, we employ lightweight virtualization mechanisms to not pose additional processing overhead on the devices. More specifically, we use Linux containers (LXC), an operating-system-level virtualization, to realize the AppScale-compatible PaaS environment, where only the platform APIs can be accessed (cf. Section 6.2.2.1), and thus avoid the overhead of full virtual machines. We performed measurements to verify that even a resource-constrained Raspberry Pi is able to launch more than 30 basic cloud services (delivering a simple website) isolated in individual LXC containers in parallel.

Beyond Social Trust

While social trust might be sufficient for resource providers to provide their family and close friends with access to storage and processing resources, users might still

fear that resource providers operate cloud services in a dishonest manner, e.g., to get access to sensitive information during the execution of a cloud service. Likewise, attacks targeting the devices that form a PriverCloud instance can subvert the trust founded on social relationships. To protect against such maliciously altered devices, PriverCloud offers the option to leverage trusted platform modules (TPMs), which are available on most modern desktop computers, to remotely attest the integrity of a device before deploying a cloud service to this device. A TPM enables hardware-based security by providing cryptographic operations such as key generation, encryption, signature generation, and cryptographic hash computation [TCG07].

Employing a TPM in PriverCloud, the goal is to ensure that the system on which we deploy a cloud service has not been tampered with and operations such as inspecting the memory of a running cloud service are not possible.

Furthermore, such an approach allows for the deployment of secrets such as private keys necessary to access encrypted data and TLS certificates to cloud services without the respective resource provider being able to access this information. Hence, we need to create a chain of trust from the TPM of a device to the process of deploying and operating a cloud service on this device. To this end, we introduce a trusted component and a trusted kernel. The trusted component is a small piece of software that runs in user space and allows users to securely bootstrap their cloud services, especially with respect to deploying secrets to a starting cloud service.

Likewise, the trusted kernel is a modified Linux kernel that ensures the correct operation of the trusted component and can later be extended to protect access to the volatile memory of cloud services.

Finally, the TPM enables users to remotely attest the integrity of the trusted kernel and thus create a chain of trust from the TPM to the deployment and operation of their cloud services [TCG07]. By leveraging the capabilities of TPMs, PriverCloud allows users to check if a specific device executes only trusted software components [MPP+08] and hence realize a trusted platform for service execution.

6.2.3 Evaluation

To assess the feasibility of PriverCloud and to thoroughly quantify its performance, we implemented a prototype for the device side of PriverCloud using the C programming language. Additionally, we realized the ControlCenter as a PriverCloud service using Python. We rely on OpenSSL for the cryptographic operations, LXC for creating virtualized environments, dnspython to interface with Dynamic DNS, the Linux kernel as the foundation for a trusted operating system, and IBM's Software Trusted Platform Module as the library for all TPM related tasks. As cryptographic primitives, we use AES with 256 bit keys in CBC mode for encrypting data and SHA-256 as HMAC for protecting the integrity of data. To securely bootstrap cloud services, we rely on the cryptographic primitives offered by TPMs, namely SHA-1 for verifying the integrity of code regions and AES with 128 bit keys in CCM mode for encrypting secrets.

We utilize two different classes of devices for our evaluation, namely embedded devices and desktop computers. As an exemplary embedded device, we chose the Raspberry Pi Model B with a 700 MHz ARM11 CPU, 512 MB of RAM, and Raspbian Jessie Linux as the operating system. For the class of desktop computers, we selected a machine with a four core 2.93 GHz Intel i7 870 CPU, 4 GB of RAM, and Ubuntu 14.04 as the operating system with our custom trusted kernel. To create a PriverCloud instance for evaluation purposes, we connect two Raspberry Pis and one desktop computer using a 100 Mbit/s switch. Whenever we measure communication between a user device and a device or cloud service in a PriverCloud instance, we use one of the Raspberry Pis as user device.

In the following, we first evaluate the processing and storage overhead of PriverCloud's secure storage. We then study the overhead of secure communication and authentication in PriverCloud, especially with respect to the deployment of cloud services in a PriverCloud instance. Finally, we investigate the trade-off between service reliability and consumed bandwidth for service monitoring.

6.2.3.1 Secure Storage

All data that cloud services persistently store in PriverCloud is automatically encrypted and integrity protected (cf. Section 6.2.2.2). This strong level of security introduces two types of overhead: (i) storage overhead for data stored in PriverCloud's DHT and (ii) processing overhead for performing the necessary cryptographic operations. In the following, we quantify these overheads.

Storage Overhead

PriverCloud automatically applies encryption and integrity mechanisms to all data before it is persistently stored to ensure that only user-controlled services can access this data. These mechanisms increase the required storage space. More specifically, for encrypting data using AES in CBC mode, we require a constant overhead of 16 bytes for the random initialization vector and additionally between 1 and 16 bytes of padding, as AES operates on blocks of size 16 bytes. Furthermore, protecting integrity based on HMAC with SHA-256 results in a constant overhead of 32 bytes per data item. In summary, PriverCloud's secure storage adds a constant storage overhead of at most 64 bytes per stored data item, irrespective of the size of the data item. To put these numbers into perspective, for a small file of size 1 KB, this results in a storage overhead of 6.4 %. If we consider larger files, e.g., a compressed image file of size 1 MB, this overhead reduces to only 0.0064 %.

Processing Overhead

Besides resulting in a modest overhead in storage size, PriverCloud's secure storage also induces processing overheads for encryption and integrity protection. To quantify these overheads, we measure the processing time required for encrypting data and applying integrity protection before data is persisted in PriverCloud's storage

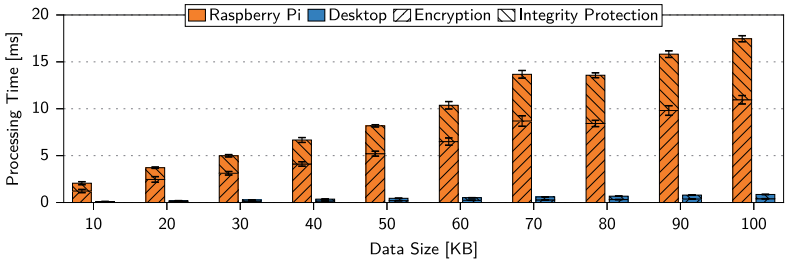

Figure 6.4 The overhead of PriverCloud's secure storage results from encryption as well as integrity protection and scales roughly linearly with increasing data size.

for different data sizes in Figure 6.4. For each measurement point, we conduct 100 repetitions and report on the mean over these repetitions with 95 % confidence intervals. As expected, the processing overheads resulting from encryption and integrity protection increase with the data size, as both AES encryption as well as SHA-256 as underlying hash function for integrity protection process data in blocks and increasing data size results in more blocks that need to be processed. Additionally, we observe that the desktop computer benefits from its stronger processing power and can perform both operations noticeably faster. For the Raspberry Pi, the overhead required for encryption is nearly twice as high as the overhead resulting from integrity protection. More specifically, for a data size of 100 KB, the Raspberry Pi requires 10.96 ms for encryption and 6.51 ms for integrity protection. In comparison, the desktop-grade machine requires only 0.40 ms for encryption and 0.46 ms for integrity protection in the same setting. Putting these numbers into perspective, a Raspberry Pi can perform the necessary cryptographic operations to persistently store 57 files of size 100 KB per second in PriverCloud's secure storage. These numbers increase to 1190 files per second for the more powerful desktop-grade machine under study. We believe that these numbers are clearly sufficient for many real-world use cases, especially when considering individual cloud services that serve only one user or at most a small group of users (cf. Section 6.1).

6.2.3.2 Secure Communication and Authentication

Besides automatically ensuring confidentiality and integrity of data during storage, PriverCloud also automatically secures and authenticates all communication, including the communication required for deploying cloud services. In the following, we evaluate the impact of PriverCloud's secure communication and authentication on TLS handshakes and cloud service performance as well as quantify the overhead of securely deploying cloud services in PriverCloud.

Impact on TLS Handshake

To support the deployment of multiple cloud services (possibly from different users) to one device with only one IP address, PriverCloud automatically demultiplexes

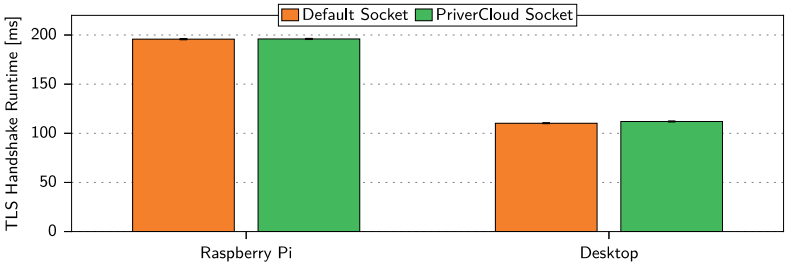

Figure 6.5 PriverCloud's secure socket implementation minimally increases the time required for performing a full TLS handshake with a cloud service running on top of PriverCloud.

incoming TLS connections to the correct cloud service running inside a secure execution environment (cf. Section 6.2.2.2). We begin our evaluation of the resulting overheads by studying the impact of using these automatically secured PriverCloud sockets on the time required for completing a TLS handshake, i.e., those operations and communication required to establish a secure communication channel. To this end, we measure the time required to perform a full TLS handshake between a client running on a Raspberry Pi and a PriverCloud cloud service running on both, another Raspberry Pi and the desktop-grade machine. We perform 10 000 handshakes both using PriverCloud's secure socket and the default socket implementation of Linux and compare their mean runtime for the complete TLS handshake with 95 % confidence intervals in Figure 6.5.

For cloud services running on the Raspberry Pi, the full handshake requires on average 195.81 ms using default sockets and 196.00 ms using PriverCloud sockets. Likewise, on the desktop-grade machine, using default sockets result in a handshake runtime of on average 110.21 ms compared to 112.03 ms when using PriverCloud sockets. Hence, PriverCloud's secure sockets have only a small, negligible impact on the time required for completing a full TLS handshake with a cloud service running on top of PriverCloud.

Impact on Cloud Service Performance

Besides the initial handshake, PriverCloud sockets potentially also impact the performance of cloud services' communication over the established secure communication channel. To capture this effect, we again measure the amount of time required for communicating a specific amount of data from a client to a cloud service and back. More specifically, a client running on a Raspberry Pi transmits a certain amount of data to a cloud service running on another Raspberry Pi as well as a desktop-grade machine. Subsequently, the cloud service echoes the received data back to the client on the Raspberry Pi. In Figure 6.6 we report on the resulting mean transmission time over 1000 measurements with 95 % confidence intervals for an increasing amount of transmitted data in each direction. We measure only the time required for the actual transmission of data and consequently omit the time

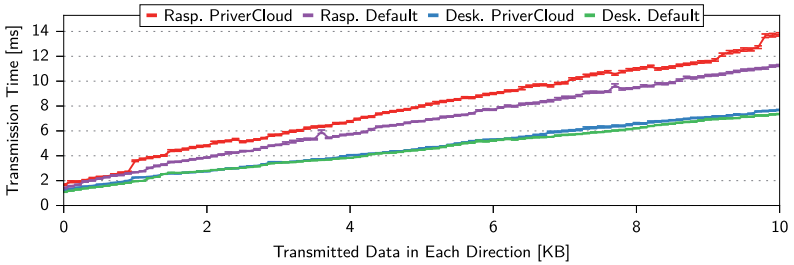

Figure 6.6 Using PriverCloud sockets adds a modest overhead for the transmission time required for communicating with a cloud service.

required for establishing the connection, i.e., for performing the TLS handshake. Again, we compare our implementation of PriverCloud sockets against the default socket implementation of Linux.

Overall, we observe that the transmission time increases roughly linearly with the amount of transmitted data when using PriverCloud sockets. For 1 KB of transmitted data (in both directions), using PriverCloud sockets increases the required transmission time by 34.46 % (from 2.67 ms to 3.59 ms) on the Raspberry Pi and by 15.90 % (from 1.95 ms to 2.26 ms) on the desktop-grade machine. Likewise, for 10 KB of transmitted data, we observe an increase of 22.20 % (from 11.26 ms to 13.76 ms) on the Raspberry Pi and of 4.35 % (from 7.36 ms to 7.68 ms) on the desktop-grade machine. Especially for larger and hence longer transmissions, this overhead, which is necessary to effectively separate different cloud services running on the same physical device, hence constitutes a manageable performance penalty.

We observe a spike in the required transmission size for PriverCloud sockets at about 1 KB on the Raspberry Pi (and to a lesser extent also for the desktop machine). This effect results from our implementation of PriverCloud sockets where a buffer of size 1024 byte requires fragmentation for data larger than about 1 KB. We do not observe this effect for larger data sizes since data then exceeds the MTU of the underlying connection and is thus fragmented already on the network layer, which further reduces the performance impact of PriverCloud sockets on cloud services.

Deployment of Cloud Services

Not only the communication with cloud services but also the initial deployment of cloud services to devices in a PriverCloud instance is influenced by the underlying security mechanisms that account for privacy when delivering cloud services.

In the following, we hence analyze the time required for deploying cloud services in PriverCloud. To this end, we report on the mean time required for deploying a cloud service over 1000 repetitions with 99 % confidence intervals. We specifically crafted a simple cloud service for these measurements that does not realize any actual functionality. In our measurements, the ControlCenter that deploys the cloud service

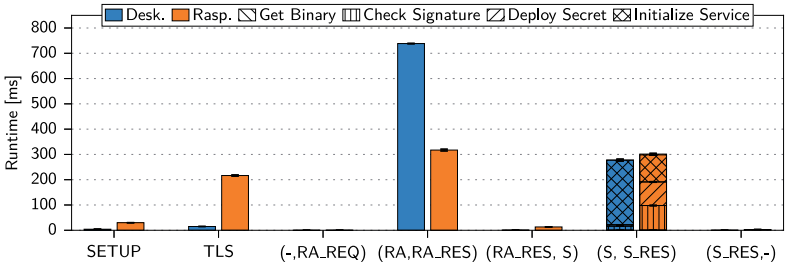

Figure 6.7 The time required for deploying a cloud service in a PriverCloud instance amounts to about 1 s and predominantly results from remote attestation and starting the cloud service within a secure execution environment.

runs on the same physical device to which it also eventually deploys the cloud service, which rules out any influences of the underlying network on the measurement results.

In Figure 6.7 we show the mean time required for performing the necessary steps for deploying a cloud service. First, the ControlCenter performs the *SETUP* for cloud service deployments such as receiving the necessary TLS certificates from the secure PriverCloud storage (these operations can be cached and only have to be performed once for the first deployment of a cloud service). The required operations require on average 4.17 ms on the desktop-grade machine and 29.98 ms on the resource-constrained Raspberry Pi. Subsequently, the ControlCenter establishes a TLS connection to the target device, i.e., the device to which the cloud service shall be deployed. This process consumes 216.51 ms on the Raspberry Pi (cf. Figure 6.5) respectively 15.02 ms on the desktop machine (this number is smaller than reported in Figure 6.5 since the handshake does not involve a Raspberry Pi as client this time). To finalize the initialization phase, the ControlCenter sends a request for remote attestation (*RA_REQ*) to the target device. Since this operation requires only the transmission of a nonce, we can perform it within a negligible 0.03 ms on the desktop machine respectively 0.25 ms on the Raspberry Pi.

Upon reception of the remote attestation request, the target device performs the remote attestation (*RA*) and sends the corresponding results (*RA_RES*) back to the control center. This step can be performed in on average 738.46 ms on the desktop machine respectively 317.23 ms on the Raspberry Pi. Notably, the Raspberry Pi outperforms the desktop machine in this step. Since the Raspberry Pi does not provide a hardware TPM, we have to rely on a software TPM for our measurements that, despite offering less security, operates considerably more efficient. Hence, executing the remote attestation heavily depends on the processing performance of the underlying (hardware) TPM. After receiving the remote attestation response (*RA_RES*), the ControlCenter performs the necessary cryptographic operations to validate the remote attestation and subsequently triggers the start of the cloud service (*S*). On a desktop machine, these operations consume about 0.69 ms compared to 13.18 ms on the Raspberry Pi. This difference mainly results from the slower processing of cryptographic operations on the Raspberry Pi (cf. Section 6.2.3.1).

Now, the target device can perform the start of the cloud service (S) and sends an acknowledgment of the successful start back to the ControlCenter (S_RES). In total, starting a cloud services requires 277.63 ms on the desktop machine and 301.33 ms on the Raspberry Pi. We break down this total runtime into its individual components in the following and illustrate this using hatching in Figure 6.7. The target device first obtains the cloud service's binary from the secure PriverCloud storage, which requires 0.01 ms on the desktop machine and 0.12 ms on the Raspberry Pi (for local communication). Subsequently, the target device verifies the integrity of the binary using its digital signature within 16.16 ms on the desktop machine respectively 97.77 ms on the Raspberry Pi. Now, the ControlCenter deploys the secret (used to protect the cloud service's data) to the cloud service. The necessary cryptographic operations for this step require 5.70 ms for the desktop machine compared to 93.30 ms for the Raspberry Pi.

Finally, the target device creates the secure execution environment and actually starts the cloud service. This process takes 255.40 ms on the desktop machine respectively 106.91 ms on the Raspberry Pi. The noticeably higher processing time on the desktop machine results from an implausibly long delay when creating LXC containers. While we could not identify the root cause of this delay we verified that it also occurs with an unmodified kernel and thus does not result from our modifications. Still, we believe that this likely malfunction can be circumvented to further speed up the cloud service deployment. Finally, processing the acknowledgment of the successful start of the cloud service (S_RES) on the ControlCenter requires only 0.28 ms on the desktop-grade machine and 3.14 ms on the Raspberry Pi.

In total, the time required for securely deploying a cloud service in PriverCloud requires 1.04 s on the desktop machine and 0.87 s on a Raspberry Pi. The numbers for the desktop machine can likely be improved by optimizing the creation of LXC containers, while the runtime on the Raspberry Pi probably increases modestly when switching from a software TPM to a hardware TPM (hardware TPMs are typically slower than those emulated in software). Still, we are able to securely deploy a cloud service within about 1 s, even in the face of resource-constrained devices such as a Raspberry Pi. Such a short deployment time is especially important when having to restart services in case of device or network failures.

Notably, the majority of computational effort occurs on the target device (98.05 % for the desktop machine and 69.83 % for the Raspberry Pi, respectively) that at the time of deployment likely has spare resources anyways, because it was selected to operate the cloud service from now on. Hence, we do not put a huge computational burden on the device that operates the ControlCenter and might not have large amounts of spare resources.

6.2.3.3 Service Reliability Trade-off

To ensure the availability and reliability of cloud services, PriverCloud monitors the reachability of deployed cloud services. More specifically, a user's ControlCenter continuously sends out heartbeat messages to all deployed cloud services of this user (cf. Section 6.2.2.2). Likewise, PriverCloud operates multiple instances of the

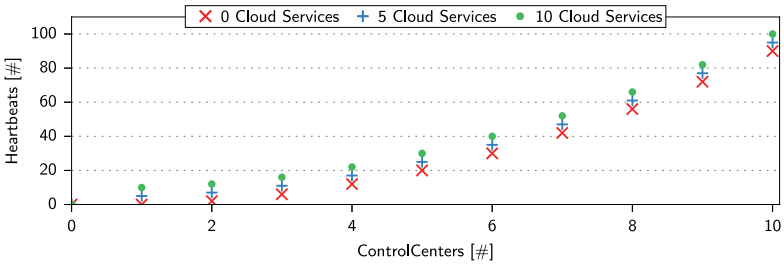

Figure 6.8 The number of heartbeats in a PriverCloud instance scales linearly in the number of deployed cloud services and exponentially in the number of backup ControlCenters.

ControlCenter that monitor each other and act as a backup in case the primary ControlCenter fails. As PriverCloud relies on OpenSSL's heartbeat mechanism, each heartbeat request results in an IP packet of size 118 byte, each response requires an IP packet of size 118 byte as well, and the acknowledgment for the response consumes another 52 byte. Consequently, each heartbeat creates 288 byte of network traffic (on the network layer). In the following, we analyze the network traffic resulting from the ControlCenters and cloud services of one user.

The number of heartbeats that are sent in a user's PriverCloud instance depends on the number of cloud services and ControlCenters this user has deployed. Here, the first ControlCenter monitors all cloud service and the other ControlCenters, while the remaining ControlCenters only monitor the other ControlCenters.

In a first step, we calculate the number of heartbeat requests that are created within one heartbeat interval, i.e., each ControlCenter sends exactly one heartbeat request to each monitored cloud service and ControlCenter. Hence, the number of heartbeats depends on the number of deployed cloud services and ControlCenters as depicted in Figure 6.8. The number of heartbeat requests is linear in the number of deployed cloud services (as each cloud service is monitored by only one ControlCenter) and exponentially in the number of ControlCenters (as each ControlCenter monitors all other ControlCenters). For example, when operating one ControlCenter, no heartbeat requests are necessary when no cloud services are deployed, while monitoring 5 cloud services requires 5 heartbeat requests and 10 deployed cloud services result in 10 heartbeat requests. In contrast, a PriverCloud instance with 10 deployed CloudCenters already requires 90 heartbeats, even if no cloud services are deployed. This number increases only modestly to 95 heartbeats for 5 deployed cloud services and 100 for 10 deployed cloud services, respectively.

To derive the network overhead actually resulting from PriverCloud's heartbeat mechanisms, we additionally have to take the heartbeat frequency, i.e., how many heartbeats a ControlCenter sends to each monitored cloud service and to each other ControlCenter per second. Intuitively, a higher heartbeat frequency allows to detect failures earlier but also puts more burden on the network.

In Figure 6.9, we study the network traffic of one ControlCenter for an increasing heartbeat frequency and number of ControlCenters in one PriverCloud instance. We

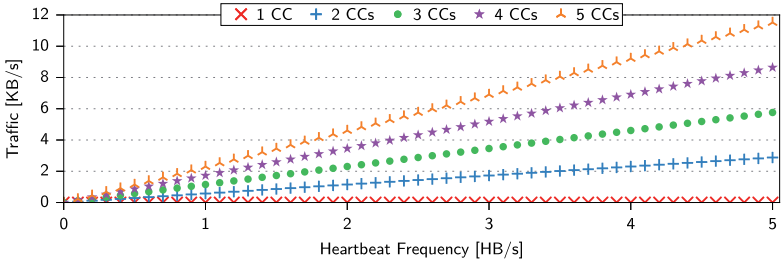

Figure 6.9 The network overhead generated by one user's ControlCenter scales linearly in the number of ControlCenters in the user's PriverCloud instance and the heartbeat frequency.

omit the traffic necessary for monitoring cloud services, since monitoring these results in less traffic (see above). When monitoring one other ControlCenter at a frequency of 1 heartbeat/s, a ControlCenter creates outgoing traffic of 0.17 KB/s and incoming traffic of 0.12 KB/s, totaling to a network traffic of 0.29 KB/s. As ControlCenters monitor each other pairwise, this number doubles to a network traffic of 0.58 KB/s on each ControlCenter for a PriverCloud instance with two ControlCenters. When considering a PriverCloud instance with five ControlCenters monitoring each other at 5 heartbeat/s, the total network traffic amounts to 11.52 KB/s per ControlCenter. These numbers are well manageable with today's residential access links that typical offer bandwidth in the order of tens of Mbit/s.

A reasonable selection of the heartbeat frequency is not only imperative to reduce network overhead but also to achieve fast detection and recovery of device, network, and software failures. Furthermore, to account for temporary aspects such as a dropped single packet, we propose a grace period, i.e., a certain number of missing consecutive heartbeats, before we assume a failure and hence start our recovery procedure, i.e., restarting the failed cloud service or ControlCenter on another device. For example, with a heartbeat frequency of 1 heartbeat/s and a grace period of 5 heartbeats, we can detect a failure within at most 6 s with modest costs in terms of network overhead. Since restarting a cloud service takes about 1 s (cf. Section 6.2.3.2), we can completely recover from failures within about 7 s. While this is sufficient for most use cases, it can be further reduced by increasing the heartbeat frequency or by decreasing the grace period. Such optimizations, however, come at the cost of an increased amount of required network traffic for larger heartbeat frequencies respectively an increased likelihood of unnecessary migration overhead if cloud services recover from short-term failures after a too short grace period.

Finally, another important trade-off between reliability and resource overhead is the question of how many ControlCenters to deploy in a user's PriverCloud instance. While the definitive answer to this question has to factor in parameters such as the number of devices available to deploy ControlCenters to and the number of cloud services that need to be monitored, an important consideration is that the network overhead scales exponentially in the number of ControlCenters. Hence, we believe that—similar to traditional cloud deployments (cf. Section 4.3.2.3)—a redundancy of

three ControlCenters constitutes a sensible trade-off between reliability and resource consumption. For larger PriverCloud instances with many cloud services increasing the number of ControlCenters up to five might be conceivable, while using even more ControlCenters is likely not advisable.

6.2.4 Summary and Future Work

PriverCloud is motivated by the observation that not all classes of cloud services unconditionally require the scalable resources made available by public cloud infrastructure. Individual services, i.e., those services where users interact only with their own data, can often be realized on a single, not very powerful device. Hence, shifting these services away from public clouds to commodity hardware provided by trustworthy persons can significantly strengthen users' privacy. However, such an approach comes with several challenges, ranging from resource constraints over preserving the advantages offered by public clouds to the need to extend trust and allow for easy deployability.

To overcome these challenges, we present PriverCloud, our approach for decentralizing individual cloud services, especially focusing on privacy-sensitive services. PriverCloud deploys users' services solely to trusted infrastructure contributed by close friends and family and thus breaks up the inherent centrality of cloud computing. To this end, PriverCloud relies on standardized security mechanisms to achieve confidentiality and integrity of data during transport and security and employs lightweight virtualization to separate different cloud services executed on the same device. By continuously monitoring devices and cloud services in a PriverCloud instance, we achieve high reliability despite relying on commodity hardware deployed in home networks. PriverCloud enables the execution of already existing cloud services that were developed for the Google App Engine platform [Goo18a] and hence eases the migration from today's public cloud offers. Additionally, PriverCloud offers the option to employ TPMs to substantiate real-life trust relationships with technical guarantees.

As our evaluation of PriverCloud shows, it is feasible to securely distribute a user's individual cloud services to commodity hardware in home networks. PriverCloud's secure storage induces a constant negligible storage size overhead and allows for the processing of tens of encrypted files per second even on constrained devices. Likewise, PriverCloud's mechanism to automatically secure all communication with cloud services using standard transport layer security mechanisms adds only a manageable performance penalty to connections with and between cloud services. When deploying a cloud service to a new device, PriverCloud is able to perform the necessary security operations in about 1 s, even when employing a TPM to further substantiate trust assumptions.

PriverCloud's monitoring approach based on TLS heartbeats allows to reliably detect and recover from device and cloud service failures within about 7 s for conservative choices of monitoring parameters. This timeframe can further be reduced with more aggressive monitoring parameters. So far, our evaluation has been focused on

measurements in reliable high-speed local networks. For future work, it would be interesting to also study the impact of higher latencies and potentially packet loss on the overhead for deploying and monitoring cloud services.

Although we focus on home devices as the most challenging deployment scenario in the scope of this dissertation, PriverCloud instances can also be deployed on more powerful infrastructures, e.g., in corporate settings or federated clouds. In contrast to using only own infrastructure to operate services, such a deployment would enable enterprises to benefit from a cloud infrastructure even if legislation or customers' concerns render the utilization of traditional cloud computing challenging. Accounting for the different security assumptions and resource constraints in such scenarios constitutes an interesting avenue for future work. Likewise, when relying on TPMs, selected aspects of PriverCloud's architecture might even be realized using resources of public cloud infrastructures, e.g., as proposed in the context of SealedCloud to safeguard against insider attacks [JMR+14].

Furthermore, besides the technical challenges discussed and approached in this chapter, the concept of PriverCloud also constitutes exciting legal and economic questions for future work. First and foremost, the question arises how law can be enforced in such a decentralized setting. In our opinion, PriverCloud shows great potential in realizing a trade-off between valid interests involved with criminal prosecution and the people's fear of mass surveillance, especially through foreign intelligence agencies. Individual devices in a PriverCloud instance can still be seized or wire-tapped if need arises, however, the inherent decentrality renders the unduly monitoring of all users virtually impossible.

Another interesting legal question concerns the liability of the device owner, especially if a cloud service is misused for cybercrimes such as sending SPAM emails or hacking. From a technical perspective, we aim to counter these threats by our trust model (cf. Section 6.2.1.2) and restricting access to resources (cf. Section 6.2.2.3). When focusing on economic questions for future work, the main concern is the motivation or compensation for providing resources for others. Here, we see two promising complementary approaches. As we build on existing social trusts, users have good reason to rely on the concept of quid pro quo. Still, should users encounter an imbalance in resource-usage and want to be compensated for this, we propose to use micro-payment schemes such as Bitcoin [Nak08] to reimburse resource providers.

In conclusion, PriverCloud constitutes an approach for retaining privacy when using cloud services by moving them from public clouds to decentralized private cloud instances. Thereby, PriverCloud breaks up the inherent centrality and non-transparency of cloud computing without the need to give up its advantages.

6.3 Conclusion

Our work presented in this chapter is driven by our anticipation that—besides all efforts to make cloud computing more privacy-friendly—users still might have extraordinary high privacy requirements and/or mistrust into cloud providers. We

complemented this view by our observation that not all categories of cloud services imperatively need large amounts of resources as typically provided by public cloud infrastructure. Especially for individual services, e.g., calendar and contact synchronization, we hence proposed to move these services with strong privacy requirements away from public cloud systems to trusted infrastructure operated by users' close friends and family.

To achieve this goal, we proposed PriverCloud, our technical approach that utilizes idle resources of commodity devices operated in home networks to form a trusted, decentralized peer-to-peer system as a foundation to securely deploy and operated cloud services. As PriverCloud solely requires cooperation between users, we effectively eliminate any trust assumptions with respect to traditional cloud service or infrastructure providers. To further strengthen trust between users with technical security measures, PriverCloud optionally allows utilizing TPMs to fortify the privacy of users' data. Notably, PriverCloud eases the migration away from today's public cloud deployments as existing cloud services developed for the AppScale or Google App Engine cloud platforms can be deployed in PriverCloud instances with minor code modifications. Our evaluation of PriverCloud's performance indicates that PriverCloud introduces only modest overhead for its security measures even on constrained devices as well as realizes high availability by efficiently monitoring the reachability of cloud services and automatically recovering from detected failures of devices and cloud services within few seconds.

In this chapter, we focused on the research question on how *users* can preserve their privacy when interacting with cloud services. Notably and in contrast to the other contributions presented in this dissertation (cf. Chapters 3 to 5), we only focus on cooperation between different users and deliberately do not rely on cooperation from the other actors in the cloud computing landscape, most notably cloud service and infrastructure providers. To achieve this goal, the work presented in this chapter mainly helps in overcoming the inherent centrality of cloud computing as well as users' missing control as core problems for privacy in cloud computing. PriverCloud does not require any cloud services to deliver its functionality and hence counters the risks resulting from a centralized cloud computing landscape. Likewise, users explicitly decide whom they trust to faithfully operate their cloud services and thus have a high level of control over the delivery of their cloud services.

Besides mainly focusing on these two core problems, we partly address the core problem of technical complexity and missing transparency, since users know which devices constitute their PriverCloud instance as well as who owns and operates these devices. By completely moving away from the traditional centralized cloud deployment model, we propose an arguably quite radical approach to delivering a specific class of cloud services with high privacy requirements by shifting the execution of services from traditional cloud infrastructure to devices operated by trustworthy individuals such as a user's relatives and close friends. Yet, we believe that this approach nicely complements our other approaches to account for privacy in the cloud computing landscape, especially by explicitly addressing also extraordinary privacy-skeptical users.

7

Conclusion

Cloud computing is a powerful concept to make service delivery on the Internet more flexible, efficient, and reliable, most notably by offering the possibility to automatically scale the amount of utilized resources based on the current demand. As cloud computing offers numerous advantages, for both the operators of services on the Internet as well as for the customers of these services, there are clear incentives for shifting the delivery of services from own infrastructure to the cloud.

Cloud computing, however, also has its drawbacks, one prominent among them being the introduction of serious privacy challenges. These privacy challenges mainly originate from four core problems (cf. Section 1.1.3): (i) technical complexity and missing transparency resulting from the immanent abstraction of resources, (ii) opaque legislation with respect to the jurisdiction that applies to users' data, (iii) inherent centrality of the cloud computing market where a small number of providers jointly dominate the market, and, as a result, (iv) missing control of users over their data when it is handled in the cloud. Overcoming these challenges is key to secure the success of cloud computing and hence to allow a wide range of corporate and private users to profit from the advantages of cloud computing without having to sacrifice their privacy to a large extent.

In this dissertation, we addressed the challenge of accounting for privacy in cloud computing from a technical perspective. We first observed that it is insufficient to solely focus on single actors in the cloud computing landscape to overcome these inherent core problems for privacy. Consequently, we turned our focus to cooperation—either between different actors or between users. Hence, we proposed technical approaches that rely on cooperation where each of the actors in the cloud computing landscape contributes the technical means they have under their control to overall increase privacy. To this end, we formulated three research questions: (i) how infrastructure providers can support service providers and cloud users in executing control over privacy, (ii) how service providers can build privacy-preserving

cloud services on top of cloud infrastructure, and (iii) how users can preserve their privacy when interacting with cloud services. These research questions guided us through the individual contributions of this dissertation which target the different actors in the cloud computing landscape and address individual aspects underlying these three research questions.

In the remainder of this chapter, we revisit our contributions and the achieved results (Section 7.1), discuss how these contributions address the core problems of privacy in cloud computing (Section 7.2), summarize the impact of our work so far (Section 7.3), identify promising future research directions (Section 7.4), and conclude this dissertation with some final remarks (Section 7.5).

7.1 Contributions and Results

We addressed the three research questions of this dissertation by providing four distinct contributions. In the following, we summarize these contributions, our main results, and discuss how our contributions answer the individual research questions.

7.1.1 Raising Awareness for Cloud Usage

Our first contribution targeted the research question on how users can preserve their privacy when interacting with cloud services and was motivated by the observation that users are often unaware of their exposure to cloud services when using everyday technology such as email, mobile apps on smartphones, and IoT devices. To overcome this situation, we strived to uncover this cloud usage and raise users' awareness of the resulting privacy risks, hence empowering to take appropriate countermeasures. Alongside two deployment domains, cloud-based email and mobile apps on smartphones, we have shown how exposure to cloud services can be detected. Additionally, we presented an approach to realize privacy-preserving comparisons that enables users to put their cloud usage into context by comparing against their peers.

For cloud-based email as our first deployment domain, we presented MailAnalyzer, which dissects protocol headers of received emails to uncover exposure to cloud services. We applied MailAnalyzer to perform large-scale measurements of the email infrastructure used when sending email as well as to detect the exposure to cloud services caused by millions of authentic received emails. Our results revealed that users' privacy is indeed impacted by the exposure to cloud services, especially since the usage of cloud services often cannot be observed by less technically proficient users simply by looking at the sender or receiver information of an email.

Likewise, for mobile apps on smartphones as our second deployment domain, CloudAnalyzer passively analyzes the network traffic of mobile apps on off-the-shelf Android smartphones to detect communication with cloud services. We evaluated the cloud usage of mobile apps based on CloudAnalyzer in a user study with volunteers, by crawling the most popular mobile websites, and by comparing the most popular apps across the five countries with the highest mobile app usage. CloudAnalyzer

uncovered that nearly all apps connect to cloud services, with an average number of more than three utilized cloud services per app, while about one-third of the apps under study communicate exclusively with cloud services.

To enable individual users to put their cloud usage into perspective, we adapted the concept of comparison-based privacy [ZHHW15] to offer users the option to anonymously compare themselves with their peer groups and based on the comparison result, change their behavior to preserve their privacy. We introduced a privacy proxy that obliviously computes noisy aggregate cloud usage statistics using secure computations without anyone learning the contributions of individual users in cleartext. We performed a feasibility study and found that our approach achieves a reasonable trade-off between privacy protection and utility, i.e., accuracy of statistics.

In summary, the results derived from our first contribution have shown that users are indeed exposed to different cloud services. Our contribution provides users with transparency over their cloud usage through individual statistics and by contextualizing these statistics through anonymous comparisons with peers. This contribution is not only valuable in itself to support users with more transparency over their cloud usage but also provides a strong motivation for more privacy-friendly cloud computing as realized by the contributions of this dissertation.

7.1.2 Data Handling Requirements-aware Cloud Infrastructure

With our second contribution, we addressed the research question on how infrastructure providers can support service providers and cloud users in having control over privacy. To this end, we proposed a data handling requirements-aware cloud infrastructure in which users annotate their data with data handling requirements before sending them to the cloud. This approach enables users to express their privacy expectations and equips cloud providers with the technical means to respect these expectations when delivering their services. Within the scope of this dissertation, we focused on providing the functionality for supporting privacy requirements and did not specifically target the orthogonal challenge of providing technical guarantees that cloud providers indeed respect these requirements. As a foundation for realizing such a cloud infrastructure, we developed a compact privacy policy language that allows users to express their privacy expectations and devised a cloud storage system that assigns data to storage nodes based on these privacy expectations.

CPPL, our compact privacy policy language enables users to express their privacy requirements in a textual, human-readable policy and subsequently compresses this policy using standardized domain knowledge. Thus, CPPL achieves size and processing efficient compression of privacy policies which, unlike related work, can be directly used for interpretation at cloud nodes without requiring prior decompression. Indeed, our evaluation of CPPL has shown that we are able to realize huge policy size savings by up to two orders of magnitude when compared to related work and hence can realize per-data item privacy policies. Furthermore, a public cloud node can interpret tens of thousands of compressed policies per second, sufficient for handling data in real-world use cases.

Based on privacy requirements expressed with CPPL, PRADA, our data handling requirements-aware cloud storage system, stores data only on those cloud nodes that fully comply with the attached requirements. We implemented PRADA for the popular cloud storage system Cassandra and thereby illustrated the applicability and feasibility of our approach. Our performance evaluation of PRADA revealed that support for data handling requirements in cloud storage systems moderately increases query completion times and adds only a constant small storage overhead. Notably, PRADA manages to keep the load of the cluster as balanced as possible under given privacy constraints and does not affect the performance when storing data without attached data handling requirements.

The results we derived from our second contribution showcase the feasibility of a data handling requirements-aware cloud infrastructure. With CPPL, we presented a approach for users to specify their privacy requirements on a per-data item level to make cloud providers aware of their privacy demands. Likewise, with PRADA we provided cloud providers with the possibility to store data only on cloud nodes that fulfill users' requirements. Consequently, we realize user control over data as cloud providers can now fulfill user demands when assigning data to storage nodes.

7.1.3 Privacy-preserving Cloud Services for the Internet of Things

For our third contribution, we studied the research question of how service providers can build privacy-preserving cloud services on top of cloud infrastructure. To this end, we selected the IoT as an exemplary application domain for cloud services because of the high privacy requirements resulting from sensitive information often sensed and collected by IoT devices. In this setting, we proposed two approaches to aid service providers with performing security operations and hence ease the processing of protected IoT data stored in the cloud and the securing of configuration, authorization, and management of devices and networks in the cloud-based IoT.

To relieve service developers from having to implement the security functionality necessary to access protected IoT data on their own, we proposed SCSlib, a security library that transparently handles the security functionality required for encrypting protected IoT data in the cloud. Hence, we support domain specialists who typically do not specialize in security to realize privacy-preserving cloud services for the IoT based on a cryptographically enforced access control system for sensitive data. We evaluated SCSlib on public cloud infrastructure to confirm the feasibility of our approach. Our evaluation has shown that especially SCSlib's caching scheme for cryptographic keys helps to improve processing times when accessing protected IoT data in the cloud in sequential or random order.

To expand privacy protection from mere data to the control of IoT devices and federated IoT networks in the cloud-based IoT, we introduced D-CAM. Our proposed design of D-CAM comprises a distributed architecture that cryptographically secures messages in a tamper-resistant log such that only authorized entities can issue control operations, e.g., changing the configuration of an IoT device. As a result, a dishonest cloud provider cannot tamper with the potentially safety-critical

IoT devices managed on top of its infrastructure. Our evaluation of the overheads introduced by D-CAM has shown that the processing, storage, and communication costs of D-CAM are reasonable for the provided additional level of security. Furthermore, to additionally protect privacy-relevant information potentially revealed by configuration, authorization, and management messages, D-CAM can also be used to realize the key management necessary for the confidentiality protection of control messages.

To summarize, the results presented in the scope of this contribution support cloud service providers and developers in protecting users' privacy in the context of cloud computing. With SCSlib, we realized support for service developers in incorporating the functionality required to access protected IoT into their cloud services. Likewise, with D-CAM, we enabled support for securely realizing federated IoT networks on top of untrustworthy cloud infrastructure.

7.1.4 Decentralizing Individual Cloud Services

Finally, with our fourth contribution, we revisited the research question of how users can preserve their privacy when interacting with cloud services. This time, however, we consider a scenario where users distrust cloud providers to handle their data appropriately and instead cooperate with each other to realize a decentralized deployment model for certain types of cloud services. More specifically, we identified that individual cloud services, e.g., calendar and contact synchronization, do not necessarily require the enormous scalability offered by cloud computing. Hence, we asked ourselves whether it is possible to shift such services from untrusted public cloud infrastructure to infrastructure contributed by trusted individuals such as a user's family and close friends.

With PriverCloud, we presented a technical approach as an answer to this question. PriverCloud deploys cloud services to a decentralized peer-to-peer network built over idle resources on off-the-shelf devices in home networks. To ease the migration away from public cloud infrastructure, PriverCloud offers support for cloud services originally developed for the popular AppScale or Google App Engine cloud platforms. Our evaluation of PriverCloud indicated that even resource-constrained devices can easily handle the required security measures. Likewise, we have shown that our efficient approach for monitoring cloud services enables PriverCloud to recover from device, network, and service failures within seconds.

The results obtained from our fourth contribution highlight that certain scenarios allow for the replacement of traditional centralized cloud infrastructure with a system built on top of cooperation between users. As a result, we were able to forgo any assumptions on trust into cloud providers and instead rely on social trust. Furthermore, we have shown how trusted hardware can be used to further alleviate required trust. While completely moving away from centralized cloud infrastructure constitutes a quite radical approach, we consider this approach a valuable addition to our portfolio of contributions to account for privacy in the cloud computing landscape, especially for cloud services with extremely high privacy requirements or users who are very skeptical with respect to privacy.

	Techn. Complex. & Miss. Transp.	Opaque Legislation	Inherent Centrality	Missing Control
Raising Awareness for Cloud Usage	●	◐	◐	◐
Data Handling Requirements-aware Cloud Infrastructure	○	●	◐	●
Privacy-preserving Cloud Services for the IoT	◐	○	◐	●
Decentralizing Individual Cloud Services	◐	○	●	●

Table 7.1 The individual contributions presented in this dissertation comprehensively cover the complete space spanned by the core problems of privacy in cloud computing. A contribution addresses (●), partially addresses (◐), or does not specifically address (○) a privacy problem.

7.2 Core Problems Revisited

The contributions presented in this dissertation are designed to overcome the four core problems for privacy in cloud computing (cf. Section 1.1.3) and hence account for privacy in the cloud computing landscape. In the following, we highlight how these problems were addressed by the contributions of this dissertation. We summarize the mapping of contributions to the core problems they address in Table 7.1.

Technical Complexity and Missing Transparency

The problem of technical complexity and missing transparency of cloud computing mainly results from the abstraction of resources which hides the complexity of cloud services as well as from the indirect use of resources due to subcontracting (cf. Section 1.1.3). In this dissertation, we addressed this problem primarily by *raising awareness for cloud usage*. To this end, we provide users with statistics on their individual exposure to cloud services when using email services and mobile apps on smartphones. Furthermore, we enable users to contextualize their cloud usage through anonymous comparisons with their peer groups.

Consequently, our contribution creates transparency over the utilization of cloud-based services and provides insights into the technical realization of cloud services, e.g., which infrastructure a cloud service uses, information that has not been available to users so far. Partly, also other contributions of this dissertation address the problem of technical complexity and missing transparency. By cryptographically enforcing the access to IoT data and devices, our contribution to provide *privacy-preserving cloud services for the IoT* gives users transparency over who has access to their IoT data and can control their devices. Likewise, *decentralizing individual cloud services* results in a conceptually less complex system, as the technology stack is rather small, and provides transparency, as users have full knowledge of which devices are used to provide their cloud services and who operates these devices.

Opaque Legislation

As a result of the technical complexity and missing transparency of cloud computing, the jurisdiction that applies to users' data is often unclear. As this jurisdiction defines who, e.g., government agencies, can gain access to stored and processed data under which conditions, users cannot control or protect against third party access to their data (cf. Section 1.1.3). For users in the European Union, this situation is likely to change as the new GDPR [GDPR16] is applicable whenever the user whose data is being processed is based in the EU. Still, the abstraction of resources in cloud computing makes it difficult even for these users to actually execute their right to privacy. In the scope of this dissertation, we mainly addressed this problem by proposing *data handling requirements-aware cloud infrastructure*. To this end, we created a mechanism for users to specify their privacy requirements, most notably with respect to the applicable legislation, before sending their data to the cloud and enabled cloud providers to respect these requirements when assigning data to storage nodes. To a lesser degree, also *raising awareness for cloud usage* tackles the problem of opaque legislation since we can partly uncover the location to which data was sent. Hence, users can gain information on the jurisdiction applicable to their data and consequently, e.g., use a different mobile app or email service.

Inherent Centrality

The problem of inherent centrality of cloud computing is manifested by a few cloud providers that jointly dominate the market. As a result, cloud services become a valuable target for attackers and law enforcement agencies (cf. Section 1.1.3). Furthermore, in a centralized market, users have only a very limited choice to select, e.g., more privacy-friendly, cloud providers. We proposed *decentralizing individual cloud services* as our main answer to this problem. With this contribution, we enable users to eliminate any dependencies on traditional cloud providers and instead rely on resources provided by trusted entities in a peer-to-peer manner to break up the inherent centrality of the cloud computing landscape. Also our other three contributions partly addressed the problem of inherent centrality. By *raising awareness for cloud usage*, we also make users aware of the inherent centrality of cloud services. Furthermore, the privacy policy language proposed as part of our *data handling requirements-aware cloud infrastructure* can also be used to automatically choose between different cloud providers based on privacy requirements, hence easing comparability of cloud providers. Likewise, our security library proposed for *privacy-preserving cloud services for the IoT* increases interoperability between cloud services and thus eases the migration away from centralized cloud providers.

Missing Control

The previous three core problems for privacy in cloud computing culminate in users' missing control over their private data after it is sent to the cloud. Any data that leaves the control of the user can be unauthorizedly shared with third parties, utilized against the user's intention, or handled in violation of legal requirements. This

missing control is especially problematic, as the transfer of data out of the control of a user often goes unnoticed (cf. Section 1.1.3). All contributions presented in this dissertation address the problem of missing control. First, our *data handling requirements-aware cloud infrastructure* allows users to specify their privacy requirements to stay in control over their data after it left their immediate influence. To realize *privacy-preserving cloud services for the IoT* that provide users with control over their data, we proposed to protect the access to IoT data and the control of IoT devices using cryptographic measures. From a different perspective, when *decentralizing individual cloud services*, users can explicitly decide which other users they trust with the operation of their cloud services, hence providing them with a high level of control over their privacy. To a lesser extent, our proposed contribution to *raise awareness for cloud usage* provides users with the necessary information to regain control over their data, e.g., by uninstalling privacy-invasive mobile apps.

In summary, our contributions take different views on tackling the core problems for privacy in cloud computing. While they certainly do not completely solve all privacy challenges of the cloud computing paradigm, they provide important steps forward towards accounting for privacy in cloud computing. This progress especially manifests when combining the different contributions presented in this dissertation and hence incorporating all the actors involved in delivering cloud services.

7.3 Impact of Our Work

The individual contributions of this dissertation have been published and presented at scientific venues. As a result, the contributions that form this dissertation provided the motivation and basis for other research efforts. Furthermore, we partly released the software underlying our contributions as open source software to ease the reproducibility of our results and to provide a foundation for further research endeavors. In the following, we summarize the resulting impact of our work so far.

7.3.1 Impact of Publications

Our motivation for realizing *data handling requirements-aware cloud infrastructure* [HGKW13, HHW13a], has inspired numerous research efforts. First, different researchers [AEÖ+14, BGR+15] discuss policy languages to realize our proposed data handling requirements-aware cloud stack. Following our problem statement of supporting data handling requirements in cloud computing, Maenhaut et al. [MMOT14, MMOT15, MMV+15, MMV+17] propose an abstraction layer to support custom data handling policies for each user, ultimately evolving into the concept of a software defined storage system. Likewise, Pasquier and Powles [PP15] apply our notion of data handling requirements to the concept of information flow control. Furthermore, Ayache et al. [AEF15] suggest attaching a privacy policy to a set of data to make policy handling more efficient. Singh et al. [SPB+16] note that meeting our goal to account for data handling requirements becomes even more complex in the context of the even more dynamic IoT.

Different researchers follow our motivation for the relevance of realizing *privacy-preserving cloud services* for the IoT [HHK+14, HHK+16], e.g., in the context of healthcare and monitoring of elderly [Hos16, ARLT17, MJ17, MRAA17], smart homes [YDAJ15, BJD16, COTC17], smart cities [BRM16, RBM16, WC16], and location privacy [SCR+17]. From a more conceptual standpoint, other researchers extend upon the concepts underlying our individual contributions [HHM+13, HHMW14, HBHW14, HWM+17]. Singh et al. [SPB15] propose to extend access control with a data-centric control mechanism for IoT data using information flow control. Likewise, Funke et al. [FDW+15] suggest extending our work to realize an end-to-end privacy architecture for the IoT. Perra [Per15] adapts our trust point-based security architecture to the context of protecting personal media content. Pacheco et al. [PAS17] evolve our architecture to work without dedicated gateways and hence target the challenge of preserving privacy when IoT devices directly communicate with the cloud. From a different perspective, Kashef et al. [KYKH16] propose a decision support tool for IoT service providers to choose between different cloud providers, e.g., with respect to privacy requirements. Leveraging a decentralized access control mechanism similar to our proposal, Ko et al. [KJK16, KJK17] provide the foundation for virtualizing IoT services. Liang et al. [LWBL17] specifically deepen the study of the problem of conflicting commands in the cloud-based IoT, while Khan and Salah [KS17] further develop our idea of a tamper-resistant message log for securely managing the IoT.

Finally, based on our idea to *decentralize individual cloud services* by operating them on infrastructure provided by a user's family and close friends, Baig et al. [BFN16, BFN18] propose to deploy cloud services in an existing community cloud where everyone can contribute and consume computing resources.

7.3.2 Impact of Open Source Activities

To enable the reproducibility of our results as well as to lay the foundation for further research efforts, we selectively released the software and data we used to verify the feasibility of the approaches that form our contributions in this dissertation under open source licenses. In the following, we briefly recap on these efforts and report on the impact of these activities where relevant.

For the approaches underlying our contribution for *raising awareness for cloud usage*, we released the source code, our compiled detection patterns for cloud services, as well as anonymized and aggregated study results for MailAnalyzer under the MIT license[10]. Likewise, we provide the source code of our CloudAnalyzer app and the additional detection patterns for mobile cloud services under the GNU GPL license (version 3)[11]. We expect that especially the patterns we compiled to detect communication with cloud services are of relevance for other researchers as well as practitioners from industry.

In the context of our contribution for realizing *data handling requirements-aware cloud infrastructure*, we released the source code as well as a library binding of

[10] `https://github.com/COMSYS/MailAnalyzer`
[11] `https://github.com/COMSYS/CloudAnalyzer`

CPPL as open source under the Apache license (version 2)[12]. In the context of the SSICLOPS project, CPPL has been integrated by researchers and practitioners in a number of industry-driven use cases [HKP+18]: At the cloud infrastructure layer, CPPL has been integrated with OpenStack to provide customers with control over the management of resources and applied to realize policy-compliant setup of network connections. With respect to the platform layer, CPPL was applied to realize privacy policy-aware data management both for XRootD at CERN to store data on high-energy particle collisions as well as for the in-memory database Hyrise. Furthermore, the ISP Orange Polska is working on applying CPPL to make the virtualization of their customers' home-gateways policy-compliant. Finally, at the software layer, F-Secure plans to extend its Security Cloud with policy-aware analysis of customers' files based on CPPL.

As a foundation for realizing *privacy-preserving cloud services for the IoT*, we released the source code of SCSlib under the open source MIT license[13]. Furthermore, we created a documentation of the underlying protocol used for encoding IoT data and our security measures [HHMW16]. We have been in contact with researchers from Alexandria University (Egypt) and Concordia University (Canada) who are currently working on applying SCSlib for their research. Lately, we started discussions on adapting our trust point-based security architecture to support the development of a user-controlled ecosystem for the sharing of personal data [MMZ+17].

7.4 Future Research Directions

During the work on this dissertation, we discovered a wide range of directions for promising future research. We already discussed specific future research directions that deepen our individual contributions in the respective chapters. In the following, we discuss promising research directions that are not directly tied to a single contribution and hence significantly extend the scope of this dissertation.

7.4.1 User Acceptance

The contributions presented in dissertation aim at providing technical approaches to increase the level of privacy when using cloud services. However, to be successful, such approaches eventually need to be adopted by users, hence requiring that users accept the technical approaches and their implementations. For example, as of now, we do not know whether an approach such as our proposed data handling requirements-aware cloud stack (cf. Chapter 4) can be outright used by less technically proficient users. Based on a study of user acceptance, we could, for example, derive the necessity to further abstract from the technical specification of our privacy policy language, e.g., by providing GUI support or even only a limited set of predefined policies from privacy experts.

[12] `https://github.com/COMSYS/cppl`
[13] `https://code.comsys.rwth-aachen.de/redmine/projects/scslib`

From our perspective, studying user acceptance of technical systems that aim to enhance privacy comes with three challenges: (i) temporal dependencies, where a technical system needs to be built first before users' interaction with this system can be studied, (ii) strong influences of the user interface on the user experience and hence acceptance of a technical system, a factor not a core focus of our work, and (iii) the necessity to clearly communicate the technical properties and privacy guarantees of a system to users, again not one of our essential targets. We hence did not cover the user acceptance of our proposed approach within the context of this dissertation. Still, we collaborated on several occasions with researchers from the sociology department to perform initial steps into this direction [EHH+14, HHK+14, HHK+16, HKH+16]. We refer to the publications of our collaborators [EHKR14, KDZ18] for further insights into initial results regarding the user acceptance of our contributions presented in Chapters 3 and 5. Still, further collaborative research efforts are required to transform the technical results of this dissertation into systems actually usable for private users.

7.4.2 Accountable Cloud Computing

Our contributions, especially those presented in Chapters 4 and 5, assume that the different actors in the cloud computing landscape all have a genuine interest in accounting for privacy. This assumption is often valid as cloud providers have to adhere to legal regulatory frameworks, are afraid of undesired consequences such as non-acceptance of services or damage to reputation, and see business opportunities in accounting for privacy (cf. Section 1.1.2). Hence, our work in this dissertation was mainly motivated by the goal to provide functional improvements over the status quo with respect to accounting for privacy in the cloud computing landscape. Still, private and especially corporate users might want to hold cloud providers accountable for delivering privacy functionality as promised, referred to as accountable cloud computing [Hae10]. Different technical approaches exist to tackle this challenge, and we briefly discuss the two most promising ones to turn the contributions of this dissertation into accountable systems in the following.

First, hardware security functions such as ARM TrustZones, Intel software guard extensions (SGX), or trusted platform modules (TPMs) found a root of trust on hardware components. In the context of cloud computing, such approaches have been used to realize, e.g., secure storage and processing of users' confidential data in secure execution environments [IKC09], trustworthy data analytics where both code and data are kept private [SCF+15], as well as confidentiality and integrity for third party coordination services [BWG+16]. Considering the contributions presented in this dissertation, we already optionally leverage TPMs to increase trust in the devices used to realize decentralized individual cloud services (cf. Chapter 6). For future research directions, hardware security could be used to enhance our data handling requirements-aware cloud stack (cf. Chapter 4) with accountability functionality. As a result, users and auditors would be empowered to verify that cloud providers correctly perform the evaluation of privacy policies (cf. Section 4.2.2) or that selected storage nodes indeed possess the promised properties (cf. Section 4.3.2). Likewise, in

the context of privacy-preserving cloud services for the IoT (cf. Chapter 5), hardware security functionality can be used to ensure that cloud services are effectively isolated against each other and the host system (cf. Section 5.2.1.2). From a completely different perspective, accountability mechanisms could also be applied to ensure the integrity of measurements provided by users for the comparison of cloud usage (cf. Section 3.4).

Besides anchoring accountability in trusted hardware components, also secure computation as a purely software-based approach can be used to realize accountable cloud computing. More specifically, secure two-party computation, which is the secure computation framework of particular relevance in the context of cloud computing, enables two mutually distrusting parties to compute a joined functionality without the need to reveal the respective private inputs to the other party [ZMHW15]. Theoretically, any computable function can be realized using secure two-party computation, e.g., based on fully homomorphic encryption [DJ10]. In the context of cloud computing, secure two-party computation has been applied, e.g., to encrypt both data and programs before performing computations in an untrusted public cloud [BNSS11], to realize data deduplication for encrypted data stored in the cloud [NWZ12], to preserve privacy during biometric authentication [CEL+14], or to perform privacy-preserving outsourced genetic disease testing [ZPH+17].

These examples show that secure computation is a valuable asset when the primary goal is to protect the confidentiality of information during computation in an untrusted cloud. While we observe that there is still a long way to go for the general feasibility of applying secure computation to arbitrary use cases [ZMHW15] and certain classes of cloud services even cannot be realized with cryptography alone [DJ10], these approaches show that special use cases can be realized with sufficient efficiency. Within the scope of this dissertation, we already applied secure computation to realize privacy-preserving comparisons of cloud usage (cf. Section 3.4). Concerning the other contributions presented in this dissertation, we consider it most promising to enhance privacy-preserving cloud services for the IoT (cf. Chapter 5) with secure computation such that cloud services no longer need access to users' data in plain text to realize their functionality. In fact, our flexible mechanism for encoding IoT data and security mechanisms (cf. Section 5.2.2.3) already allows to encode data encrypted for secure computations, e.g., when using homomorphic encryption.

7.4.3 Beyond Cloud Computing

While the concept of cloud computing matures, we observe an increasing trend of shifting computation closer to the users. In edge computing, computation is pushed to the edge of networks [SCZ+16, SD16], while fog computing goes one step further and realizes computation on devices in the same local network [SW14, VR14]. The core motivation for both deployment models is to decrease the amount of data sent to and received from the cloud, reduce communication latency, and thus realize shorter response times compared to cloud services in remote data centers. Intuitively, moving computation closer to the user, and thus her trusted domain, might help in overcoming the privacy problems of cloud computing (cf. Section 1.1.3). However,

to a certain extent, these privacy problems persist: Edge and fog computing deployments are still technically complex and lack transparency while offering users little control. Furthermore, edge and especially fog computing introduce additional privacy challenges. For example, the number of actors involved in delivering a service can increase noticeably, especially in the case of fog computing where it is envisioned to deploy computation also to devices of other, untrusted users [VR14].

While not specifically designed and evaluated for these evolving deployment domains, the contributions presented in this dissertation—after further research efforts—can prove beneficial to tackle privacy problems of edge and fog computing deployments as well. Also in the context of edge and fog computing, we consider it feasible and promising to analyze network traffic as a foundation to raise users' awareness on their usage of edge and fog resources (cf. Chapter 3), hence lifting the fog of missing transparency in these deployment domains. By adapting the concept of data handling requirements-aware cloud infrastructure (cf. Chapter 4) to edge and fog computing, users could be provided with more control over the realization and placement of services that operate on their data. Our compact privacy policy language (cf. Section 4.2) already affords for other deployment domains and hence future research in this direction mainly needs to be concerned with the distinct privacy requirements of users in these scenarios and how these can be mapped to a technical system. Likewise, our contribution to realize privacy-preserving cloud services for the IoT (cf. Chapter 5) presents a starting point to provide users with control over the access to their data in edge and fog computing deployments (cf. Section 5.2) as well as to enable the secure management of such deployments in a decentralized manner (cf. Section 5.3). Furthermore and especially when moving away from centralized cloud deployments, considering accountability, as discussed above for cloud computing, becomes a prime candidate for future research.

7.4.4 Beyond Privacy

This dissertation deliberately focused on the challenge of accounting for the privacy of users when interacting with cloud services. However, we observe an increasing trend of companies outsourcing their own (and not only their users') data as well as business intelligence to the cloud, e.g., in the context of the Industrial IoT and cyber manufacturing systems [JBM+17, GHW+19]. This trend is fueled by the vision of Industry 4.0, the alleged fourth industrial revolution, where an increased amount of data collection as well as cooperation and coordination of production steps across individual factories or even companies is envisioned [LFK+14]. Hence, one promising future research direction lies in the transition of our contributions from targeting the privacy of users towards the more holistic problem space of corporate secrecy and secure information sharing between corporations in Industry 4.0 [SWW15]. These challenges naturally arise with increased data collection, outsourcing of data storage and processing to the cloud, as well as cooperation and coordination across companies. Notably, the concerns here are not restricted to the mere confidentiality of collected data, but rather also involve, e.g., the interaction patterns of different companies or machine learning models to optimize production processes.

We recently started our work to also account for these concerns, especially in the context of cloud computing, and we are convinced that the contributions presented in this dissertation provide a valuable starting point for these efforts. For example, the trust point-based security architecture underlying our privacy-preserving cloud services (cf. Section 5.2.2.2) could also be applied to protect production data.

Yet, to fully embrace the advantages promised by Industry 4.0, access control decisions likely have to be taken dynamically and automatically, which we believe could be realized based on accountability mechanisms such as trusted hardware or secure computations as discussed above. Likewise, our compact privacy policy language (cf. Section 4.2) could be employed in a decentralized system in which companies automatically establish who should cooperate with whom under which conditions. For example, an external supplier could provide access to the raw production data of an individual component only under certain conditions. Finally, a more general approach for anonymous comparisons (cf. Section 3.4.3) can provide the foundation of anonymous performance benchmarking, e.g., enabling a company to compare its production output against competitors utilizing the same machine model without having to disclose own confidential production data.

7.5 Final Remarks

This dissertation proposed different technical approaches to account for privacy in the cloud computing landscape with the motivation to enable more private and corporate users to benefit from the advantages of cloud computing without the often inherent need to sacrifice privacy. To this end, we specifically and deliberately focused on the advantages offered by cooperation between the different actors in the cloud computing landscape. The results derived during the course of this dissertation highlight that it is indeed promising and feasible to leverage cooperation to derive technical systems that improve users' privacy in the cloud computing landscape. This aspect is further supported by the initial adoption and further evolution of the ideas and approaches presented in this dissertation by other researchers as well as practitioners from industry.

However, to fully address and solve the pressing problem of privacy in cloud computing, much larger efforts are required to comprehensively integrate different and often conflicting views such as legislation, user acceptance, and business perspectives with technical approaches such as ours. With the contributions presented in this dissertation, we strongly believe to provide valuable technical foundations to eventually achieve this goal and relieve users of having to choose between their privacy and the benefits of cloud computing. Finally, we hope that our contributions serve as an inspiration for future research in the area of cloud computing privacy and beyond.

A

Appendix

A.1 Full Example of a CPPL Policy

In Section 4.2.2, we presented the compression of a policy together with a reasoning of our design decisions. To fully embrace the inner workings of CPPL, we now present a detailed example for the specification and compression of a privacy policy as well as a description of its evaluation at a cloud node.

Specifying a Policy with CPPL

Listing A.1 shows the textual representation of the policy which we compress with the help of the domain parameters specification (CPPL dialect) given in Listing A.2. The policy is an extended version of the policy discussed in Section 4.2.2.1 (cf. Listing 4.1) which, in the extended version, additionally incorporates a redundant variable (`"CompanyA"`) as well as a redundant relation (`encryption = true`) to showcase the corresponding compression mechanisms.

Compressing a Policy with CPPL

The resulting compressed policy is depicted in Listing A.3. First, the policy header encodes the version (23 in our example) used by this policy and hence the applicable CPPL dialect. The formula stack encodes the boolean operands *OR* (10) and *AND* (11). Furthermore, it refers to relations on the relation stack: either next relation (00) or to a relation at a specific position on this stack (011<*position*>). The *position* is specified as index of the relation on the relation stack starting with index 0 for the first relation. Thereby, the number of bits to encode the position of a variable is fixed and can be derived from the domain parameters (8 bits in our example). The end of the formula stack is signaled by the bit sequence 010.

```
  provider != "CompanyA"
& ( tenant != "CompanyA" | encryption = true )
& log_access = true
& deleteAfter(1735693210)
& backupHistory("1M")
& replication >= 2
& ( location = "DE" | (location = "EU" & encryption = true) )
```

Listing A.1 Extended version of the previously used CPPL policy (Listing 4.1). The extended version features a redundant variable and a redundant relation to showcase their processing during compression.

```
{
  "version": 23,
  "relationPositionLen": 8,
  "variablePositionLen": 8,
  "variables": [
    { "name": "provider", "type": "string" },
    { "name": "tenant", "type": "string" },
    { "name": "log_access", "type": "boolean" },
    { "name": "deleteAfter", "type": "function", "parameters":
      [ "int32" ] },
    { "name": "backupHistory", "type": "function", "parameters":
      [ "string" ] },
    { "name": "location", "type": "string", "values":
      ["DE", "FR", "US", "GB", "NL", "EU"] },
    { "name": "encryption", "type": "boolean" },
    { "name": "replication", "type": "int32" }
  ]
}
```

Listing A.2 Underlying domain parameters specification (CPPL dialect) in JSON format.

Following the formula stack, the relation stack encodes the relations $=$ (000), $\neq$ (001), $<$ (010), $\leq$ (011), $>$ (100), $\geq$ (101), $= True$ (110), $= False$ (111). Thereby, it refers to one or two variables (depending on the relation type) on the variable stack: either to the next variable (0) or to a variable at a specific position on this stack (1<*position*>). Again, the position is given as the index of the corresponding variable on the variable stack starting with index 0 for the first variable. The length of the position field is specified by the domain parameters (here we use 8 bits which allows for referencing variables with index up to 255—far more than required for the real-world policies in Section 4.2.3.2)

Finally, the variable stack encodes booleans (0000), variable identifiers that refer to variables specified in the domain parameters (0001), strings (0010), enumerated variables (0011), functions (0100), int64 (0101), int32 (0110), int16 (0111), int8 (1000), uint32 (1001), uint16 (1010), uint8 (1011), and double values (1100). Each of these type identifiers is followed by the actual value of the variable whose length is determined by the type, e.g., fixed to 8 bits for uint8 or terminated by a special symbol, e.g., for null-byte terminated strings.

```
Policy Header
  0000000000010111  version (23)
Formula Stack
  11                AND
  00                Next Relation
  11                AND
  10                OR
  00                Next Relation
  11                AND
  011 00000001      Reference to Relation at index 1
  00                Next Relation
  11                AND
  00                Next Relation
  11                AND
  00                Next Relation
  11                AND
  00                Next Relation
  11                AND
  00                Next Relation
  10                OR
  00                Next Relation
  00                Next Relation
  010               End of formula stack
Relation Stack
  001 0 0           ≠, Next Var, Next Var
  110 0             =True, Next Var
  001 0 1 00000001  ≠, Next Var, Reference to Variable at index 1
  110 0             =True, Next Var
  110 0             =True, Next Var
  110 0             =True, Next Var
  000 0 0           =, Next Var, Next Var
  000 0 0           =, Next Var, Next Var
  101 0 0           ≥, Next Var, Next Var
Variable Stack
  0001 001          ID 1 (tenant)
  0010              string
    01000011011011110110110101110000011000010110111001111001010000001
    00000000        "CompanyA"
  0001 110          ID 6 (encryption)
  0001 000          ID 0 (provider)
  0001 010          ID 2 (log_access)
  0100 011          Function, ID 3 (deleteAfter)
    01100111011101001001001110011010
                    int32 (value: 1735693210)
  0100 100          Function, ID 4 (backupHistory)
    00110001010011010000000
                    string "1M"
  0001 101          ID 5 (location)
  0011 101          enum value 5 ("EU")
  0001 101          ID 5 (location)
  0011 000          enum value 0 ("DE")
  0001 111          ID 7 (replication)
  1011 00000010     uint8 (value: 2)
```

Listing A.3 The resulting compressed policy representation is shown as a sequence of bits complemented by descriptive text for their respective meanings.

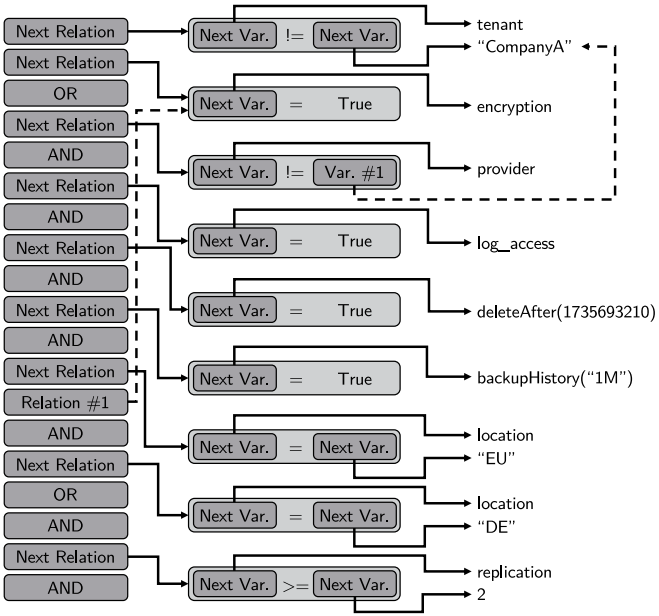

Figure A.1 Decompression of a policy during evaluation at a cloud node. First, the algorithm iterates over the formula stack to find the beginning of the relations, thereby pushing elements of the formula stack onto an interpretation stack (left) which yields the policy in reverse polish notation. In a second step, the algorithm evaluates the policy based on the reverse polish notation, i.e., it resolves and evaluates relations and applies the boolean operations to the corresponding results.

This encoding enables us to reduce the 180 byte textual encoding to a 42 byte representation of the policy. By doing so, we lay the foundation for efficient transmission and storage of data annotations.

Interpreting a Policy with CPPL

When a cloud node receives a data item, e.g., to store or process it, the node first must check if the desired action is possible given the requirements specified by the policy. Compressed data typically requires decompression before its processing. However, with CPPL we are able to omit a separate decompression step and instead efficiently integrate decompression into the interpretation of the policy (cf. Section 4.2.2.3). In the following, we describe the interpretation of CPPL policies in more detail based on our example.

At the beginning of the compressed policy, the header enables the matching algorithm to determine the domain parameters specification (CPPL dialect) that applies

to this policy. Following this, the formula stack encodes the boolean interconnection of relations in polish notation. During the matching process, the algorithm iterates over the formula stack until its end to find the beginning of the relation stack. Thereby, it sequentially pushes the content of the formula stack onto an interpretation stack. For our example, we depict this stack in Figure A.1 (left). When reaching the end of the formula stack, the interpretation stack contains the policy in *reverse* polish notation. This order is used for the actual interpretation of the policy.

To this end, the algorithm sequentially takes the next element from the top of the interpretation stack. This element may be a reference to a relation or a boolean operand. The typical case for relations is a reference to the *next* relation on the relation stack. In this case, the algorithm locates the next relation on the relation stack, resolves corresponding variables, interprets the relation, and stores the truth value. Here, values for variable identifiers are retrieved from the node parameters that define the properties of this cloud node, e.g., its location. In case of a reference to a *specific* relation (as identified by a relation position), we know that this relation has already been evaluated and the truth value can be reused (cf. reference to Relation #1 in Figure A.1).

When retrieving a boolean operand from the interpretation stack, the algorithm can directly apply it as the reverse polish notation ensures that the corresponding relations already have been interpreted. Furthermore, reverse polish notation ensures that the last element of the interpretation stack is a boolean operand whose application yields the final result of the interpretation process.

A.2 Latencies Between Cloud Nodes

As foundation for our evaluation of the applicability of PRADA (cf. Section 4.3.3.4), our cloud storage system that realizes compliance with DHRs, we measured the latency between different nodes of the Microsoft Azure Cloud using `hping3` [San06], a command-line oriented network testing tool. More specifically, we measured the pair-wise RTTs between nodes in the following regions of the Microsoft Azure Cloud [Mic16b]: `asia-east`, `asia-southeast`, `canada-east`, `eu-north`, `eu-west`, `japan-east`, `us-central`, `us-east`, `us-southcentral`, and `us-west`. We summarize the measured pair-wise RTTs between the ten nodes (one in each region) in Table A.1.

We observe that the RTT between different regions within North America lies in the range of 25.1 ms (`us-southcentral` → `us-central`) and 78.4 ms (`us-west` → `canada-east`). In contrast, the RTT for communication between different continents ranges from 179.0 ms (`asia-southeast` → `eu-west`) to 286.2 ms (`asia-east` → `eu-west`) between Asia and Europe, from 108.4 ms (`japan-east` → `us-west`) to 244.6 ms (`asia-southeast` → `canada-east`) between Asia and North America, and from 85.2 ms (`eu-west` → `us-east`) to 145.3 ms (`us-west` → `eu-west`) between Europe and North America. These results are in line with the results reported by Sanghrajka et al. [SMS11] for measurements of inter-region RTTs for Amazon Web Services' cloud offer performed in 2011.

	us-east	us-central	us-west	canada-east	us-southcentral
us-east	—	35.1 ms	66.7 ms	38.1 ms	31.9 ms
us-central	35.0 ms	—	41.1 ms	41.5 ms	25.8 ms
us-west	66.1 ms	42.2 ms	—	78.4 ms	35.1 ms
canada-east	38.3 ms	41.0 ms	78.0 ms	—	63.8 ms
us-southcentral	31.8 ms	25.1 ms	35.4 ms	62.8 ms	—
eu-west	85.2 ms	117.0 ms	145.0 ms	111.0 ms	113.2 ms
asia-east	203.8 ms	183.3 ms	157.9 ms	220.9 ms	174.2 ms
asia-southeast	220.9 ms	207.9 ms	175.8 ms	244.6 ms	191.4 ms
japan-east	153.7 ms	142.3 ms	108.4 ms	179.7 ms	124.7 ms
eu-north	85.9 ms	104.5 ms	141.6 ms	89.6 ms	121.2 ms

	eu-west	asia-east	asia-southeast	japan-east	eu-north
us-east	85.3 ms	213.6 ms	241.1 ms	179.5 ms	86.3 ms
us-central	116.6 ms	182.9 ms	208.6 ms	143.4 ms	105.1 ms
us-west	145.3 ms	159.4 ms	182.8 ms	115.6 ms	142.4 ms
canada-east	109.9 ms	220.6 ms	244.2 ms	178.8 ms	90.0 ms
us-southcentral	114.3 ms	175.0 ms	192.7 ms	124.7 ms	120.5 ms
eu-west	—	285.6 ms	179.7 ms	235.9 ms	24.5 ms
asia-east	286.2 ms	—	37.8 ms	51.9 ms	284.4 ms
asia-southeast	179.0 ms	38.5 ms	—	70.3 ms	180.0 ms
japan-east	236.0 ms	53.4 ms	70.8 ms	—	243.7 ms
eu-north	24.3 ms	283.7 ms	181.2 ms	243.7 ms	—

Table A.1 Round-trip times between cloud nodes in the different regions of Microsoft Azure.

Abbreviations and Acronyms

AES	Advanced Encryption Standard
API	application programming interface
AWS	Amazon Web Services
BGP	border gateway protocol
CA	Certificate Authority
CBC	cipher block chaining
CCM	counter with CBC-MAC
CDN	content delivery network
CPS	Cyber-physical Systems
CPU	central processing unit
CRUD	create, read, update, and delete
DHR	data handling requirement
DHT	distributed hash table
DNS	domain name system
DoS	denial of service
DRM	digital rights management
ECDSA	Elliptic Curve Digital Signature Algorithm
EU	European Union
FIFO	first in first out
GDPR	General Data Protection Regulation
GPS	Global Positioning System
GUI	graphical user interface

HIPAA	Health Insurance Portability and Accountability Act
HMAC	keyed-hash message authentication code
IaaS	Infrastructure as a Service
IETF	Internet Engineering Task Force
IMEI	International Mobile Equipment Identity
IoT	Internet of Things
IP	Internet Protocol
ISP	internet service provider
JSON	JavaScript Object Notation
JWE	JSON Web Encryption
JWK	JSON Web Key
JWS	JSON Web Signature
LRU	least recently used
LXC	Linux containers
MAC	message authentication code
MTU	maximum transmission unit
MX	mail exchange
NAS	network-attached storage
NAT	network address translation
NIST	National Institute of Standards and Technology
PaaS	Platform as a Service
PII	personally identifiable information
QCT	query completion time
QoS	quality of service
RSA	Rivest-Shamir-Adleman cryptosystem
RTT	round-trip time
SaaS	Software as a Service
SDK	software development kit

SDN	software-defined networking
SGX	software guard extensions
SHA	Secure Hash Algorithm
SLA	service level agreement
SME	small and medium-sized enterprise
SMTP	Simple Mail Transfer Protocol
SNI	server name indication
SSH	secure shell
SVM	support vector machine
TCP	Transmission Control Protocol
TLS	transport layer security
TPM	trusted platform module
VM	virtual machine
VPN	virtual private network
XML	Extensible Markup Language

Bibliography

[AB13] Veronika Abramova and Jorge Bernardino. NoSQL Databases: MongoDB vs Cassandra. In *Proceedings of the International C* Conference on Computer Science and Software Engineering (C3S2E)*, pages 14–22. ACM, 2013.

[ABF+04] Rakesh Agrawal, Roberto Bayardo, Christos Faloutsos, Jerry Kiernan, Ralf Rantzau, and Ramakrishnan Srikant. Auditing Compliance with a Hippocratic Database. In *Proceedings of the Thirtieth International Conference on Very Large Data Bases (VLDB)*, pages 516–527. VLDB Endowment, 2004.

[ABP09] Muhammad Ali, Laurent Bussard, and Ulrich Pinsdorf. Obligation Language and Framework to Enable Privacy-Aware SOA. In *Proceedings of the 4th International Workshop on Data Privacy Management (DPM)*, pages 18–32. Springer, 2009.

[Acc10] Rafael Accorsi. BBox: A Distributed Secure Log Architecture. In *Proceedings of the 7th European Workshop on Public Key Infrastructures, Services and Applications (EuroPKI)*, pages 109–124. Springer, 2010.

[ADBK10] Armen Aghasaryan, Marie-Pascale Dupont, Stéphane Betgé-Brezetz, and Guy-Bertrand Kamga. Privacy Data Envelops for Moving Privacy-sensitive Data. In *Proceedings of the W3C Workshop on Privacy and Data Usage Control*. World Wide Web Consortium, 2010.

[ADD+14] N. Asokan, Lucas Davi, Alexandra Dmitrienko, Stephan Heuser, Kari Kostiainen, Elena Reshetova, and Ahmad-Reza Sadeghi. *Mobile Platform Security*, volume 4(3) of *Synthesis Lectures on Information Security, Privacy, and Trust*. Morgan & Claypool, 2014.

[Ade16] Adestra. 2016 Adestra Consumer Adoption & Usage Study, 2016.

[AEF15] Meryeme Ayache, Mohammed Erradi, and Bernd Freisleben. Access Control Policies Enforcement in a Cloud Environment: Openstack. In *Proceedings of the 2015 11th International Conference on Information Assurance and Security (IAS)*, pages 26–31. IEEE, 2015.

[AEÖ+14] Monir Azraoui, Kaoutar Elkhiyaoui, Melek Önen, Karin Bernsmed, Anderson Santana Oliveira, and Jakub Sendor. A-PPL: An Accountability Policy Language. In *Proceedings of the 9th International Workshop on Data Privacy Management (DPM)*, pages 319–326. Springer, 2014.

[AFG+09] Michael Armbrust, Armando Fox, Rean Griffith, Anthony D. Joseph, Randy H. Katz, Andrew Konwinski, Gunho Lee, David A. Patterson, Ariel Rabkin, Ion Stoica, and Matei Zaharia. Above the Clouds: A Berkeley View of Cloud Computing. Technical Report UCB/EECS-2009-28, EECS Department, University of California, Berkeley, 2009.

[AFG+10] Michael Armbrust, Armando Fox, Rean Griffith, Anthony D. Joseph, Randy Katz, Andy Konwinski, Gunho Lee, David Patterson, Ariel Rabkin, Ion Stoica, and Matei Zaharia. A View of Cloud Computing. *Communications of the ACM*, 53(4):50–58, 2010.

[AGM10] Mohamed Al Morsy, John Grundy, and Ingo Müller. An Analysis of the Cloud Computing Security Problem. In *Proceedings of the APSEC 2010 Cloud Workshop*, 2010.

[AHA+14] Ismet Aktaş, Martin Henze, Muhammad Hamad Alizai, Kevin Möllering, and Klaus Wehrle. Graph-based Redundancy Removal Approach for Multiple Cross-Layer Interactions. In *Proceedings of the 2014 Sixth International Conference on Communication Systems and Networks (COMSNETS)*, pages 1–8. IEEE, 2014.

[AIM10] Luigi Atzori, Antonio Iera, and Giacomo Morabit. The Internet of Things: A survey. *Computer Networks*, 54(15):2787–2805, 2010.

[AKK12] Jose M. Alcaraz Calero, Benjamin König, and Johannes Kirschnick. Cross-Layer Monitoring in Cloud Computing. In Habib F. Rashvand and Yousef S. Kavian, editors, *Using Cross-Layer Techniques for Communication Systems*, pages 328–348. IGI Global, 2012.

[AKSX02] Rakesh Agrawal, Jerry Kiernan, Ramakrishnan Srikant, and Yirong Xu. Hippocratic Databases. In *Proceedings of the 28th International Conference on Very Large Data Bases (VLDB)*, pages 143–154. VLDB Endowment, 2002.

[AL07] Yonatan Aumann and Yehuda Lindell. Security Against Covert Adversaries: Efficient Protocols for Realistic Adversaries. In *Proceedings of the 4th Theory of Cryptography Conference (TCC)*, pages 137–156. Springer, 2007.

[Ald15] Fritz Alder. Distributed Storage for Secure Peer-to-Peer Clouds. Bachelor's thesis, RWTH Aachen University, April 2015.

[Ale16] Alexa. Actionable Analytics for the Web. `http://www.alexa.com/`, 2016. [Online, accessed 2016-07-06].

[And18a] Android. Intent – Android Developer. `https://developer.android.com/reference/android/content/Intent`, 2018. [Online, accessed 2018-07-01].

[And18b] Android. UI/Application Exerciser Monkey – Android Studio. `https://developer.android.com/studio/test/monkey`, 2018. [Online, accessed 2018-07-01].

[ANSF16] Muneeb Ali, Jude Nelson, Ryan Shea, and Michael J. Freedman. Blockstack: A Global Naming and Storage System Secured by Blockchains. In *Proceedings of the 2016 USENIX Annual Technical Conference (USENIX ATC)*, pages 181–194. USENIX, 2016.

[Apa18a] Apache Software Foundation. Apache Cassandra. `https://cassandra.apache.org/`, 2018. [Online, accessed 2018-07-01].

[Apa18b] Apache Software Foundation. Apache James – Java Apache Mail Enterprise Server. `http://james.apache.org/`, 2018. [Online, accessed 2018-07-01].

[Apa18c] Apache Software Foundation. Cassandra Query Language (CQL) v3.3.1. `https://cassandra.apache.org/doc/old/CQL-2.2.html`, 2018. [Online, accessed 2018-07-01].

[App15] App Annie. App Annie Index™: Market Q2 2015, 2015.

[App17a] AppBrain. Ad networks – Android library statistics. `https://www.appbrain.com/stats/libraries/ad`, 2017. [Online, accessed 2017-02-15].

[App17b] AppBrain. Android analytics libraries. `https://www.appbrain.com/stats/libraries/tag/analytics/android-analytics-libraries`, 2017. [Online, accessed 2017-02-15].

[App17c] AppBrain. Android crash reporting libraries. `https://www.appbrain.com/stats/libraries/tag/crash-reporting/android-crash-reporting-libraries`, 2017. [Online, accessed 2017-02-15].

[App17d] AppBrain. Social SDKs – Android library statistics. `https://www.appbrain.com/stats/libraries/social`, 2017. [Online, accessed 2017-02-15].

[App17e] AppBrain. Video ads. `https://www.appbrain.com/stats/libraries/tag/video-ads/video-ads`, 2017. [Online, accessed 2017-02-15].

[App18a] Apple Home. `https://www.apple.com/ios/home/`, 2018. [Online, accessed 2018-07-01].

[App18b] AppScale. https://www.appscale.com, 2018. [Online, accessed 2018-07-01].

[ARF+14] Steven Arzt, Siegfried Rasthofer, Christian Fritz, Eric Bodden, Alexandre Bartel, Jacques Klein, Yves Le Traon, Damien Octeau, and Patrick McDaniel. FlowDroid: Precise Context, Flow, Field, Object-sensitive and Lifecycle-aware Taint Analysis for Android Apps. In *Proceedings of the 35th ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI)*, pages 259–269. ACM, 2014.

[ARLT17] Iman Azimi, Amir M. Rahmani, Pasi Liljeberg, and Hannu Tenhunen. Internet of things for remote elderly monitoring: a study from user-centered perspective. *Journal of Ambient Intelligence and Humanized Computing*, 8(2):273–289, 2017.

[ASS+12] Mustafa Y. Arslan, Indrajeet Singh, Shailendra Singh, Harsha V. Madhyastha, Karthikeyan Sundaresan, and Srikanth V. Krishnamurthy. Computing While Charging: Building a Distributed Computing Infrastructure Using Smartphones. In *Proceedings of the 8th International Conference on Emerging Networking Experiments and Technologies (CoNEXT)*, pages 193–204. ACM, 2012.

[AWS17] Amazon Web Services (AWS). Amazon Web Services General Reference Version 1.0, 2017.

[AWS18a] Amazon Web Services (AWS). Amazon EC2 Instance Types. https://aws.amazon.com/ec2/instance-types/, 2018. [Online, accessed 2018-07-01].

[AWS18b] Amazon Web Services (AWS). Amazon EC2 Pricing. https://aws.amazon.com/ec2/pricing/on-demand/, 2018. [Online, accessed 2018-06-22].

[AWS18c] Amazon Web Services (AWS). AWS GovCloud (US). https://aws.amazon.com/de/govcloud-us/, 2018. [Online, accessed 2018-07-01].

[Bar08] Jeff Barr. Animoto – Scaling Through Viral Growth. https://aws.amazon.com/de/blogs/aws/animoto-scali/, 2008. [Online, accessed 2018-07-01].

[Bar15] Elaine Barker. Recommendation for Key Management – Part 1: General (Revision 4). NIST Special Publication 800-57, National Institute of Standards and Technology, 2015.

[BBB+13] Bart Braem, Chris Blondia, Christoph Barz, Henning Rogge, Felix Freitag, Leandro Navarro, Joseph Bonicioli, Stavros Papathanasiou, Pau Escrich, Roger Baig Viñas, Aaron L. Kaplan, Axel Neumann, Ivan Vilata i Balaguer, Blaine Tatum, and Malcolm Matson. A Case for Research with and on Community Networks. *ACM SIGCOMM Computer Communication Review*, 43(3):68–73, 2013.

[BBGR03] Eberhard Becker, Willms Buhse, Dirk Günnewig, and Niels Rump, editors. *Digital Rights Management: Technological, Economic, and Legal and Political Aspects*. Springer, 2003.

[BDPP16] Alessio Botta, Walter de Donato, Valerio Persico, and Antonio Pescapé. Integration of Cloud computing and Internet of Things: A survey. *Future Generation Computer Systems*, 56:684–700, 2016.

[BE13] Elie Bursztein and Vijay Eranti. Internet-wide efforts to fight email phishing are working. `https://security.googleblog.com/2013/12/internet-wide-efforts-to-fight-email.html`, 2013. [Online, accessed 2018-07-01].

[Beh11] Akhil Behl. Emerging Security Challenges in Cloud Computing: An insight to cloud security challenges and their mitigation. In *Proceedings of the 2011 World Congress on Information and Communication Technologies (WICT)*, pages 217–222. IEEE, 2011.

[BEP+14] Jean Bacon, David Eyers, Thomas F. J.-M. Pasquier, Jatinder Singh, Ioannis Papagiannis, and Peter Pietzuch. Information Flow Control for Secure Cloud Computing. *IEEE Transactions on Network and Service Management*, 11(1):76–89, 2014.

[Ber09] Daniel J. Bernstein. Cryptography in NaCl. Technical report, University of Illinois at Chicago, 2009.

[Ber14] Sebastian Bereda. Flexible Configuration and Service Abstraction for Encrypted Sensor Data in the Cloud. Bachelor's thesis, RWTH Aachen University, March 2014.

[BFN16] Roger Baig, Felix Freitag, and Leandro Navarro. Fostering Collaborative Edge Service Provision in Community Clouds with Docker. In *Proceedings of the 2016 IEEE International Conference on Computer and Information Technology (CIT)*, pages 560–567. IEEE, 2016.

[BFN18] Roger Baig, Felix Freitag, and Leandro Navarro. Cloudy in guifi.net: Establishing and sustaining a community cloud as open commons. *Future Generation Computer Systems*, 87:868–887, 2018.

[BGL+17] Sean Brooks, Michael Garcia, Naomi Lefkovitz, Suzanne Lightman, and Ellen Nadeau. An Introduction to Privacy Engineering and Risk Management in Federal Systems. NIST Internal Report 8062, National Institute of Standards and Technology, 2017.

[BGR+15] Walid Benghabrit, Hervé Grall, Jean-Claude Royer, Mohamed Sellami, Monir Azraoui, Kaoutar Elkhiyaoui, Melek Önen, Anderson Santana De Oliveira, and Karin Bernsmed. From Regulatory Obligations to Enforceable Accountability Policies in the Cloud. In *International Conference on Cloud Computing and Services Science (CLOSER)*, pages 134–150. Springer, 2015.

[BH13] Carsten Bormann and Paul E. Hoffman. Concise Binary Object Representation (CBOR). Request for Comments 7049, Internet Engineering Task Force, 2013.

[BHJ+14] Ravi Bhoraskar, Seungyeop Han, Jinseong Jeon, Tanzirul Azim, Shuo Chen, Jaeyeon Jung, Suman Nath, Rui Wang, and David Wetherall. Brahmastra: Driving Apps to Test the Security of Third-Party Components. In *Proceedings of the 23rd USENIX Security Symposium*, pages 1021–1036. USENIX, 2014.

[BJD16] Joseph Bugeja, Andreas Jacobsson, and Paul Davidsson. On Privacy and Security Challenges in Smart Connected Homes. In *Proceedings of the 2016 European Intelligence and Security Informatics Conference (EISIC)*, pages 172–175. IEEE, 2016.

[BKDG13] Stephane Betgé-Brezetz, Guy-Bertrand Kamga, Marie-Pascale Dupont, and Aoues Guesmi. End-to-End Privacy Policy Enforcement in Cloud Infrastructure. In *Proceedings of the 2013 IEEE 2nd International Conference on Cloud Networking (CloudNet)*, pages 25–32. IEEE, 2013.

[BKTM11] David Bermbach, Markus Klems, Stefan Tai, and Michael Menzel. MetaStorage: A Federated Cloud Storage System to Manage Consistency-Latency Tradeoffs. In *Proceedings of the 2011 IEEE International Conference on Cloud Computing (CLOUD)*, pages 452–459. IEEE, 2011.

[BL07] Kari Barlow and Jenny Lane. Like Technology from an Advanced Alien Culture: Google Apps for Education at ASU. In *Proceedings of the 35th Annual ACM SIGUCCS Fall Conference*, pages 8–10. ACM, 2007.

[BLS+09] David Bernstein, Erik Ludvigson, Krishna Sankar, Steve Diamond, and Monique Morrow. Blueprint for the Intercloud – Protocols and Formats for Cloud Computing Interoperability. In *Proceedings of the Fourth International Conference on Internet and Web Applications and Services (ICIW)*, pages 328–336. IEEE, 2009.

[BMB10] Moritz Y. Becker, Alexander Malkis, and Laurent Bussard. A Practical Generic Privacy Language. In *Proceedings of the 6th International Conference on Information Systems Security (ICISS)*, pages 125–139. Springer, 2010.

[BMM+12] Ignacio N. Bermudez, Marco Mellia, Maurizio M. Munafò, Ram Keralapura, and Antonio Nucci. DNS to the Rescue: Discerning Content and Services in a Tangled Web. In *Proceedings of the 2012 Internet Measurement Conference (IMC)*, pages 413–426. ACM, 2012.

[BMT12] Ozalp Babaoglu, Moreno Marzolla, and Michele Tamburini. Design and Implementation of a P2P Cloud System. In *Proceedings of the*

27th Annual ACM Symposium on Applied Computing (SAC), pages 412–417. ACM, 2012.

[BNP10] Laurent Bussard, Gregory Neven, and Franz-Stefan Preiss. Downstream Usage Control. In *Proceedings of the 2010 IEEE International Symposium on Policies for Distributed Systems and Networks (POLICY)*, pages 22–29. IEEE, 2010.

[BNSS11] Sven Bugiel, Stefan Nürnberger, Ahmad-Reza Sadeghi, and Thomas Schneider. Twin Clouds: Secure Cloud Computing with Low Latency. In *Proceedings of the 12th IFIP TC 6/TC 11 International Conference on Communications and Multimedia Security (CMS)*, pages 32–44. Springer, 2011.

[BOT13] Joshua W. S. Brown, Olga Ohrimenko, and Roberto Tamassia. Haze: Privacy-preserving Real-time Traffic Statistics. In *Proceedings of the 21st ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems (SIGSPATIAL)*, pages 540–543. ACM, 2013.

[Box18] Boxcryptor – Encryption software to secure cloud files. `https://www.boxcryptor.com/`, 2018. [Online, accessed 2018-07-01].

[Boy17] Andrew Boyd. Could your Fitbit data be used to deny you health insurance? `http://theconversation.com/could-your-fitbit-data-be-used-to-deny-you-health-insurance-72565`, 2017. [Online, accessed 2018-07-01].

[BPW13] Theodore Book, Adam Pridgen, and Dan S. Wallach. Longitudinal Analysis of Android Ad Library Permissions. arXiv preprint arXiv:1303.0857 [cs.CR], 2013.

[BRAY17] Justin Brookman, Phoebe Rouge, Aaron Alva, and Christina Yeung. Cross-Device Tracking: Measurement and Disclosures. *Proceedings on Privacy Enhancing Technologies (PoPETS)*, 2017(2):133–148, 2017.

[BRC10] Rajkumar Buyya, Rajiv Ranjan, and Rodrigo N. Calheiros. InterCloud: Utility-Oriented Federation of Cloud Computing Environments for Scaling of Application Services. In *Proceedings of the 10th International Conference on Algorithms and Architectures for Parallel Processing (ICA3PP)*, pages 13–31. Springer, 2010.

[BRM16] July Katherine Díaz Barriga, Christian David Gómez Romero, and José Ignacio Rodríguez Molano. Proposal of a standard architecture of IoT for Smart Cities. In *Proceedings of the 5th International Workshop on Learning Technology for Education in Cloud (LTEC)*, pages 77–89. Springer, 2016.

[BSPW17] Anne Bowser, Katie Shilton, Jennifer Preece, and Elizabeth Warrick. Accounting for Privacy in Citizen Science: Ethical Research in a Context of Openness. In *Proceedings of the 2017 ACM Conference on Computer Supported Cooperative Work and Social Computing (CSCW)*, pages 2124–2136. ACM, 2017.

[BTMM13] Ignacio Bermudez, Stefano Traverso, Marco Mellia, and Maurizio Munafò. Exploring the Cloud from Passive Measurements: the Amazon AWS Case. In *Proceedings of the 2013 IEEE Conference on Computer Communications (INFOCOM)*, pages 230–234. IEEE, 2013.

[Bug18] Bug Labs, Inc. dweet.io – Share your thing like it ain't no thang. `https://dweet.io/`, 2018. [Online, accessed 2018-07-01].

[BW07] Dan Boneh and Brent Waters. Conjunctive, Subset, and Range Queries on Encrypted Data. In *Proceedings of the 4th Theory of Cryptography Conference (TCC)*, pages 535–554. Springer, 2007.

[BWG+16] Stefan Brenner, Colin Wulf, David Goltzsche, Nico Weichbrodt, Matthias Lorenz, Christof Fetzer, Peter Pietzuch, and Rüdiger Kapitza. SecureKeeper: Confidential ZooKeeper using Intel SGX. In *Proceedings of the 17th International Middleware Conference (Middleware)*, pages 14:1–14:13. ACM, 2016.

[BWHT12] Payam Barnaghi, Wei Wang, Cory Henson, and Kerry Taylor. Semantics for the Internet of Things: Early Progress and Back to the Future. *International Journal on Semantic Web and Information Systems (IJSWIS)*, 8(1):1–21, 2012.

[BYV+09] Rajkumar Buyya, Chee Shin Yeo, Srikumar Venugopal, James Broberg, and Ivona Brandic. Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility. *Future Generation Computer Systems*, 25(6):599–616, 2009.

[CAF13] Ruichuan Chen, Istemi Ekin Akkus, and Paul Francis. SplitX: High-performance Private Analytics. In *Proceedings of the ACM SIGCOMM 2013 Conference*, pages 315–326. ACM, 2013.

[Can17] Canasys. Cloud infrastructure market up 49%, intensifying global data center competition. Press release 2017/1630, 2017.

[Cat11] Rick Cattell. Scalable SQL and NoSQL Data Stores. *ACM SIGMOD Record*, 39(4):12–27, 2011.

[Cav08] Ann Cavoukian. Privacy in the clouds. *Identity in the Information Society*, 1(1):89–108, 2008.

[Cav11] Ann Cavoukian. Privacy by design – the 7 foundational principles. Information and Privacy Commissioner of Ontario, 2011.

[CBKA09] Justin Cappos, Ivan Beschastnikh, Arvind Krishnamurthy, and Tom Anderson. Seattle: A Platform for Educational Cloud Computing. In *Proceedings of the 40th ACM Technical Symposium on Computer Science Education (SIGCSE)*, pages 111–115. ACM, 2009.

[CCH+15] Amir Chaudhry, Jon Crowcroft, Heidi Howard, Anil Madhavapeddy, Richard Mortier, Hamed Haddadi, and Derek McAuley. Personal Data: Thinking Inside the Box. In *Proceedings of The Fifth Decennial Aarhus Conference on Critical Alternatives (AA)*, pages 29–32. Aarhus University Press, 2015.

[CCM10] Irving M. Copi, Carl Cohen, and Kenneth McMahon. *Introduction to Logic*. Pearson, 14th edition, 2010.

[CD16] Konstantinos Christidis and Michael Devetsikiotis. Blockchains and Smart Contracts for the Internet of Things. *IEEE Access*, 4:2292–2303, 2016.

[CDE+13] James C. Corbett, Jeffrey Dean, Michael Epstein, Andrew Fikes, Christopher Frost, J. J. Furman, Sanjay Ghemawat, Andrey Gubarev, Christopher Heiser, Peter Hochschild, Wilson Hsieh, Sebastian Kanthak, Eugene Kogan, Hongyi Li, Alexander Lloyd, Sergey Melnik, David Mwaura, David Nagle, Sean Quinlan, Rajesh Rao, Lindsay Rolig, Yasushi Saito, Michal Szymaniak, Christopher Taylor, Ruth Wang, and Dale Woodford. Spanner: Google's Globally-distributed Database. *ACM Transactions on Computer Systems (TOCS)*, 31(3):8:1–8:22, 2013.

[CDG+13] Ronan-Alexandra Cherrueau, Rémi Douence, Hervé Grall, Jean-Claude Royer, Mohamed Sellami, Mario Südholt, Monir Azraoui, Kaoutar Elhhiyaoui, Refik Molva, Melek Önen, Alexandr Garaga, Anderson Santa Oliveira, Jakub Sendor, and Karin Bernsmed. Policy Representation Framework. Technical report, A4Cloud Consortium, 2013.

[CEL+14] Hu Chun, Yousef Elmehdwi, Feng Li, Prabir Bhattacharya, and Wei Jiang. Outsourceable Two-Party Privacy-Preserving Biometric Authentication. In *Proceedings of the 9th ACM Symposium on Information, Computer and Communications Security (ASIACCS)*, pages 401–412. ACM, 2014.

[CGJ+09] Richard Chow, Philippe Golle, Markus Jakobsson, Elaine Shi, Jessica Staddon, Ryusuke Masuoka, and Jesus Molina. Controlling Data in the Cloud: Outsourcing Computation Without Outsourcing Control. In *Proceedings of the 2009 ACM Workshop on Cloud Computing Security (CCSW)*, pages 85–90. ACM, 2009.

[CHC+14] Simon Caton, Christian Haas, Kyle Chard, Kris Bubendorfer, and Omer F. Rana. A Social Compute Cloud: Allocating and Sharing

Infrastructure Resources via Social Networks. *IEEE Transactions on Services Computing*, 7(3):359–372, 2014.

[CHHD12] Daniele Catteddu, Giles Hogben, Thomas Haeberlen, and Lionel Dupré. Cloud Computing – Benefits, Risks and Recommendations for Information Security, Rev. B. White Paper, European Network and Information Security Agency, 2012.

[CHK11] Dave Crocker, Tony Hansen, and Murray S. Kucherawy. DomainKeys Identified Mail (DKIM) Signatures. Request for Comments 6376, Internet Engineering Task Force, 2011.

[Cho10] Yung Chou. Cloud Computing Primer for IT Pros. `https://blogs.technet.microsoft.com/yungchou/2010/11/15/cloud-computing-primer-for-it-pros/`, 2010. [Online, accessed 2018-07-01].

[Cis16] Cisco Systems, Inc. SenderBase. `http://www.senderbase.org/`, 2016. [Online, accessed 2016-11-16].

[CL11] Chih-Chung Chang and Chih-Jen Lin. LIBSVM: A Library for Support Vector Machines. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 2(3):27:1–27:27, 2011.

[CL13a] Leucio Antonio Cutillo and Antonio Lioy. Privacy-by-Design Cloud Computing Through Decentralization and Real Life Trust. In *Proceedings of the 2013 IEEE Thirteenth International Conference on Peer-to-Peer Computing (P2P)*, pages 1–2. IEEE, 2013.

[CL13b] Leucio Antonio Cutillo and Antonio Lioy. Towards Privacy-by-Design Peer-to-Peer Cloud Computing. In *Proceedings of the 10th International Conference on Trust, Privacy, and Security in Digital Business (TrustBus)*, pages 85–96. Springer, 2013.

[Cla97] Roger Clarke. Introduction to Dataveillance and Information Privacy, and Definitions of Terms, 1997.

[Clo15] Cloud Industry Forum. UK Cloud adoption snapshot & trends for 2016 – The business case for Cloud. White Paper, 2015.

[Clo16] CloudEmailSecurity.org. Cloud Email Security Comparison. `http://cloudemailsecurity.org/`, 2016. [Online, accessed 2016-11-16].

[CLSX12] T.-H. Hubert Chan, Mingfei Li, Elaine Shi, and Wenchang Xu. Differentially Private Continual Monitoring of Heavy Hitters from Distributed Streams. In *Proceedings of the 12th Privacy Enhancing Technologies Symposium (PETS)*, pages 140–159. Springer, 2012.

[Clu18] Clustrix, Inc. Clustrix – Scale-out RDBMS. `http://www.clustrix.com/`, 2018. [Online, accessed 2018-07-01].

[CLZ99] Antonio Corradi, Letizia Leonardi, and Franco Zambonelli. Diffusive Load-Balancing Policies for Dynamic Applications. *IEEE Concurrency*, 7(1):22–31, 1999.

[CMT12] Dawn Cappelli, Andrew Moore, and Randall Trzeciak. *The CERT Guide to Insider Threats: How to Prevent, Detect, and Respond to Information Technology Crimes (Theft, Sabotage, Fraud)*. Addison-Wesley, 2012.

[CN12] William R. Claycomb and Alex Nicoll. Insider Threats to Cloud Computing: Directions for New Research Challenges. In *Proceedings of the 2012 IEEE 36th Annual Computer Software and Applications Conference (COMPSAC)*, pages 387–394. IEEE, 2012.

[Coo18] Rob Coombs. Arm launches first set of Threat Models for PSA: IoT Security should start with analysis. `https://community.arm.com/iot/b/blog/posts/arm-launches-first-set-of-threat-models-for-psa`, 2018. [Online, accessed 2018-07-01].

[Cor17] Nigel Cory. Cross-Border Data Flows: Where Are the Barriers, and What Do They Cost? Technical report, Information Technology & Innovation Foundation, 2017.

[COTC17] Rasel Chowdhury, Hakima Ould-Slimane, Chamseddine Talhi, and Mohamed Cheriet. Attribute-Based Encryption for Preserving Smart Home Data Privacy. In *Proceedings of the 15th International Conference on Smart Homes and Health Telematics (ICOST)*, pages 185–197. Springer, 2017.

[Cou13] Martin Courtney. Premium binds. *Engineering & Technology*, 8(6):68–73, 2013.

[CPH03] Cheun Ngen Chong, Zhonghong Peng, and Pieter H. Hartel. Secure Audit Logging with Tamper-Resistant Hardware. In *Proceedings of the IFIP TC11 18th International Conference on Information Security (SEC)*, pages 73–84. Springer, 2003.

[CRFG12] Ruichuan Chen, Alexey Reznichenko, Paul Francis, and Johannes Gehrke. Towards Statistical Queries over Distributed Private User Data. In *Proceedings of the 9th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, pages 169–182. USENIX, 2012.

[CRKH11] Delphine Christin, Andreas Reinhardt, Salil S. Kanhere, and Matthias Hollick. A survey on privacy in mobile participatory sensing applications. *Journal of Systems and Software*, 84(11):1928–1946, 2011.

[CSA96] Canadian Standards Association. Model Code for the Protection of Personal Information. National Standard of Canada CAN/CSA-Q830-96, 1996.

[CSA10] Cloud Security Alliance. Top Threats to Cloud Computing V1.0, 2010.

[CUKB14] Terence Chen, Imdad Ullah, Mohamed Ali Kaafar, and Roksana Boreli. Information Leakage Through Mobile Analytics Services. In *Proceedings of the 15th Workshop on Mobile Computing Systems and Applications (HotMobile)*, pages 15:1–15:6. ACM, 2014.

[Cun16] Clark D. Cunningham. Feds: We can read all your email, and you'll never know. `http://theconversation.com/feds-we-can-read-all-your-email-and-youll-never-know-65620`, 2016. [Online, accessed 2018-07-01].

[CZFK12] Dunren Che, Mengxia Zhu, Jason Fairfield, and Mustafa Khaleel. CuteCloud: Putting "Credit Union" Cloud Computing into Practice. In *Proceedings of the 2012 ACM Research in Applied Computation Symposium (RACS)*, pages 80–85. ACM, 2012.

[DAM+15] Zakir Durumeric, David Adrian, Ariana Mirian, Michael Bailey, and J. Alex Halderman. A Search Engine Backed by Internet-Wide Scanning. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security (CCS)*, pages 542–553. ACM, 2015.

[Dat17a] Datanyze. CDN market share in the Alexa top 1M. `https://www.datanyze.com/market-share/cdn/Alexa%20top%201M`, 2017. [Online, accessed 2017-02-17].

[Dat17b] DataStax, Inc. Apache Cassandra™ 2.0 Documentation. `http://docs.datastax.com/en/archived/cassandra/2.0/`, 2017. [Online, accessed 2018-07-01].

[DEG+15] Chris Dibben, Mark Elliot, Heather Gowans, Darren Lightfoot, and Data Linkage Centres. The data linkage environment. In Katie Harron, Harvey Goldstein, and Chris Dibben, editors, *Methodological Developments in Data Linkage*, chapter 3, pages 36–62. John Wiley & Sons, 2015.

[DG04] Jeffrey Dean and Sanjay Ghemawat. MapReduce: Simplified Data Processing on Large Clusters. In *Proceedings of the 6th Symposium on Operating System Design and Implementation (OSDI)*, pages 137–150. USENIX, 2004.

[DHJ+07] Giuseppe DeCandia, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati, Avinash Lakshman, Alex Pilchin, Swaminathan Sivasubramanian, Peter Vosshall, and Werner Vogels. Dynamo: Amazon's Highly Available Key-value Store. In *Proceedings of Twenty-first ACM SIGOPS Symposium on Operating Systems Principles (SOSP)*, pages 205–220. ACM, 2007.

[DJ10] Marten van Dijk and Ari Juels. On the Impossibility of Cryptography Alone for Privacy-Preserving Cloud Computing. In *Proceedings of the 5th USENIX Workshop on Hot Topics in Security (HotSec)*, pages 1–8. USENIX, 2010.

[DK12] David Dittrich and Erin Kenneally. The Menlo Report: Ethical Principles Guiding Information and Communication Technology Research. Technical report, U.S. Department of Homeland Security, 2012.

[DKG+10] Adam Dou, Vana Kalogeraki, Dimitrios Gunopulos, Taneli Mielikainen, and Ville H. Tuulos. Misco: A MapReduce Framework for Mobile Systems. In *Proceedings of the 3rd International Conference on PErvasive Technologies Related to Assistive Environments (PETRA)*, pages 32:1–32:8. ACM, 2010.

[DM16] Nikos Drakos and Jeffrey Mann. Survey Analysis: Microsoft Dominates Cloud Email in Large Public Companies but Shares the Rest With Google. Gartner Report G00292300, 2016.

[DMM+12] Idilio Drago, Marco Mellia, Maurizio M. Munafò, Anna Sperotto, Ramin Sadre, and Aiko Pras. Inside Dropbox: Understanding Personal Cloud Storage Services. In *Proceedings of the 2012 Internet Measurement Conference (IMC)*, pages 481–494. ACM, 2012.

[Dom16] DomainTools. Statistics About Mail Servers. `http://research.domaintools.com/statistics/mailservers/`, 2016. [Online, accessed 2016-11-16].

[DR08] Tim Dierks and Eric Rescorla. The Transport Layer Security (TLS) Protocol Version 1.2. Request for Comments 5246, Internet Engineering Task Force, 2008.

[Dri15] Doug Drinkwater. Hackers route via Tor for stealthy 'slow-death' DoS attacks. `https://www.scmagazineuk.com/hackers-route-via-tor-for-stealthy-slow-death-dos-attacks/article/537484/`, 2015. [Online, accessed 2018-07-01].

[Dri16] Arthur Drichel. Large Scale Analysis of the Cloud Usage of Smartphone Applications. Bachelor's thesis, RWTH Aachen University, September 2016.

[Dro15] Dropbox Inc. 400 million strong. `https://blogs.dropbox.com/dropbox/2015/06/400-million-users/`, 2015. [Online, accessed 2018-07-01].

[DUM10] Ali Dehghantanha, Nur Izura Udzir, and Ramlan Mahmod. Towards a Pervasive Formal Privacy Language. In *Proceedings of the 2010 IEEE 24th International Conference on Advanced Information Networking and Applications Workshops (WAINA)*, pages 1085–1091. IEEE, 2010.

[Dwo06] Cynthia Dwork. Differential Privacy. In *Proceedings of the 33rd International Colloquium on Automata, Languages and Programming (ICALP)*, volume II, pages 1–12. Springer, 2006.

[Eas11] Donald Eastlake. Transport Layer Security (TLS) Extensions: Extension Definitions. Request for Comments 6066, Internet Engineering Task Force, 2011.

[EGH+14] William Enck, Peter Gilbert, Seungyeop Han, Vasant Tendulkar, Byung-Gon Chun, Landon P. Cox, Jaeyeon Jung, Patrick McDaniel, and Anmol N. Sheth. TaintDroid: An Information-Flow Tracking System for Realtime Privacy Monitoring on Smartphones. *ACM Transactions on Computer Systems*, 32(2):5:1–5:29, 2014.

[EGSR16] Ittay Eyal, Adem Efe Gencer, Emin Gün Sirer, and Robbert van Renesse. Bitcoin-NG: A Scalable Blockchain Protocol. In *Proceedings of the 13th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, pages 45–59. USENIX, 2016.

[EHH+14] Michael Eggert, Roger Häußling, Martin Henze, Lars Hermerschmidt, René Hummen, Daniel Kerpen, Antonio Navarro Pérez, Bernhard Rumpe, Dirk Thißen, and Klaus Wehrle. SensorCloud: Towards the Interdisciplinary Development of a Trustworthy Platform for Globally Interconnected Sensors and Actuators. In Helmut Krcmar, Ralf Reussner, and Bernhard Rumpe, editors, *Trusted Cloud Computing*, pages 203–218. Springer, 2014.

[EHKR14] Michael Eggert, Roger Häußling, Daniel Kerpen, and Kirsten Rüssmann. SensorCloud: Sociological Contextualization of an Innovative Cloud Platform. In Helmut Krcmar, Ralf Reussner, and Bernhard Rumpe, editors, *Trusted Cloud Computing*, pages 295–313. Springer, 2014.

[Ela13] ElasticInbox – Scalable Email Store for the Cloud. `http://www.elasticinbox.com/`, 2013. [Online, accessed 2018-07-01].

[Ele14] Nikolay Elenkov. *Android Security Internals: An In-depth Guide to Android's Security Architecture*. No Starch Press, 1st edition, 2014.

[ELL+14] Daniel Espling, Lars Larsson, Wubin Li, Johan Tordsson, and Erik Elmroth. Modeling and Placement of Cloud Services with Internal Structure. *IEEE Transactions on Cloud Computing*, 4(4):429–439, 2014.

[EMM06] Mohamed Eltoweissy, Mohammed Moharrum, and Ravi Mukkamala. Dynamic Key Management in Sensor Networks. *IEEE Communications Magazine*, 44(4):122–130, 2006.

[EMP13] Thomas Erl, Zaigham Mahmood, and Ricardo Puttini. *Cloud Computing: Concepts, Technology & Architecture*. Pearson Education, 2013.

[EPK14] Úlfar Erlingsson, Vasyl Pihur, and Aleksandra Korolova. RAPPOR: Randomized Aggregatable Privacy-Preserving Ordinal Response. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security (CCS)*, pages 1054–1067. ACM, 2014.

[ESM09] Peter R. Elespuru, Sagun Shakya, and Shivakant Mishra. MapReduce System over Heterogeneous Mobile Devices. In *Proceedings of the 7th IFIP WG 10.2 International Workshop on Software Technologies for Embedded and Ubiquitous Systems (SEUS)*, pages 168–179. Springer, 2009.

[EU95] Directive 95/46/EC of the European Parliament and of the Council of 24 October 1995 on the protection of individuals with regard to the processing of personal data and on the free movement of such data. Official Journal of the European Union, L281, 23/11/1995, pages 31–50, 1995.

[FBL15] Benjamin Fabian, Annika Baumann, and Jessika Lackner. Topological analysis of cloud service connectivity. *Computers & Industrial Engineering*, 88:151–165, 2015.

[FDW+15] Sebastian Funke, Jörg Daubert, Alexander Wiesmaier, Panayotis Kikiras, and Max Muehlhaeuser. End-2-End Privacy Architecture for IoT. In *Proceedings of the 2015 IEEE Conference on Communications and Network Security (CNS)*, pages 705–706. IEEE, 2015.

[Fed14] Federal Office for Information Security (BSI). Protection Profile for the Gateway of a Smart Metering System (Smart Meter Gateway PP). Version 1.3 (Final Release), Certification-ID: BSI-CC-PP-0073, 2014.

[FG06] Carlos Flavián and Miguel Guinalíu. Consumer trust, perceived security and privacy policy: Three basic elements of loyalty to a web site. *Industrial Management & Data Systems*, 106(5):601–620, 2006.

[FKB+15] Denzil Ferreira, Vassilis Kostakos, Alastair R. Beresford, Janne Lindqvist, and Anind K. Dey. Securacy: An Empirical Investigation of Android Applications' Network Usage, Privacy and Security. In *Proceedings of the 8th ACM Conference on Security & Privacy in Wireless and Mobile Networks (WiSec)*, pages 11:1–11:11. ACM, 2015.

[FKH15] Xun Fan, Ethan Katz-Bassett, and John Heidemann. Assessing Affinity Between Users and CDN Sites. In *Proceedings of the 7th International Workshop on Traffic Monitoring and Analysis (TMA)*, pages 95–110. Springer, 2015.

[FM12] Primavera De Filippi and Smari McCarthy. Cloud Computing: Centralization and Data Sovereignty. *European Journal of Law and Technology*, 3(2), 2012.

[Fre15] Julien Freudiger. How Talkative is Your Mobile Device?: An Experimental Study of Wi-Fi Probe Requests. In *Proceedings of the 8th ACM Conference on Security & Privacy in Wireless and Mobile Networks (WiSec)*, pages 8:1–8:6. ACM, 2015.

[FSC15] Pierdomenico Fiadino, Mirko Schiavone, and Pedro Casas. Vivisecting WhatsApp in Cellular Networks: Servers, Flows, and Quality of Experience. In *Proceedings of the 7th International Workshop on Traffic Monitoring and Analysis (TMA)*, pages 49–63. Springer, 2015.

[FWF13] Rachel L. Finn, David Wright, and Michael Friedewald. Seven Types of Privacy. In Serge Gutwirth, Ronald Leenes, Paul de Hert, and Yves Poullet, editors, *European Data Protection: Coming of Age*, chapter 1, pages 3–32. Springer, 2013.

[GB14] Nikolay Grozev and Rajkumar Buyya. Inter-Cloud architectures and application brokering: taxonomy and survey. *Software: Practice and Experience*, 44(3):369–390, 2014.

[GBMP13] Jayavardhana Gubbi, Rajkumar Buyya, Slaven Marusic, and Marimuthu Palaniswami. Internet of Things (IoT): A vision, architectural elements, and future directions. *Future Generation Computer Systems*, 29(7):1645–1660, 2013.

[GCEC12] Clint Gibler, Jonathan Crussell, Jeremy Erickson, and Hao Chen. AndroidLeaks: Automatically Detecting Potential Privacy Leaks in Android Applications on a Large Scale. In *Proceedings of the 5th International Conference on Trust and Trustworthy Computing (TRUST)*, pages 291–307. Springer, 2012.

[GDPR16] Regulation (EU) 2016/679 of the European Parliament and of the Council of 27 April 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing Directive 95/46/EC (General Data Protection Regulation). Official Journal of the European Union, L119, 4/5/2016, pages 1–88, 2016.

[Gee05] David Geer. Will binary XML speed network traffic? *Computer*, 38(4):16–18, 2005.

[Gel09] Robert Gellman. Privacy in the Clouds: Risks to Privacy and Confidentiality from Cloud Computing. World Privacy Forum, 2009.

[Gel13] Barton Gellman. Edward Snowden, after months of NSA revelations, says his mission's accomplished. *The Washington Post*, December 24, 2013.

[GG11] Gerd Gigerenzer and Wolfgang Gaissmaier. Heuristic Decision Making. *Annual Review of Psychology*, 62(1):451–482, 2011.

[GGBM15] Mateusz Guzek, Alicja Gniewek, Pascal Bouvry, and Jedrzej Musial. Cloud Brokering: Current Practices and Upcoming Challenges. *IEEE Cloud Computing*, 2(2):40–47, 2015.

[GGJ17] Amal Ghorbel, Mahmoud Ghorbel, and Mohamed Jmaiel. Privacy in cloud computing environments: a survey and research challenges. *The Journal of Supercomputing*, 73(6):2763–2800, 2017.

[GHMP08] Albert Greenberg, James Hamilton, David A. Maltz, and Parveen Patel. The Cost of a Cloud: Research Problems in Data Center Networks. *ACM SIGCOMM Computer Communication Review*, 39(1), 2008.

[GHTC13] Katarina Grolinger, Wilson Higashino, Abhinav Tiwari, and Miriam Capretz. Data management in cloud environments: NoSQL and NewSQL data stores. *Journal of Cloud Computing: Advances, Systems and Applications*, 2(1), 2013.

[GHW+19] René Glebke, Martin Henze, Klaus Wehrle, Philipp Niemietz, Daniel Trauth, Patrick Mattfeld, and Thomas Bergs. A Case for Integrated Data Processing in Large-Scale Cyber-Physical Systems. In *Proceedings of the 52nd Hawaii International Conference on System Sciences (HICSS)*, 2019.

[Gie14] Johannes van der Giet. Data Annotation Handling in a Highly Scalable Distributed Database System. Master's thesis, RWTH Aachen University, October 2014.

[GLBA99] United States Congress. Gramm-Leach-Bliley Act (GLBA). Pub.L. 106-102, 113 Stat. 1338, 1999.

[Gol04] Oded Goldreich. *Foundations of Cryptography: Volume 2, Basic Applications*. Cambridge University Press, 2004.

[Goo16] Google. Top Free in Android Apps – Android Apps on Google Play. `https://play.google.com/store/apps/collection/topselling_free`, 2016. [Online, accessed 2016-08-01].

[Goo18a] Google. Google App Engine. `https://cloud.google.com/appengine/`, 2018. [Online, accessed 2018-07-01].

[Goo18b] Google. Google Apps for Government. `http://gov.googleapps.com/`, 2018. [Online, accessed 2018-07-01].

[Goo18c] Google. How Gmail ads work. `https://support.google.com/mail/answer/6603`, 2018. [Online, accessed 2018-07-01].

[Gös07] Stefan Gössner. JSONPath – XPath for JSON. `http://goessner.net/articles/JsonPath/`, 2007. [Online, accessed 2018-07-01].

[GR12] John Gantz and David Reinsel. The Digital Universe in 2020: Big Data, Bigger Digital Shadows, and Biggest Growth in the Far East. IDC iView, 2012.

[Gre17] Graham Greenleaf. Global Data Privacy Laws 2017: 120 National Data Privacy Laws, Including Indonesia and Turkey. 145 Privacy Laws & Business International Report, 10-13; UNSW Law Research Paper No. 45, 2017.

[Gro13] Marcel Großfengels. Machine-readable Data Handling Annotations for the Cloud. Bachelor's thesis, RWTH Aachen University, October 2013.

[GSMG12] Raúl Gracia-Tinedo, Marc Sánchez-Artigas, Adrián Moreno-Martínez, and Pedro García-López. FriendBox: A Hybrid F2F Personal Storage Application. In *Proceedings of the 2012 IEEE 5th International Conference on Cloud Computing (CLOUD)*, pages 131–138. IEEE, 2012.

[GW10] Oscar Garcia-Morchon and Klaus Wehrle. Modular Context-aware Access Control for Medical Sensor Networks. In *Proceedings of the 15th ACM Symposium on Access Control Models and Technologies (SACMAT)*, pages 129–138. ACM, 2010.

[GZ15] Andy Greenberg and Kim Zetter. How the Internet of Things Got Hacked. `https://www.wired.com/2015/12/2015-the-year-the-internet-of-things-got-hacked/`, 2015. [Online, accessed 2018-07-01].

[Hae10] Andreas Haeberlen. A Case for the Accountable Cloud. *ACM SIGOPS Operating Systems Review*, 44(2):52–57, 2010.

[Hal16] Vanessa Halter. Privacy as a strategic advantage for healthcare products & services. `http://www.healthtechsydney.com.au/blog/2016/03/07/privacy-as-a-strategic-advantage-for-healthcare-products-services/`, 2016. [Online, accessed 2018-07-01].

[Han00] M. David Hanson. The Client/Server Architecture. In Gilbert Held, editor, *Server Management*, chapter 1, pages 3–13. CRC Press, 2000.

[HB96] John Hawkinson and Tony Bates. Guidelines for creation, selection, and registration of an Autonomous System (AS). Request for Comments 1930, Internet Engineering Task Force, 1996.

[HBHW14] Martin Henze, Sebastian Bereda, René Hummen, and Klaus Wehrle. SCSlib: Transparently Accessing Protected Sensor Data in the Cloud. In *Proceedings of the 6th International Symposium on Applications of Ad hoc and Sensor Networks (AASNET)*, volume 37 of *Procedia Computer Science*, pages 370–375. Elsevier, 2014.

[Hea17] Olly Headey. Running a high-availability SaaS infrastructure without breaking the bank. `http://engineering.freeagent.com/2017/02/06/ha-infrastructure-without-breaking-the-bank/`, 2017. [Online, accessed 2018-07-01].

[Hel15] David Hellmanns. Making Individual Cloud Usage of Smartphone Users Transparent. Bachelor's thesis, RWTH Aachen University, May 2015.

[Hem05] Stephen Hemminger. Network Emulation with NetEm. In *linux.conf.au*, 2005.

[HFW+13] Keqiang He, Alexis Fisher, Liang Wang, Aaron Gember, Aditya Akella, and Thomas Ristenpart. Next Stop, the Cloud: Understanding Modern Web Service Deployment in EC2 and Azure. In *Proceedings of the 2013 Conference on Internet Measurement Conference (IMC)*, pages 177–190. ACM, 2013.

[HG15] Wouter Haerick and Milon Gupta. 5G and the Factories of the Future. White paper, 5G Infrastructure Public Private Partnership (5G PPP), 2015.

[HGC12] Oliver Hohlfeld, Thomas Graf, and Florin Ciucu. Longtime Behavior of Harvesting Spam Bots. In *Proceedings of the 2012 Internet Measurement Conference (IMC)*, pages 453–460. ACM, 2012.

[HGKW13] Martin Henze, Marcel Großfengels, Maik Koprowski, and Klaus Wehrle. Towards Data Handling Requirements-aware Cloud Computing. In *Proceedings of the 2013 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*, pages 266–269. IEEE, 2013.

[HHCW12] René Hummen, Martin Henze, Daniel Catrein, and Klaus Wehrle. A Cloud Design for User-controlled Storage and Processing of Sensor Data. In *Proceedings of the 2012 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*, pages 232–240. IEEE, 2012.

[HHH+17] Martin Henze, Jens Hiller, René Hummen, Roman Matzutt, Klaus Wehrle, and Jan Henrik Ziegeldorf. Network Security and Privacy for Cyber-Physical Systems. In Houbing Song, Glenn A. Fink, and Sabina Jeschke, editors, *Security and Privacy in Cyber-Physical Systems: Foundations, Principles and Applications*, chapter 2, pages 25–56. Wiley-IEEE Press, 2017.

[HHHW13] René Hummen, Jens Hiller, Martin Henze, and Klaus Wehrle. Slimfit – A HIP DEX Compression Layer for the IP-based Internet of Things. In *Proceedings of the 2013 IEEE 9th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob)*, pages 259–266. IEEE, 2013.

[HHHW16] Martin Henze, Jens Hiller, Oliver Hohlfeld, and Klaus Wehrle. Moving Privacy-Sensitive Services from Public Clouds to Decentralized Private Clouds. In *Proceedings of the 2016 IEEE International Conference on Cloud Engineering Workshops (IC2EW)*, pages 130–135. IEEE, 2016.

[HHK+14] Martin Henze, Lars Hermerschmidt, Daniel Kerpen, Roger Häußling, Bernhard Rumpe, and Klaus Wehrle. User-driven Privacy Enforcement for Cloud-based Services in the Internet of Things. In *Proceedings of the 2014 International Conference on Future Internet of Things and Cloud (FiCloud)*, pages 191–196. IEEE, 2014.

[HHK+16] Martin Henze, Lars Hermerschmidt, Daniel Kerpen, Roger Häußling, Bernhard Rumpe, and Klaus Wehrle. A Comprehensive Approach to Privacy in the Cloud-based Internet of Things. *Future Generation Computer Systems (FGCS)*, 56:701–718, 2016.

[HHM+13] Martin Henze, René Hummen, Roman Matzutt, Daniel Catrein, and Klaus Wehrle. Maintaining User Control While Storing and Processing Sensor Data in the Cloud. *International Journal of Grid and High Performance Computing*, 5(4):97–112, 2013.

[HHMW14] Martin Henze, René Hummen, Roman Matzutt, and Klaus Wehrle. A Trust Point-based Security Architecture for Sensor Data in the Cloud. In Helmut Krcmar, Ralf Reussner, and Bernhard Rumpe, editors, *Trusted Cloud Computing*, pages 77–106. Springer, 2014.

[HHMW16] Martin Henze, René Hummen, Roman Matzutt, and Klaus Wehrle. The SensorCloud Protocol: Securely Outsourcing Sensor Data to the Cloud. Technical Report AIB-2016-06, Department of Computer Science, RWTH Aachen University, 2016.

[HHS+16] Martin Henze, Jens Hiller, Sascha Schmerling, Jan Henrik Ziegeldorf, and Klaus Wehrle. CPPL: Compact Privacy Policy Language. In *Proceedings of the 15th ACM Workshop on Privacy in the Electronic Society (WPES)*, pages 99–110. ACM, 2016.

[HHS+18] Jens Hiller, Martin Henze, Martin Serror, Eric Wagner, Jan Niklas Richter, and Klaus Wehrle. Secure Low Latency Communication for Constrained Industrial IoT Scenarios. In *Proceedings of the 43rd IEEE Conference on Local Computer Networks (LCN)*. IEEE, 2018.

[HHW13a] Martin Henze, René Hummen, and Klaus Wehrle. The Cloud Needs Cross-Layer Data Handling Annotations. In *Proceedings of the 2013 IEEE Security and Privacy Workshops (SPW)*, pages 18–22. IEEE, 2013.

[HHW+13b] René Hummen, Jens Hiller, Hanno Wirtz, Martin Henze, Hossein Shafagh, and Klaus Wehrle. 6LoWPAN Fragmentation Attacks and Mitigation Mechanisms. In *Proceedings of the Sixth ACM Conference on Security and Privacy in Wireless and Mobile Networks (WiSec)*, pages 55–66. ACM, 2013.

[HIFZ17] Martin Henze, Ritsuma Inaba, Ina Berenice Fink, and Jan Henrik Ziegeldorf. Privacy-preserving Comparison of Cloud Exposure Induced

by Mobile Apps. In *Proceedings of the 14th EAI International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services (MobiQuitous)*. ACM, 2017.

[Hil14] Jens Hiller. PriverCloud - A Peer-to-Peer Cloud for Secure Service Operation. Master's thesis, RWTH Aachen University, September 2014.

[HIPA96] United States Congress. Health Insurance Portability and Accountability Act of 1996 (HIPAA). Pub.L. 104–191, 110 Stat. 1936, 1996.

[HJS+03] Nicholas J. A. Harvey, Michael B. Jones, Stefan Saroiu, Marvin Theimer, and Alec Wolman. SkipNet: A Scalable Overlay Network with Practical Locality Properties. In *Proceedings of the 4th USENIX Symposium on Internet Technologies and Systems (USITS)*. USENIX, 2003.

[HKH+16] Martin Henze, Daniel Kerpen, Jens Hiller, Michael Eggert, David Hellmanns, Erik Mühmer, Oussama Renuli, Henning Maier, Christian Stüble, Roger Häußling, and Klaus Wehrle. Towards Transparent Information on Individual Cloud Service Usage. In *Proceedings of the 2016 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*, pages 366–370. IEEE, 2016.

[HKP+18] Jens Hiller, Maël Kimmerlin, Max Plauth, Seppo Heikkila, Stefan Klauck, Ville Lindfors, Felix Eberhardt, Dariusz Bursztynowski, Jesus Llorente Santos, Oliver Hohlfeld, and Klaus Wehrle. Giving Customers Control over Their Data: Integrating a Policy Language into the Cloud. In *Proceedings of the 2018 IEEE International Conference on Cloud Engineering (IC2E)*, pages 241–249. IEEE, 2018.

[HMH+17] Martin Henze, Roman Matzutt, Jens Hiller, Erik Mühmer, Jan Henrik Ziegeldorf, Johannes van der Giet, and Klaus Wehrle. Practical Data Compliance for Cloud Storage. In *Proceedings of the 2017 IEEE International Conference on Cloud Engineering (IC2E)*, pages 252–258. IEEE, 2017.

[HMH+18] Martin Henze, Roman Matzutt, Jens Hiller, Erik Mühmer, Jan Henrik Ziegeldorf, Johannes van der Giet, and Klaus Wehrle. Complying with Data Handling Requirements in Cloud Storage Systems. arXiv preprint arXiv:1806.11448 [cs.NI], 2018.

[HMR+14] W. Kuan Hon, Christopher Millard, Chris Reed, Jatinder Singh, Ian Walden, and Jon Crowcroft. Policy, Legal and Regulatory Implications of a Europe-Only Cloud. Queen Mary School of Law Legal Studies Research Paper 191/2015, 2014.

[HNLL04] Jason I. Hong, Jennifer D. Ng, Scott Lederer, and James A. Landay. Privacy Risk Models for Designing Privacy-sensitive Ubiquitous Computing Systems. In *Proceedings of the 5th Conference on Designing*

Interactive Systems: Processes, Practices, Methods, and Techniques (DIS), pages 91–100. ACM, 2004.

[Hol07] Jan Holvast. History of privacy. In Karl de Leeuw and Jan Bergstra, editors, *The History of Information Security: A Comprehensive Handbook*, chapter 27, pages 737–769. Elsevier, 2007.

[Hor08] John B. Horrigan. Use of Cloud Computing Applications and Services. Data memo, Pew Research Center, 2008.

[Hos16] M. Shamim Hossain. Patient State Recognition System for Healthcare Using Speech and Facial Expressions. *Journal of Medical Systems*, 40(12), 2016.

[HPB+07] Manuel Hilty, Alexander Pretschner, David Basin, Christian Schaefer, and Thomas Walter. A Policy Language for Distributed Usage Control. In *Proceedings of the 12th European Symposium On Research In Computer Security (ESORICS)*, pages 531–546. Springer, 2007.

[HPH+17] Martin Henze, Jan Pennekamp, David Hellmanns, Erik Mühmer, Jan Henrik Ziegeldorf, Arthur Drichel, and Klaus Wehrle. CloudAnalyzer: Uncovering the Cloud Usage of Mobile Apps. In *Proceedings of the 14th EAI International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services (MobiQuitous)*. ACM, 2017.

[HRGD08] Andreas Haeberlen, Rodrigo Rodrigues, Krishna Gummadi, and Peter Druschel. Pretty Good Packet Authentication. In *Proceedings of the Fourth Conference on Hot Topics in System Dependability (HotDep)*. USENIX, 2008.

[HRL14] Pei-Fang Hsu, Soumya Ray, and Yu-Yu Li-Hsieh. Examining cloud computing adoption intention, pricing mechanism, and deployment model. *International Journal of Information Management*, 34(4):474–488, 2014.

[HS09] Gerrit Hornung and Christoph Schnabel. Data protection in Germany I: The population census decision and the right to informational self-determination. *Computer Law & Security Review*, 25(1):84–88, 2009.

[HSF+09] Shuang Hao, Nadeem Ahmed Syed, Nick Feamster, Alexander G Gray, and Sven Krasser. Detecting Spammers with SNARE: Spatio-temporal Network-level Automatic Reputation Engine. In *Proceedings of the 18th USENIX Security Symposium*, pages 101–118. USENIX, 2009.

[HSH17] Martin Henze, Mary Peyton Sanford, and Oliver Hohlfeld. Veiled in Clouds? Assessing the Prevalence of Cloud Computing in the Email Landscape. In *Proceedings of the 2017 Network Traffic Measurement and Analysis Conference (TMA)*, pages 1–9. IEEE, 2017.

[HWM+17] Martin Henze, Benedikt Wolters, Roman Matzutt, Torsten Zimmermann, and Klaus Wehrle. Distributed Configuration, Authorization and Management in the Cloud-based Internet of Things. In *Proceedings of the 2017 IEEE International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*, pages 185–192. IEEE, 2017.

[HWZ04] Minhuan Huang, Chunlei Wang, and Lufeng Zhang. Toward a Reusable and Generic Security Aspect Library. In *Proceedings of the AOSD 2004 Workshop on AOSD Technology for Application-level Security (AOSD-SEC)*, 2004.

[IBM14] IBM. The Hartford Signs Agreement With IBM To Move IT To The Cloud. `http://www-03.ibm.com/press/us/en/pressrelease/43695.wss`, 2014. [Online, accessed 2018-07-01].

[IBM17] IBM. IBM ILOG CPLEX Optimization Studio, 2017.

[IDC17] International Data Corporation (IDC). Smartphone OS Market Share, 2017 Q1. `https://www.idc.com/promo/smartphone-market-share/os`, 2017. [Online, accessed 2018-07-01].

[IK04] Wassim Itani and Ayman Kayssi. SPECSA: a scalable, policy-driven, extensible, and customizable security architecture for wireless enterprise applications. *Computer Communications*, 27(18):1825–1839, 2004.

[IKC09] Wassim Itani, Ayman Kayssi, and Ali Chehab. Privacy as a Service: Privacy-Aware Data Storage and Processing in Cloud Computing Architectures. In *Proceedings of the Eighth IEEE International Conference on Dependable, Autonomic and Secure Computing (DASC)*, pages 711–716. IEEE, 2009.

[Ina17] Ritsuma Inaba. Incorporation of Security Features in Cloud Usage Analysis Tool. Internship report (Undergraduate Research Opportunities Program), RWTH Aachen University and University of Michigan, July 2017.

[Int12] Intel IT Center. Peer Research: What's Holding Back the Cloud? White Paper, 2012.

[ISKČ11] Iulia Ion, Niharika Sachdeva, Ponnurangam Kumaraguru, and Srdjan Čapkun. Home is Safer Than the Cloud!: Privacy Concerns for Consumer Cloud Storage. In *Proceedings of the Seventh Symposium on Usable Privacy and Security (SOUPS)*, pages 13:1–13:20. ACM, 2011.

[ISO13] Information technology – Security techniques – Code of practice for information security controls, International Standards Organization/International Electrotechnical Commission Standard ISO/IEC 27002, Revision 2013, 2013.

[ISO14] Information technology – Cloud computing – Overview and vocabulary, International Standards Organization/International Electrotechnical Commission Standard ISO/IEC 17788, Revision 2014, 2014.

[JBM+17] Sabina Jeschke, Christian Brecher, Tobias Meisen, Denis Özdemir, and Tim Eschert. Industrial Internet of Things and Cyber Manufacturing Systems. In Sabina Jeschke, Christian Brecher, Houbing Song, and Danda B. Rawat, editors, *Industrial Internet of Things: Cybermanufacturing Systems*, pages 3–19. Springer, 2017.

[JBS15] Michael Jones, John Bradley, and Nat Sakimura. JSON Web Signature (JWS). Request for Comments 7515, Internet Engineering Task Force, 2015.

[JG11] Wayne Jansen and Timothy Grance. Guidelines on Security and Privacy in Public Cloud Computing. NIST Special Publication 800-144, National Institute of Standards and Technology, 2011.

[JH15] Michael Jones and Joe Hildebrand. JSON Web Encryption (JWE). Request for Comments 7516, Internet Engineering Task Force, 2015.

[JLG08] Paul T. Jaeger, Jimmy Lin, and Justin M. Grimes. Cloud Computing and Information Policy: Computing in a Policy Cloud? *Journal of Information Technology & Politics*, 5(3):269–283, 2008.

[JMR+14] Hubert A. Jäger, Arnold Monitzer, Ralf Rieken, Edmund Ernst, and Khiem Dau Nguyen. Sealed Cloud – A Novel Approach to Safeguard against Insider Attacks. In Helmut Krcmar, Ralf Reussner, and Bernhard Rumpe, editors, *Trusted Cloud Computing*, pages 15–34. Springer, 2014.

[JNC12] YoungHoon Jung, Richard Neill, and Luca P. Carloni. A Broadband Embedded Computing System for MapReduce Utilizing Hadoop. In *Proceedings of the 2012 IEEE 4th International Conference on Cloud Computing Technology and Science (CloudCom)*, pages 1–9. IEEE, 2012.

[Jon15] Michael Jones. JSON Web Key (JWK). Request for Comments 7517, Internet Engineering Task Force, 2015.

[JRSJ15] Mosarrat Jahan, Mohsen Rezvani, Aruna Seneviratne, and Sanjay Jha. Method for Providing Secure and Private Fine-grained Access to Outsourced Data. In *Proceedings of the 2015 IEEE 40th Conference on Local Computer Networks (LCN)*, pages 406–409. IEEE, 2015.

[JSA+17] Cullen Jennings, Zach Shelby, Jari Arkko, Ari Keränen, and Carsten Bormann. Media Types for Sensor Measurement Lists (SenML). Internet-Draft draft-ietf-core-senml-11, Internet Engineering Task Force, 2017. Work in Progress.

[JZV+12] Martin Gilje Jaatun, Gansen Zhao, Athanasios V. Vasilakos, Åsmund Ahlmann Nyre, Stian Alapnes, and Yong Tang. The design of a redundant array of independent net-storages for improved confidentiality in cloud computing. *Journal of Cloud Computing*, 1(1), 2012.

[Kas05] Debbie V. S. Kasper. The Evolution (or Devolution) of Privacy. *Sociological Forum*, 20(1):69–92, 2005.

[KCLC07] Ponnurangam Kumaraguru, Lorrie Faith Cranor, Jorge Lobo, and Seraphin B. Calo. A Survey of Privacy Policy Languages. In *Proceedings of the SOUPS Workshop on Usable IT Security Management (USM)*, 2007.

[KDZ18] Daniel Kerpen, Matthias Dorgeist, and Sascha Zantis. Intersecting the Digital Maze. Considering Ethics in Cloud-Based Services' Research. In Farina Madita Dobrick, Jana Fischer, and Lutz M. Hagen, editors, *Research Ethics in the Digital Age: Ethics for the Social Sciences and Humanities in Times of Mediatization and Digitization*, pages 143–152. Springer, 2018.

[KFJ03] Lalana Kagal, Tim Finin, and Anupam Joshi. A policy language for a pervasive computing environment. In *Proceedings of the IEEE 4th International Workshop on Policies for Distributed Systems and Networks (POLICY)*, pages 63–74. IEEE, 2003.

[Kin17] Rachel King. Here's Why Amazon's Cloud Suffered a Meltdown This Week. `http://fortune.com/2017/03/02/amazon-cloud-outage/`, 2017. [Online, accessed 2018-07-01].

[Kit14] Scott Kitterman. Sender Policy Framework (SPF) for Authorizing Use of Domains in Email, Version 1. Request for Comments 7208, Internet Engineering Task Force, 2014.

[KJK16] Hajoon Ko, Jiong Jin, and Sye Loong Keoh. Secure Service Virtualization in IoT by Dynamic Service Dependency Verification. *IEEE Internet of Things Journal*, 3(6):1006–1014, 2016.

[KJK17] Hajoon Ko, Jiong Jin, and Sye Loong Keoh. ViotSOC: Controlling Access to Dynamically Virtualized IoT Services using Service Object Capability. In *Proceedings of the 3rd ACM Workshop on Cyber-Physical System Security (CPSS)*, pages 69–80. ACM, 2017.

[KKLL09] Won Kim, Soo Dong Kim, Eunseok Lee, and Sungyoung Lee. Adoption Issues for Cloud Computing. In *Proceedings of the 7th International Conference on Advances in Mobile Computing and Multimedia (MoMM)*, pages 2–5. ACM, 2009.

[KL10] Seny Kamara and Kristin Lauter. Cryptographic Cloud Storage. In *Proceedings of the 14th International Conference on Financial Cryptography and Data Security (FC) Workshops*, pages 136–149. Springer, 2010.

[Kle08] John C. Klensin. Simple Mail Transfer Protocol. Request for Comments 5321, Internet Engineering Task Force, 2008.

[KNSV13] Amin M. Khan, Leandro Navarro, Leila Sharifi, and Luís Veiga. Clouds of Small Things: Provisioning Infrastructure-as-a-Service from within Community Networks. In *Proceedings of the 2013 IEEE 9th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob)*, pages 16–21. IEEE, 2013.

[Kop13] Maik Koprowski. Realizing Data Handling Annotation Support in the Cloud Stack with Customized Data Distribution. Bachelor's thesis, RWTH Aachen University, September 2013.

[KPPK11] Prachi Kumari, Alexander Pretschner, Jonas Peschla, and Jens-Michael Kuhn. Distributed Data Usage Control for Web Applications: A Social Network Implementation. In *Proceedings of the First ACM Conference on Data and Application Security and Privacy (CODASPY)*, pages 85–96. ACM, 2011.

[Kra96] Hugo Krawczyk. SKEME: A Versatile Secure Key Exchange Mechanism for Internet. In *Proceedings of Internet Society Symposium on Network and Distributed Systems Security (NDSS)*, pages 114–127. IEEE, 1996.

[Kri14] Aivar Kripsaar. Access Control for Sensor Data in the Cloud. Bachelor's thesis, RWTH Aachen University, January 2014.

[KS17] Minhaj Ahmad Khan and Khaled Salah. IoT security: Review, blockchain solutions, and open challenges. *Future Generation Computer Systems*, 82:395–411, 2017.

[KV10] Ronald L. Krutz and Russell Dean Vines. *Cloud Security: A Comprehensive Guide to Secure Cloud Computing*. Wiley, 2010.

[KY04] Bryan Klimt and Yiming Yang. Introducing the Enron Corpus. In *Proceedings of the First Conference on Email and Anti-Spam (CEAS)*, 2004.

[KYKH16] Mohammad Mahdi Kashef, Hyenyoung Yoon, Mehdi Keshavarz, and Junseok Hwang. Decision Support Tool for IoT Service Providers for Utilization of Multi Clouds. In *Proceedings of the 2016 18th International Conference on Advanced Communication Technology (ICACT)*, pages 91–96. IEEE, 2016.

[Lam81] Leslie Lamport. Password Authentication with Insecure Communication. *Communications of the ACM*, 24(11):770–772, 1981.

[LC16] Adam Langley and Wan-Teh Chang. QUIC Crypto. Technical Report Revision 20161206, Google, 2016.

[Leh14] Hendrik vom Lehn. On data markets as a means to privacy protection: An ethical evaluation of the treatment of personal data as a commodity. Master's thesis, Delft University of Technology, August 2014.

[LFK+14] Heiner Lasi, Peter Fettke, Hans-Georg Kemper, Thomas Feld, and Michael Hoffmann. Industry 4.0. *Business & Information Systems Engineering*, 6(4):239–242, 2014.

[LGW06] Olaf Landsiedel, Stefan Götz, and Klaus Wehrle. Towards Scalable Mobility in Distributed Hash Tables. In *Proceedings of the Sixth IEEE International Conference on Peer-to-Peer Computing (P2P)*, pages 203–209. IEEE, 2006.

[LHBC12] Ahmed Lounis, Abdelkrim Hadjidj, Abdelmadjid Bouabdallah, and Yacine Challal. Secure and Scalable Cloud-Based Architecture for e-Health Wireless Sensor Networks. In *Proceedings of the 2012 21st International Conference on Computer Communications and Networks (ICCCN)*, pages 1–7. IEEE, 2012.

[LHFY13] Songbin Liu, Xiaomeng Huang, Haohuan Fu, and Guangwen Yang. Understanding Data Characteristics and Access Patterns in a Cloud Storage System. In *Proceedings of the 2013 13th IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing (CCGrid)*, pages 327–334. IEEE, 2013.

[LHL15] Jianghua Liu, Xinyi Huang, and Joseph K. Liu. Secure sharing of Personal Health Records in cloud computing: Ciphertext-Policy Attribute-Based Signcryption. *Future Generation Computer Systems*, 52:67–76, 2015.

[Lin00] John Linn. Generic Security Service Application Program Interface Version 2, Update 1. Request for Comments 2743, Internet Engineering Task Force, 2000.

[LLSH14] Jialiu Lin, Bin Liu, Norman Sadeh, and Jason I. Hong. Modeling Users' Mobile App Privacy Preferences: Restoring Usability in a Sea of Permission Settings. In *Proceedings of the Tenth Symposium on Usable Privacy and Security (SOUPS)*, pages 199–212. USENIX, 2014.

[LLV07] Ninghui Li, Tiancheng Li, and Suresh Venkatasubramanian. t-Closeness: Privacy Beyond k-Anonymity and l-Diversity. In *Proceedings of the 2007 IEEE 23rd International Conference on Data Engineering (ICDE)*, pages 106–115. IEEE, 2007.

[LM10] Avinash Lakshman and Prashant Malik. Cassandra: A Decentralized Structured Storage System. *ACM SIGOPS Operating Systems Review*, 44(2):35–40, 2010.

[Loh12] Steve Lohr. The Age of Big Data. `http://www.nytimes.com/2012/02/12/sunday-review/big-datas-impact-in-the-world.html`, 2012. [Online, accessed 2018-07-01].

[LPGD16] Lydia Leong, Gregor Petri, Bob Gill, and Mike Dorosh. Magic Quadrant for Cloud Infrastructure as a Service, Worldwide. Gartner Report G00278620, 2016.

[LSW04] Karthik Lakshminarayanan, Ion Stoica, and Klaus Wehrle. Support for Service Composition in i3. In *Proceedings of the 12th Annual ACM International Conference on Multimedia*, pages 108–111. ACM, 2004.

[LTM+11] Fang Liu, Jin Tong, Jian Mao, Robert Bohn, John Messina, Lee Badger, and Dawn Leaf. NIST Cloud Computing Reference Architecture. NIST Special Publication 500-292, National Institute of Standards and Technology, 2011.

[LVCD13] Fei Li, Michael Voegler, Markus Claessens, and Schahram Dustdar. Efficient and Scalable IoT Service Delivery on Cloud. In *Proceedings of the 2013 IEEE Sixth International Conference on Cloud Computing (CLOUD)*, pages 740–747. IEEE, 2013.

[LVL+15] Anh Le, Janus Varmarken, Simon Langhoff, Anastasia Shuba, Minas Gjoka, and Athina Markopoulou. AntMonitor: A System for Monitoring from Mobile Devices. In *Proceedings of the 2015 ACM SIGCOMM Workshop on Crowdsourcing and Crowdsharing of Big (Internet) Data (C2B(1)D)*, pages 15–20. ACM, 2015.

[LWBL17] Yuzhu Liang, Tian Wang, Md Zakirul Alam Bhuiyan, and Anfeng Liu. Research on Coupling Reliability Problem in Sensor-Cloud System. In *Proceedings of the 10th International Conference on Security, Privacy and Anonymity in Computation, Communication and Storage (SpaCCS)*, pages 468–478. Springer, 2017.

[LYZ+13] Ming Li, Shucheng Yu, Yao Zheng, Kui Ren, and Wenjing Lou. Scalable and Secure Sharing of Personal Health Records in Cloud Computing Using Attribute-Based Encryption. *IEEE Transactions on Parallel and Distributed Systems*, 24(1):131–143, 2013.

[MAB+08] Nick McKeown, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker, and Jonathan Turner. OpenFlow: Enabling Innovation in Campus Networks. *ACM SIGCOMM Computer Communication Review*, 38(2):69–74, 2008.

[Man13] Alessandro Mantelero. The EU Proposal for a General Data Protection Regulation and the roots of the 'right to be forgotten'. *Computer Law & Security Review*, 29(3):229–235, 2013.

[Mar16] Patrick Marx. Behavioural Nudging through Privacy-Preserving Comparisons. Master's thesis, RWTH Aachen University, November 2016.

[Mat13] Roman Matzutt. User-controlled Utilization of Sensor Data for Cloud Computing. Bachelor's thesis, RWTH Aachen University, March 2013.

[MB02] Petros Maniatis and Mary Baker. Secure History Preservation through Timeline Entanglement. In *Proceedings of the 11th USENIX Security Symposium*, pages 297–312. USENIX, 2002.

[MBK+12] Ildar Muslukhov, Yazan Boshmaf, Cynthia Kuo, Jonathan Lester, and Konstantin Beznosov. Understanding Users' Requirements for Data Protection in Smartphones. In *Proceedings of the 2012 IEEE 28th International Conference on Data Engineering Workshops (ICDEW)*, pages 228–235. IEEE, 2012.

[McA16] Rebecca McAdams. The Forrester Wave™: Email Marketing Service Providers, Q3 2016. Forrester Research, Inc., 2016.

[McM12] Robert McMillan. (Real) Storm Crushes Amazon Cloud, Knocks out Netflix, Pinterest, Instagram. `https://www.wired.com/2012/06/real-clouds-crush-amazon/`, 2012. [Online, accessed 2018-07-01].

[ME10] Tyler Moore and Benjamin Edelman. Measuring the Perpetrators and Funders of Typosquatting. In *Proceedings of the 14th International Conference on Financial Cryptography and Data Security (FC)*, pages 175–191. Springer, 2010.

[MFB+15] Simone Mutti, Yanick Fratantonio, Antonio Bianchi, Luca Invernizzi, Jacopo Corbetta, Dhilung Kirat, Christopher Kruegel, and Giovanni Vigna. BareDroid: Large-Scale Analysis of Android Apps on Real Devices. In *Proceedings of the 31st Annual Computer Security Applications Conference (ACSAC)*, pages 71–80. ACM, 2015.

[MG11] Peter Mell and Timothy Grance. The NIST Definition of Cloud Computing. NIST Special Publication 800-145, National Institute of Standards and Technology, 2011.

[MGM+10] Richard Mortier, Chris Greenhalgh, Derek McAuley, Alexa Spence, Anil Madhavapeddy, Jon Crowcroft, and Steven Hand. The Personal Container, or Your Life in Bits. In *Digital Futures Workshop*, 2010.

[MH12] Ming Mao and Marty Humphrey. A Performance Study on the VM Startup Time in the Cloud. In *Proceedings of the 2012 IEEE Fifth International Conference on Cloud Computing (CLOUD)*, pages 423–430. IEEE, 2012.

[MHCK07] Gabriel Montenegro, Jonathan Hui, David Culler, and Nandakishore Kushalnagar. Transmission of IPv6 Packets over IEEE 802.15.4 Networks. Request for Comments 4944, Internet Engineering Task Force, 2007.

[MHH+16] Roman Matzutt, Oliver Hohlfeld, Martin Henze, Robin Rawiel, Jan Henrik Ziegeldorf, and Klaus Wehrle. POSTER: I Don't Want That Content! On the Risks of Exploiting Bitcoin's Blockchain as a

Content Store. In *Proceedings of the 23rd ACM Conference on Computer and Communications Security (CCS)*, pages 1769–1771. ACM, 2016.

[MHH+18] Roman Matzutt, Jens Hiller, Martin Henze, Jan Henrik Ziegeldorf, Dirk Müllmann, Oliver Hohlfeld, and Klaus Wehrle. A Quantitative Analysis of the Impact of Arbitrary Blockchain Content on Bitcoin. In *Proceedings of the 22nd International Conference on Financial Cryptography and Data Security (FC)*. Springer, 2018.

[MHZ+18] Roman Matzutt, Martin Henze, Jan Henrik Ziegeldorf, Jens Hiller, and Klaus Wehrle. Thwarting Unwanted Blockchain Content Insertion. In *Proceedings of the 2018 IEEE International Conference on Cloud Engineering (IC2E)*, pages 364–370. IEEE, 2018.

[Mic94] Giovanni De Micheli. *Synthesis and Optimization of Digital Circuits*. McGraw-Hill, 1st edition, 1994.

[Mic16a] Microsoft. A Cloud for Global Good – A policy roadmap for a trusted, responsible, and inclusive cloud, 2016.

[Mic16b] Microsoft Azure. Azure Regions. `https://azure.microsoft.com/en-us/regions/`, 2016. [Online, accessed 2016-09-08].

[Mic17] Microsoft. Microsoft Security Intelligence Report, Volume 22, January through March, 2017.

[Mil13] Christopher Millard, editor. *Cloud Computing Law*. Oxford University Press, 2013.

[Mil16] Ron Miller. How AWS came to be. `https://techcrunch.com/2016/07/02/andy-jassys-brief-history-of-the-genesis-of-aws/`, 2016. [Online, accessed 2018-07-01].

[MJ17] Azizbek Marakhimov and Jaehun Joo. Consumer adaptation and infusion of wearable devices for healthcare. *Computers in Human Behavior*, 76:135–148, 2017.

[MKGV07] Ashwin Machanavajjhala, Daniel Kifer, Johannes Gehrke, and Muthuramakrishnan Venkitasubramaniam. l-Diversity: Privacy Beyond k-Anonymity. *ACM Transactions on Knowledge Discovery from Data*, 1(1), 2007.

[MKH+13] Philip Mayer, Annabelle Klarl, Rolf Hennicker, Mariachiara Puviani, Francesco Tiezzi, Rosario Pugliese, Jaroslav Keznikl, and Tomáš Bureš. The Autonomic Cloud: A Vision of Voluntary, Peer-2-Peer Cloud Computing. In *Proceedings of the 2013 IEEE 7th International Conference on Self-Adaptation and Self-Organizing Systems Workshops (SASOW)*, pages 89–94. IEEE, 2013.

[MKL09] Tim Mather, Subra Kumaraswamy, and Shahed Latif. *Cloud Security and Privacy: An Enterprise Perspective on Risks and Compliance*. O'Reilly, 2009.

[MLB+11] Sean Marston, Zhi Li, Subhajyoti Bandyopadhyay, Juheng Zhang, and Anand Ghalsasi. Cloud computing – The business perspective. *Decision Support Systems*, 51(1):176–189, 2011.

[MM10] Frank McSherry and Ratul Mahajan. Differentially-Private Network Trace Analysis. In *Proceedings of the ACM SIGCOMM 2010 Conference*, pages 123–134. ACM, 2010.

[MMOT14] Pieter-Jan Maenhaut, Hendrik Moens, Veerle Ongenae, and Filip De Turck. Scalable User Data Management in Multi-Tenant Cloud Environments. In *Proceedings of the 2014 10th International Conference on Network and Service Management (CNSM)*, pages 268–271. IEEE, 2014.

[MMOT15] Pieter-Jan Maenhaut, Hendrik Moens, Veerle Ongenae, and Filip De Turck. Design and Evaluation of a Hierarchical Multi-Tenant Data Management Framework for Cloud Applications. In *Proceedings of the 2015 IFIP/IEEE International Symposium on Integrated Network Management (IM)*, pages 1208–1213. IEEE, 2015.

[MMV+15] Pieter-Jan Maenhaut, Hendrik Moens, Bruno Volckaert, Veerle Ongenae, and Filip De Turck. Design of a Hierarchical Software-Defined Storage System for Data-Intensive Multi-Tenant Cloud Applications. In *Proceedings of the 2015 11th International Conference on Network and Service Management (CNSM)*, pages 22–28. IEEE, 2015.

[MMV+17] Pieter-Jan Maenhaut, Hendrik Moens, Bruno Volckaert, Veerle Ongenae, and Filip De Turck. A Dynamic Tenant-Defined Storage System for Efficient Resource Management in Cloud Applications. *Journal of Network and Computer Applications*, 93(Supplement C):182–196, 2017.

[MMZ+17] Roman Matzutt, Dirk Müllmann, Eva-Maria Zeissig, Christiane Horst, Kai Kasugai, Sean Lidynia, Simon Wieninger, Jan Henrik Ziegeldorf, Gerhard Gudergan, Indra Spiecker gen. Döhmann, Klaus Wehrle, and Martina Ziefle. myneData: Towards a Trusted and User-controlled Ecosystem for Sharing Personal Data. In *Proceedings of INFORMATIK 2017*, pages 1073–1084. Gesellschaft für Informatik, 2017.

[MNP+11] Philippe Massonet, Syed Naqvi, Christophe Ponsard, Joseph Latanicki, Benny Rochwerger, and Massimo Villari. A Monitoring and Audit Logging Architecture for Data Location Compliance in Federated Cloud Infrastructures. In *Proceedings of the 2011 IEEE International Symposium on Parallel and Distributed Processing Workshops and Phd Forum (IPDPSW)*, pages 1510–1517. IEEE, 2011.

[Moc87] Paul V. Mockapetris. Domain names – concepts and facilities. Request for Comments 1034, Internet Engineering Task Force, 1987.

[MPP+08] Jonathan M. McCune, Bryan J. Parno, Adrian Perrig, Michael K. Reiter, and Hiroshi Isozaki. Flicker: An Execution Infrastructure for TCB Minimization. In *Proceedings of the 3rd ACM SIGOPS/EuroSys European Conference on Computer Systems 2008 (EuroSys)*, pages 315–328. ACM, 2008.

[MPS+13] Delfina Malandrino, Andrea Petta, Vittorio Scarano, Luigi Serra, Raffaele Spinelli, and Balachander Krishnamurthy. Privacy Awareness About Information Leakage: Who Knows What About Me? In *Proceedings of the 12th ACM Workshop on Workshop on Privacy in the Electronic Society (WPES)*, pages 279–284. ACM, 2013.

[MRAA17] Ghulam Muhammad, SK Md Mizanur Rahman, Abdulhameed Alelaiwi, and Atif Alamri. Smart Health Solution Integrating IoT and Cloud: A Case Study of Voice Pathology Monitoring. *IEEE Communications Magazine*, 55(1):69–73, 2017.

[MS10] Krish Muralidhar and Rathindra Sarathy. Does Differential Privacy Protect Terry Gross' Privacy? In *Proceedings of the International Conference on Privacy in Statistical Databases (PSD)*, pages 200–209. Springer, 2010.

[MSPC12] Daniele Miorandi, Sabrina Sicari, Francesco De Pellegrini, and Imrich Chlamtac. Internet of Things: Vision, Applications and Research Challenges. *Ad Hoc Networks*, 10(7):1497–1516, 2012.

[MSWP14] Yves-Alexandre de Montjoye, Erez Shmueli, Samuel S. Wang, and Alex Sandy Pentland. openPDS: Protecting the Privacy of Metadata through SafeAnswers. *PLOS ONE*, 9(7), 2014.

[MT09] Di Ma and Gene Tsudik. A New Approach to Secure Logging. *ACM Transactions on Storage*, 5(1):2:1–2:21, 2009.

[Müh14] Erik Mühmer. Analyzing Cloud Usage by Observing Network Traffic. Bachelor's thesis, RWTH Aachen University, September 2014.

[Nak08] Satoshi Nakamoto. Bitcoin: A Peer-to-Peer Electronic Cash System, 2008.

[NG15] Jan Kristof Nidzwetzki and Ralf Hartmut Güting. Distributed SECONDO: A Highly Available and Scalable System for Spatial Data Processing. In *Proceedings of the 14th International Symposium on Spatial and Temporal Databases (SSTD)*, pages 491–496. Springer, 2015.

[NLB13] Rimma V. Nehme, Hyo-Sang Lim, and Elisa Bertino. Fence: Continuous access control enforcement in dynamic data stream environments. In *Proceedings of the Third ACM Conference on Data and Application Security and Privacy (CODASPY)*, pages 243–254. ACM, 2013.

[NSV+15] David Naylor, Kyle Schomp, Matteo Varvello, Ilias Leontiadis, Jeremy Blackburn, Diego R. López, Konstantina Papagiannaki, Pablo Rodriguez Rodriguez, and Peter Steenkiste. Multi-Context TLS (mcTLS): Enabling Secure In-Network Functionality in TLS. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication (SIGCOMM)*, pages 199–212. ACM, 2015.

[NWZ12] Wee Keong Ng, Yonggang Wen, and Huafei Zhu. Private Data Deduplication Protocols in Cloud Storage. In *Proceedings of the 27th Annual ACM Symposium on Applied Computing*, pages 441–446. ACM, 2012.

[Oas13] OASIS Open. eXtensible Access Control Markup Language (XACML) Version 3.0. OASIS Standard, 2013.

[OECD80] Organisation for Economic Co-operation and Development. OECD Guidelines on the Protection of Privacy and Transborder Flows of Personal Data, 1980.

[Ölc13] Devran Ölcer. Efficient Signature Schemes for Sensor Data in the Cloud. Master's thesis, RWTH Aachen University, November 2013.

[OSGJ13] Anderson Santana De Oliveira, Jakub Sendor, Alexander Garaga, and Kateline Jenatton. Monitoring Personal Data Transfers in the Cloud. In *Proceedings of the 2013 IEEE 5th International Conference on Cloud Computing Technology and Science (CloudCom)*, pages 347–354. IEEE, 2013.

[ÖV11] M. Tamer Özsu and Patrick Valduriez. *Principles of Distributed Database Systems*. Springer, 3rd edition, 2011.

[Own18] ownCloud – The last cloud collaboration platform you'll ever need. `https://owncloud.org/`, 2018. [Online, accessed 2018-07-01].

[Pai99] Pascal Paillier. Public-Key Cryptosystems Based on Composite Degree Residuosity Classes. In *Proceedings of the International Conference on the Theory and Application of Cryptographic Techniques (EUROCRYPT)*, pages 223–238. Springer, 1999.

[PAS17] Luis Pacheco, Eduardo Alchieri, and Priscila Solis. Architecture for Privacy in Cloud of Things. In *Proceedings of the 19th International Conference on Enterprise Information Systems (ICEIS)*, volume 2, pages 487–494. SciTePress, 2017.

[PB10] Siani Pearson and Azzedine Benameur. Privacy, Security and Trust Issues Arising from Cloud Computing. In *Proceedings of the 2010 IEEE Second International Conference on Cloud Computing Technology and Science (CloudCom)*, pages 693–702. IEEE, 2010.

[PBS+15] Pawani Porambage, An Braeken, Corinna Schmitt, Andrei Gurtov, Mika Ylianttila, and Burkhard Stiller. Group Key Establishment for

Secure Multicasting in IoT-enabled Wireless Sensor Networks. In *Proceedings of the 2015 IEEE 40th Conference on Local Computer Networks (LCN)*, pages 482–485. IEEE, 2015.

[PBSE16] Thomas F. J.-M. Pasquier, Jean Bacon, Jatinder Singh, and David Eyers. Data-Centric Access Control for Cloud Computing. In *Proceedings of the 21st ACM on Symposium on Access Control Models and Technologies (SACMAT)*, pages 81–88. ACM, 2016.

[PCB15] Mithun Paul, Christian Collberg, and Derek Bambauer. A Possible Solution for Privacy Preserving Cloud Data Storage. In *Proceedings of the 2015 IEEE International Conference on Cloud Engineering (IC2E)*, pages 397–403. IEEE, 2015.

[PCI15] PCI Security Standards Council. Payment Card Industry (PCI) Data Security Standard – Requirements and Security Assessment Procedures, Version 3.1, 2015.

[PDG+16] Maria Rita Palattella, Mischa Dohler, Alfredo Grieco, Gianluca Rizzo, Johan Torsner, Thomas Engel, and Latif Ladid. Internet of Things in the 5G Era: Enablers, Architecture and Business Models. *IEEE Journal on Selected Areas in Communications*, 34(3):510–527, 2016.

[Pea09] Siani Pearson. Taking Account of Privacy when Designing Cloud Computing Services. In *Proceedings of the 2009 ICSE Workshop on Software Engineering Challenges of Cloud Computing (CLOUD)*, pages 44–52. IEEE, 2009.

[Pea13] Siani Pearson. Privacy, Security and Trust in Cloud Computing. In Siani Pearson and George Yee, editors, *Privacy and Security for Cloud Computing*, chapter 1, pages 3–42. Springer, 2013.

[Per15] Cristian Perra. A Framework for User Control Over Media Data Based on a Trusted Point. In *Proceedings of the 2015 IEEE International Conference on Consumer Electronics (ICCE)*, pages 1–2. IEEE, 2015.

[Per17] Nicole Perlroth. All 3 Billion Yahoo Accounts Were Affected by 2013 Attack. `https://www.nytimes.com/2017/10/03/technology/yahoo-hack-3-billion-users.html`, 2017. [Online, accessed 2018-07-01].

[PFNW12] Paul Pearce, Adrienne Porter Felt, Gabriel Nunez, and David Wagner. AdDroid: Privilege Separation for Applications and Advertisers in Android. In *Proceedings of the 7th Symposium on Information, Computer and Communications Security (ASIACCS)*, pages 71–72. ACM, 2012.

[PGB11] Zachary N. J. Peterson, Mark Gondree, and Robert Beverly. A Position Paper on Data Sovereignty: The Importance of Geolocating Data in the Cloud. In *Proceedings of the 3rd USENIX Workshop on Hot Topics in Cloud Computing (HotCloud)*. USENIX, 2011.

[PHF15] Enric Pujol, Oliver Hohlfeld, and Anja Feldmann. Annoyed Users: Ads and Ad-Block Usage in the Wild. In *Proceedings of the 2015 Internet Measurement Conference (IMC)*, pages 93–106. ACM, 2015.

[PHW17] Jan Pennekamp, Martin Henze, and Klaus Wehrle. A Survey on the Evolution of Privacy Enforcement on Smartphones and the Road Ahead. *Pervasive and Mobile Computing*, 42:58–76, 2017.

[PIPE00] Parliament of Canada. Personal Information Protection and Electronic Documents Act (PIPEDA). S.C. 2000, c. 5, 2000.

[PJ12] Jayaraj Poroor and Bharat Jayaraman. C2L: A formal policy language for secure cloud configurations. In *Proceedings of the 3rd International Conference on Ambient Systems, Networks and Technologies (ANT)*, pages 499–506. Elsevier, 2012.

[Pla99] John C. Platt. Fast Training of Support Vector Machines Using Sequential Minimal Optimization. In Christopher J. C. Burges, Bernhard Schölkopf, and Alexander J. Smola, editors, *Advances in Kernel Methods: Support Vector Learning*, chapter 12. MIT Press, 1999.

[Plu17] Libby Plummer. Volkswagen Saves Time and Money by Moving to a Private Cloud Network. `https://www.intel.co.uk/content/www/uk/en/it-managers/volkswagen-private-cloud.html`, 2017. [Online, accessed 2017-08-31].

[PLZ+16] Andriy Panchenko, Fabian Lanze, Andreas Zinnen, Martin Henze, Jan Pennekamp, Klaus Wehrle, and Thomas Engel. Website Fingerprinting at Internet Scale. In *Proceedings of the 23rd Annual Network and Distributed System Security Symposium (NDSS)*. The Internet Society, 2016.

[PM11] Siani Pearson and Marco Casassa Mont. Sticky Policies: An Approach for Managing Privacy across Multiple Parties. *Computer*, 44(9):60–68, 2011.

[PMCR11] Siani Pearson, Marco Casassa Mont, Liqun Chen, and Archie Reed. End-to-End Policy-Based Encryption and Management of Data in the Cloud. In *Proceedings of the 2011 IEEE Third International Conference on Cloud Computing Technology and Science (CloudCom)*, pages 764–771. IEEE, 2011.

[PMH+17] Andriy Panchenko, Asya Mitseva, Martin Henze, Fabian Lanze, Klaus Wehrle, and Thomas Engel. Analysis of Fingerprinting Techniques for Tor Hidden Services. In *Proceedings of the 15th ACM Workshop on Privacy in the Electronic Society (WPES)*, pages 165–175. ACM, 2017.

[Pos81] Jon Postel. Internet Protocol. Request for Comments 791, Internet Engineering Task Force, 1981.

[PP12] Ioannis Papagiannis and Peter Pietzuch. CloudFilter: Practical Control of Sensitive Data Propagation to the Cloud. In *Proceedings of the 2012 ACM Cloud Computing Security Workshop (CCSW)*, pages 97–102. ACM, 2012.

[PP15] Thomas F. J.-M. Pasquier and Julia E. Powles. Expressing and Enforcing Location Requirements in the Cloud Using Information Flow Control. In *Proceedings of the 2015 IEEE International Conference on Cloud Engineering (IC2E)*, pages 410–415. IEEE, 2015.

[PPL14] PPL FI-WARE Data Handling Generic Enabler. `https://github.com/fdicerbo/fiware-ppl`, 2014. [Online, accessed 2018-07-01].

[PPP13] Boja Pooja, M. M. Manohara Pai, and Radhika M. Pai. A Dual Cloud Based Secure Environmental Parameter Monitoring System: A WSN Approach. In *Proceedings of the 4th International Conference on Cloud Computing (CloudComp 2013)*, pages 189–198. Springer, 2013.

[PQ95] Terence J. Parr and Russell W. Quong. ANTLR: A predicated-LL(k) parser generator. *Software: Practice and Experience*, 25(7):789–810, 1995.

[PRZB11] Raluca Ada Popa, Catherine M. S. Redfield, Nickolai Zeldovich, and Hari Balakrishnan. CryptDB: Protecting Confidentiality with Encrypted Query Processing. In *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles (SOSP)*, pages 85–100. ACM, 2011.

[PSBE16] Thomas F. J.-M. Pasquier, Jatinder Singh, Jean Bacon, and David Eyers. Information Flow Audit for PaaS Clouds. In *Proceedings of the 2016 IEEE International Conference on Cloud Engineering (IC2E)*, pages 42–51. IEEE, 2016.

[PSM09] Siani Pearson, Yun Shen, and Miranda Mowbray. A Privacy Manager for Cloud Computing. In *Proceedings of the First International Conference on Cloud Computing (CloudCom)*, pages 90–106. Springer, 2009.

[PTPS14] Pablo Picazo-Sanchez, Juan E. Tapiador, Pedro Peris-Lopez, and Guillermo Suarez-Tangil. Secure Publish-Subscribe Protocols for Heterogeneous Medical Wireless Body Area Networks. *Sensors*, 14(12):22619–22642, 2014.

[PUK+11] Ingmar Poese, Steve Uhlig, Mohamed Ali Kaafar, Benoit Donnet, and Bamba Gueye. IP Geolocation Databases: Unreliable? *ACM SIGCOMM Computer Communication Review*, 41(2):53–56, 2011.

[Pul15] John Patrick Pullen. Where Did Cloud Computing Come From, Anyway? `http://time.com/collection-post/3750915/cloud-computing-origin-story/`, 2015. [Online, accessed 2018-07-01].

[QG12] Han Qi and Abdullah Gani. Research on Mobile Cloud Computing: Review, Trend and Perspectives. In *Proceedings of the 2012 Second International Conference on Digital Information and Communication Technology and it's Applications (DICTAP)*, pages 195–202. IEEE, 2012.

[RA14] Lee Rainie and Janna Anderson. The Future of Privacy. Pew Research Center, 2014.

[Rad16] The Radicati Group, Inc. Email Statistics Report, 2016–2020 (Executive Summary), 2016.

[RBM16] Christian David Gómez Romero, July Katherine Díaz Barriga, and José Ignacio Rodríguez Molano. Big data meaning in the architecture of IoT for smart cities. In *Proceedings of the First International Conference on Data Mining and Big Data (DMBD)*, pages 457–465. Springer, 2016.

[RDGT08] Robbert van Renesse, Dan Dumitriu, Valient Gough, and Chris Thomas. Efficient Reconciliation and Flow Control for Anti-entropy Protocols. In *Proceedings of the 2nd Workshop on Large-Scale Distributed Systems and Middleware (LADIS)*, pages 6:1–6:7. ACM, 2008.

[Res01] Peter W. Resnick. Internet Message Format. Request for Comments 2822, Internet Engineering Task Force, 2001.

[RF06] Anirudh Ramachandran and Nick Feamster. Understanding the Network-level Behavior of Spammers. In *Proceedings of the 2006 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM)*, pages 291–302. ACM, 2006.

[RFVE11] Thorsten Ries, Volker Fusenig, Christian Vilbois, and Thomas Engel. Verification of Data Location in Cloud Networking. In *Proceedings of the 2011 Fourth IEEE International Conference on Utility and Cloud Computing (UCC)*, pages 439–444. IEEE, 2011.

[RG10] Karen Renaud and Dora Gálvez-Cruz. Privacy: Aspects, Definitions and a Multi-Faceted Privacy Preservation Approach. In *Proceedings of the 2010 Information Security for South Africa Conference (ISSA)*, pages 1–8. IEEE, 2010.

[RGS+12] Tilmann Rabl, Sergio Gómez-Villamor, Mohammad Sadoghi, Victor Muntés-Mulero, Hans-Arno Jacobsen, and Serge Mankovskii. Solving Big Data Challenges for Enterprise Application Performance Management. *Proceedings of the VLDB Endowment*, 5(12):1724–1735, 2012.

[Rig17] RightScale, Inc. RightScale 2017 State of the Cloud Report, 2017.

[RJSP16] Roland van Rijswijk-Deij, Mattijs Jonker, Anna Sperotto, and Aiko Pras. A High-Performance, Scalable Infrastructure for Large-Scale Active DNS Measurements. *IEEE Journal on Selected Areas in Communications*, 34(6):1877–1888, 2016.

[RKB+13] Scott Ruoti, Nathan Kim, Ben Burgon, Timothy van der Horst, and Kent Seamons. Confused Johnny: When Automatic Encryption Leads to Confusion and Mistakes. In *Proceedings of the Ninth Symposium on Usable Privacy and Security (SOUPS)*, pages 5:1–5:12. ACM, 2013.

[RKW+10] Carlos Oberdan Rolim, Fernando Luiz Koch, Carlos Becker Westphall, Jorge Werner, Armando Fracalossi, and Giovanni Schmitt Salvador. A Cloud Computing Solution for Patient's Data Collection in Health Care Institutions. In *Proceedings of the Second International Conference on eHealth, Telemedicine, and Social Medicine (ETELEMED)*, pages 95–99. IEEE, 2010.

[RKW12] Franziska Roesner, Tadayoshi Kohno, and David Wetherall. Detecting and Defending Against Third-Party Tracking on the Web. In *Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation (NSDI)*. USENIX, 2012.

[RMX+15] Ulrich Rührmair, J. L. Martinez-Hurtado, Xiaolin Xu, Christian Kraeh, Christian Hilgers, Dima Kononchuk, Jonathan J. Finley, and Wayne P. Burleson. Virtual Proofs of Reality and their Physical Implementation. In *Proceedings of the 2015 IEEE Symposium on Security and Privacy (SP)*, pages 70–85. IEEE, 2015.

[Rob09] William Jeremy Robison. Free at What Cost?: Cloud Computing Privacy Under the Stored Communications Act. *The Georgetown Law Journal*, 98:1195–1239, 2009.

[Ros12] Jeffrey Rosen. The Right to Be Forgotten. *Stanford Law Review Online*, 64:88–92, 2012.

[RRL+16] Jingjing Ren, Ashwin Rao, Martina Lindorfer, Arnaud Legout, and David Choffnes. ReCon: Revealing and Controlling PII Leaks in Mobile Network Traffic. In *Proceedings of the 14th Annual International Conference on Mobile Systems, Applications, and Services (MobiSys)*, pages 361–374. ACM, 2016.

[RTSS09] Thomas Ristenpart, Eran Tromer, Hovav Shacham, and Stefan Savage. Hey, You, Get off of My Cloud: Exploring Information Leakage in Third-party Compute Clouds. In *Proceedings of the 16th ACM Conference on Computer and Communications Security (CCS)*, pages 199–212. ACM, 2009.

[RVS+16] Abbas Razaghpanah, Narseo Vallina-Rodriguez, Srikanth Sundaresan, Christian Kreibich, Phillipa Gill, Mark Allman, and Vern Paxson. Haystack: A Multi-Purpose Mobile Vantage Point in User Space. arXiv preprint arXiv:1510.01419 [cs.NI], 2016.

[Rya14] Mark D. Ryan. Enhanced Certificate Transparency and End-to-End Encrypted Mail. In *Proceedings of the 21st Annual Network and Distributed System Security Symposium (NDSS)*. The Internet Society, 2014.

[RZO+17] Lukas Rupprecht, Rui Zhang, Bill Owen, Peter Pietzuch, and Dean Hildebrand. SwiftAnalytics: Optimizing Object Storage for Big Data Analytics. In *Proceedings of the 2017 IEEE International Conference on Cloud Engineering (IC2E)*, pages 245–251. IEEE, 2017.

[San06] Salvatore Sanfilippo. hping. `http://www.hping.org/`, 2006. [Online, accessed 2018-07-01].

[San16a] Sandvine. 2016 Global Internet Phenomena – Latin America & North America, 2016.

[San16b] Mary Peyton Sanford. Mail Analyzer: Analyzing cloud-based email use. Internship report (Undergraduate Research Opportunities Program), RWTH Aachen University and University of Pennsylvania, July 2016.

[SBC+14] Jatinder Singh, Jean Bacon, Jon Crowcroft, Anil Madhavapeddy, Thomas F. J.-M. Pasquier, W. Kuan Hon, and Christopher Millard. Regional clouds: technical considerations. Technical Report UCAM-CL-TR-863, University of Cambridge, Computer Laboratory, 2014.

[SBHD17] Hossein Shafagh, Lukas Burkhalter, Anwar Hithnawi, and Simon Duquennoy. Towards Blockchain-based Auditable Storage and Sharing of IoT Data. In *Proceedings of the 2017 Cloud Computing Security Workshop (CCSW)*, pages 45–50. ACM, 2017.

[SCF+15] Felix Schuster, Manuel Costa, Cédric Fournet, Christos Gkantsidis, Marcus Peinado, Gloria Mainar-Ruiz, and Mark Russinovich. VC3: Trustworthy Data Analytics in the Cloud Using SGX. In *Proceedings of the 2015 IEEE Symposium on Security and Privacy (SP)*, pages 38–54. IEEE, 2015.

[Sch15] Sascha Schmerling. A Space and Processing Efficient Cloud Privacy Policy Language. Master's thesis, RWTH Aachen University, December 2015.

[SCR+17] Gang Sun, Victor Chang, Muthu Ramachandran, Zhili Sun, Gangmin Li, Hongfang Yu, and Dan Liao. Efficient location privacy algorithm for internet of things (iot) services and applications. *Journal of Network and Computer Applications*, 89:3–13, 2017.

[SCZ+16] Weisong Shi, Jie Cao, Quan Zhang, Youhuizi Li, and Lanyu Xu. Edge Computing: Vision and Challenges. *IEEE Internet of Things Journal*, 3(5):637–646, 2016.

[SD16] Weisong Shi and Schahram Dustdar. The Promise of Edge Computing. *Computer*, 49(5):78–81, 2016.

[SDW12] Shashi Shekhar, Michael Dietz, and Dan S. Wallach. AdSplit: Separating Smartphone Advertising from Applications. In *Proceedings of the 21st USENIX Security Symposium*, pages 28–28. USENIX, 2012.

[SDX11] H. Jeff Smith, Tamara Dinev, and Heng Xu. Information Privacy Research: An Interdisciplinary Review. *MIS Quarterly*, 35(4):989–1016, 2011.

[Sea18] Seafile – Open Source File Sync and Share Software. `https://www.seafile.com/`, 2018. [Online, accessed 2018-07-01].

[See13] Marc Seebold. Privacy-aware Operations on Encrypted Sensor Data in the Cloud. Bachelor's thesis, RWTH Aachen University, March 2013.

[Seu15] Annika Seufert. Load Balancing for Data Handling-aware Distributed Databases. Bachelor's thesis, RWTH Aachen University, June 2015.

[SG16] Mariusz Slabicki and Krzysztof Grochla. Performance Evaluation of CoAP, SNMP and NETCONF Protocols in Fog Computing Architecture. In *Proceedings of the 2016 IEEE/IFIP Network Operations and Management Symposium (NOMS)*, pages 1315–1319. IEEE, 2016.

[SH15] Yihang Song and Urs Hengartner. PrivacyGuard: A VPN-based Platform to Detect Information Leakage on Android Devices. In *Proceedings of the 5th Annual ACM CCS Workshop on Security and Privacy in Smartphones and Mobile Devices (SPSM)*, pages 15–26. ACM, 2015.

[SHH+18] Martin Serror, Martin Henze, Sacha Hack, Marko Schuba, and Klaus Wehrle. Towards In-Network Security for Smart Homes. In *Proceedings of the International Conference on Availability, Reliability and Security (ARES)*. ACM, 2018.

[SHI+13] Benjamin Satzger, Waldemar Hummer, Christian Inzinger, Philipp Leitner, and Schahram Dustdar. Winds of Change: From Vendor Lock-In to the Meta Cloud. *IEEE Internet Computing*, 17(1):69–73, 2013.

[SHKV14] Gianluca Stringhini, Oliver Hohlfeld, Christopher Kruegel, and Giovanni Vigna. The Harvester, the Botmaster, and the Spammer: On the Relations Between the Different Actors in the Spam Landscape. In *Proceedings of the 9th ACM Symposium on Information, Computer and Communications Security (ASIACCS)*, pages 353–364. ACM, 2014.

[Sil13] Karine e Silva. Europe's fragmented approach towards cyber security. *Internet Policy Review*, 2(4), 2013.

[Sim91] Herbert A. Simon. Bounded Rationality and Organizational Learning. *Organization Science*, 2(1):125–134, 1991.

[SK99] Bruce Schneier and John Kelsey. Secure Audit Logs to Support Computer Forensics. *ACM Transactions on Information and System Security*, 2(2):159–176, 1999.

[SKS15] Suranga Seneviratne, Harini Kolamunna, and Aruna Seneviratne. A Measurement Study of Tracking in Paid Mobile Applications. In *Proceedings of the 8th ACM Conference on Security & Privacy in Wireless and Mobile Networks (WiSec)*, pages 7:1–7:6. ACM, 2015.

[Sky16] Skyhigh. Cloud Adoption & Risk Report Q4 2016, 2016.

[SM12] Andreas Schaad and Anja Monakva. Annotating Business Processes with Usage Controls. In *Proceedings of the WWW 2012 workshop on Data Usage Management on the Web (DUMW)*, pages 23–28. Technical University of Munich, 2012.

[Smi12] Ian G. Smith, editor. *The Internet of Things 2012 – New Horizons*. IERC, 2012.

[SMS11] Sumit Sanghrajka, Nilesh Mahajan, and Radu Sion. Cloud Performance Benchmark Series: Network Performance – Amazon EC2, ver. 0.2. Cloud Commons Online, 2011.

[SMS13] Josef Spillner, Johannes Müller, and Alexander Schill. Creating optimal cloud storage systems. *Future Generation Computer Systems*, 29(4):1062–1072, 2013.

[SMSD10] Dominik Schatzmann, Wolfgang Mühlbauer, Thrasyvoulos Spyropoulos, and Xenofontas Dimitropoulos. Digging into HTTPS: Flow-based Classification of Webmail Traffic. In *Proceedings of the 10th ACM SIGCOMM Conference on Internet Measurement (IMC)*, pages 322–327. ACM, 2010.

[Sol06] Daniel J. Solove. A Taxonomy of Privacy. *University of Pennsylvania Law Review*, 154(3):477–560, 2006.

[SOX02] United States Congress. Sarbanes-Oxley Act (SOX). Pub.L. 107–204, 116 Stat. 745, 2002.

[SPB15] Jatinder Singh, Thomas F. J.-M. Pasquier, and Jean Bacon. Securing Tags to Control Information Flows within the Internet of Things. In *Proceedings of the 2015 International Conference on Recent Advances in Internet of Things (RIoT)*, pages 1–6. IEEE, 2015.

[SPB+16] Jatinder Singh, Thomas F. J.-M. Pasquier, Jean Bacon, Hajoon Ko, and David Eyers. Twenty Security Considerations for Cloud-Supported Internet of Things. *IEEE Internet of Things Journal*, 3(3):269–284, 2016.

[SPP01] Dawn Song, Adrian Perrig, and Doantam Phan. AGVI — Automatic Generation, Verification, and Implementation of Security Protocols. In *Proceedings of the 13th International Conference on Computer Aided Verification (CAV)*, pages 241–245. Springer, 2001.

[SRLO15] Johannes Sametinger, Jerzy Rozenblit, Roman Lysecky, and Peter Ott. Security Challenges for Medical Devices. *Communications of the ACM*, 58(4):74–82, 2015.

[SS75] Jerome H. Saltzer and Michael D. Schroeder. The Protection of Information in Computer Systems. *Proceedings of the IEEE*, 63(9):1278–1308, 1975.

[SSFS12] Dawn Song, Elaine Shi, Ian Fischer, and Umesh Shankar. Cloud Data Protection for the Masses. *Computer*, 45(1), 2012.

[SSL12] Smitha Sundareswaran, Anna Squicciarini, and Dan Lin. Ensuring Distributed Accountability for Data Sharing in the Cloud. *IEEE Transactions on Dependable and Secure Computing*, 9(4):556–568, 2012.

[SSY+16] Chad Spensky, Jeffrey Stewart, Arkady Yerukhimovich, Richard Shay, Ari Trachtenberg, Rick Housley, and Robert K. Cunningham. SoK: Privacy on Mobile Devices – It's Complicated. *Proceedings on Privacy Enhancing Technologies (PoPETS)*, 2016(3):96–116, 2016.

[Sta14] John A. Stankovic. Research Directions for the Internet of Things. *IEEE Internet of Things Journal*, 1(1):3–9, 2014.

[STW12] Robin Seggelmann, Michael Tuexen, and Michael Glenn Williams. Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS) Heartbeat Extension. Request for Comments 6520, Internet Engineering Task Force, 2012.

[SV10] Pierangela Samarati and Sabrina De Capitani di Vimercati. Data Protection in Outsourcing Scenarios: Issues and Directions. In *Proceedings of the 5th ACM Symposium on Information, Computer and Communications Security (ASIACSS)*, pages 1–14. ACM, 2010.

[SW13] Michael Stonebraker and Ariel Weisberg. The VoltDB Main Memory DBMS. *IEEE Data Engineering Bulletin*, 36(2):21–27, 2013.

[SW14] Ivan Stojmenovic and Sheng Wen. The fog computing paradigm: Scenarios and security issues. In *Proceedings of the 2014 Federated Conference on Computer Science and Information Systems (FedCSIS)*, pages 1–8. IEEE, 2014.

[Swe00] Latanya Sweeney. Simple Demographics Often Identify People Uniquely. Data Privacy Working Paper 3, Carnegie Mellon University, 2000.

[Swe02] Latanya Sweeney. k-anonymity: A model for protecting privacy. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 10(5):557–570, 2002.

[SWL16] Mingshen Sun, Tao Wei, and John Lui. TaintART: A Practical Multi-level Information-Flow Tracking System for Android RunTime. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security (CCS)*, pages 331–342. ACM, 2016.

[SWW15] Ahmad-Reza Sadeghi, Christian Wachsmann, and Michael Waidner. Security and Privacy Challenges in Industrial Internet of Things. In *Proceedings of the 52nd Annual Design Automation Conference (DAC)*, pages 54:1–54:6. ACM, 2015.

[SWZC16] Quirin Scheitle, Matthias Wachs, Johannes Zirngibl, and Georg Carle. Analyzing Locality of Mobile Messaging Traffic using the MATAdOR Framework. In *Proceedings of the 17th International Conference on Passive and Active Measurement (PAM)*, pages 190–202. Springer, 2016.

[SYC04] Richard T. Snodgrass, Shilong Stanley Yao, and Christian Collberg. Tamper Detection in Audit Logs. In *Proceedings of the Thirtieth International Conference on Very Large Data Bases (VLDB)*, pages 504–515. VLDB Endowment, 2004.

[TCG07] Trusted Computing Group. TCG Specification Architecture Overview. Specification Revision 1.4, 2007.

[TCN+14] Danan Thilakanathan, Shiping Chen, Surya Nepal, Rafael Calvo, and Leila Alem. A platform for secure monitoring and sharing of generic health data in the Cloud. *Future Generation Computer Systems*, 35:102–113, 2014.

[TD17] Alin Tomescu and Srinivas Devadas. Catena: Efficient Non-equivocation via Bitcoin. In *Proceedings of the 2017 IEEE Symposium on Security and Privacy (SP)*, pages 393–409. IEEE, 2017.

[TGG+12] Amin Tootoonchian, Sergey Gorbunov, Yashar Ganjali, Martin Casado, and Rob Sherwood. On Controller Performance in Software-Defined Networks. In *Proceedings of the 2nd USENIX Workshop on Hot Topics in Management of Internet, Cloud, and Enterprise Networks and Services (Hot-ICE)*. USENIX, 2012.

[TJA10] Hassan Takabi, James B.D. Joshi, and Gail-Joon Ahn. Security and Privacy Challenges in Cloud Computing Environments. *IEEE Security & Privacy*, 8(6):24–31, 2010.

[TLL16] Cory Thoma, Adam J. Lee, and Alexandros Labrinidis. PolyStream: Cryptographically Enforced Access Controls for Outsourced Data Stream Processing. In *Proceedings of the 21st ACM on Symposium on Access Control Models and Technologies (SACMAT)*, pages 227–238. ACM, 2016.

[TM11] Romuald Thion and Daniel Le Metayer. FLAVOR: A Formal Language for a Posteriori Verification of Legal Rules. In *Proceedings of the 2011 IEEE International Symposium on Policies for Distributed Systems and Networks (POLICY)*, pages 1–8. IEEE, 2011.

[TPPG13] Marianthi Theoharidou, Nick Papanikolaou, Siani Pearson, and Dimitris Gritzalis. Privacy Risk, Security, Accountability in the Cloud. In *Proceedings of the 2013 IEEE 5th International Conference on Cloud Computing Technology and Science (CloudCom)*, pages 177–184. IEEE, 2013.

[Twi15] Twissandra. `https://github.com/twissandra/twissandra/`, 2015. [Online, accessed 2018-07-01].

[Udo01] Godwin J. Udo. Privacy and security concerns as major barriers for e-commerce: a survey study. *Information Management & Computer Security*, 9(4):165–174, 2001.

[UN48] United Nations General Assembly. The Universal Declaration of Human Rights. General Assembly Resolution 217 A, 1948.

[VEM+15] Anjo Vahldiek-Oberwagner, Eslam Elnikety, Aastha Mehta, Deepak Garg, Peter Druschel, Rodrigo Rodrigues, Johannes Gehrke, and Ansley Post. Guardat: Enforcing data policies at the storage layer. In *Proceedings of the Tenth European Conference on Computer Systems (EuroSys)*, pages 13:1–13:16. ACM, 2015.

[VMC02] John Viega, Matt Messier, and Pravir Chandra. *Network Security with OpenSSL: Cryptography for Secure Communications*. O'Reilly, 2002.

[VR14] Luis M. Vaquero and Luis Rodero-Merino. Finding your Way in the Fog: Towards a Comprehensive Definition of Fog Computing. *ACM SIGCOMM Computer Communication Review*, 44(5):27–32, 2014.

[VSF+12] Narseo Vallina-Rodriguez, Jay Shah, Alessandro Finamore, Yan Grunenberger, Konstantina Papagiannaki, Hamed Haddadi, and Jon Crowcroft. Breaking for Commercials: Characterizing Mobile Advertising. In *Proceedings of the 2012 Internet Measurement Conference (IMC)*, pages 343–356. ACM, 2012.

[VSR+16] Narseo Vallina-Rodriguez, Srikanth Sundaresan, Abbas Razaghpanah, Rishab Nithyanand, Mark Allman, Christian Kreibich, and Phillipa Gill. Tracking the Trackers: Towards Understanding the Mobile Advertising and Tracking Ecosystem. arXiv preprint arXiv:1609.07190 [cs.CY], 2016.

[Wal96] John Walker. HotBits: Genuine random numbers, generated by radioactive decay, 1996.

[Wal16] Matthew Wall. Can we trust cloud providers to keep our data safe? `http://www.bbc.com/news/business-36151754`, 2016. [Online, accessed 2018-07-01].

[WB90] Samuel D. Warren and Louis D. Brandeis. The right to privacy. *Harvard Law Review*, 4(5):193–220, 1890.

[WBDS04] Brent R. Waters, Dirk Balfanz, Glenn Durfee, and Diana K. Smetters. Building an Encrypted and Searchable Audit Log. In *Proceedings of the Network and Distributed System Security Symposium (NDSS)*. The Internet Society, 2004.

[WBMM06] Sage A. Weil, Scott A. Brandt, Ethan L. Miller, and Carlos Maltzahn. CRUSH: Controlled, Scalable, Decentralized Placement of Replicated Data. In *Proceedings of the 2006 ACM/IEEE Conference on Supercomputing (SC)*. ACM, 2006.

[WC16] Edward Wang and Richard Chow. What Can I Do Here? IoT Service Discovery in Smart Cities. In *Proceedings of the 2016 IEEE International Conference on Pervasive Computing and Communication Workshops (PerCom Workshops)*, pages 1–6. IEEE, 2016.

[WDB14] Dale Willis, Arkodeb Dasgupta, and Suman Banerjee. ParaDrop: A Multi-tenant Platform to Dynamically Install Third Party Services on Wireless Gateways. In *Proceedings of the 9th ACM Workshop on Mobility in the Evolving Internet Architecture (MobiArch)*, pages 43–48. ACM, 2014.

[WDL13] Kevin Wiesner, Florian Dorfmeister, and Claudia Linnhoff-Popien. Privacy-Preserving Calibration for Participatory Sensing. In *Proceedings of the 10th International Conference on Mobile and Ubiquitous Systems: Computing, Networking, and Services (MobiQuitous)*, pages 276–288. Springer, 2013.

[Wes67] Alan Westin. *Privacy and Freedom*. Atheneum, 1967.

[Wes03] Alan F. Westin. Social and Political Dimensions of Privacy. *Journal of Social Issues*, 59(2):431–453, 2003.

[WGG10] Klaus Wehrle, Mesut Günes, and James Gross. *Modeling and Tools for Network Simulation*. Springer, 2010.

[WGNF12] Xuetao Wei, Lorenzo Gomez, Iulian Neamtiu, and Michalis Faloutsos. ProfileDroid: Multi-layer Profiling of Android Applications. In *Proceedings of the 18th Annual International Conference on Mobile Computing and Networking (Mobicom)*, pages 137–148. ACM, 2012.

[WGR05] Klaus Wehrle, Stefan Götz, and Simon Rieche. Distributed Hash Tables. In Ralf Steinmetz and Klaus Wehrle, editors, *Peer-to-Peer Systems and Applications*, chapter 7, pages 79–93. Springer, 2005.

[Whi71] James E. White. Network Specifications for Remote Job Entry and Remote Job Output Retrieval at UCSB. Request for Comments 105, Internet Engineering Task Force, 1971.

[Wik16] WikiLeaks. `http://wikileaks.org/`, 2016. [Online, accessed 2016-10-13].

[WLFW06] Raymond Chi-Wing Wong, Jiuyong Li, Ada Wai-Chee Fu, and Ke Wang. (α, K)-anonymity: An Enhanced K-anonymity Model for Privacy Preserving Data Publishing. In *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 754–759. ACM, 2006.

[WMF13] Tobias Wüchner, Steffen Müller, and Robin Fischer. Compliance-Preserving Cloud Storage Federation Based on Data-Driven Usage Control. In *Proceedings of the 2013 IEEE 5th International Conference on Cloud Computing Technology and Science (CloudCom)*, pages 285–288. IEEE, 2013.

[Wol14] Benedikt Wolters. Distributed Authorization Management for Secure Sensor Data in the Cloud. Bachelor's thesis, RWTH Aachen University, March 2014.

[WS14] Melanie Willett and Rossouw Von Solms. Cloud-based Email Adoption at Higher Education Institutions in South Africa. *Journal of International Technology and Information Management*, 23(2):17–29, 2014.

[WSA+12] Gaven J. Watson, Reihaneh Safavi-Naini, Mohsen Alimomeni, Michael E. Locasto, and Shivaramakrishnan Narayan. LoSt: Location Based Storage. In *Proceedings of the 2012 ACM Cloud Computing Security Workshop (CCSW)*, pages 59–70. ACM, 2012.

[WSC17] Matthias Wachs, Quirin Scheitle, and Georg Carle. Push Away Your Privacy: Precise User Tracking Based on TLS Client Certificate Authentication. In *Proceedings of the 2017 Network Traffic Measurement and Analysis Conference (TMA)*. IEEE, 2017.

[WWRL10] Cong Wang, Qian Wang, Kui Ren, and Wenjing Lou. Privacy-Preserving Public Auditing for Data Storage Security in Cloud Computing. In *Proceedings of the 29th IEEE International Conference on Computer Communications (INFOCOM)*, pages 1–9. IEEE, 2010.

[XEG+11] Qiang Xu, Jeffrey Erman, Alexandre Gerber, Zhuoqing Mao, Jeffrey Pang, and Shobha Venkataraman. Identifying Diverse Usage Behaviors of Smartphone Apps. In *Proceedings of the 2011 ACM SIGCOMM Conference on Internet Measurement Conference (IMC)*, pages 329–344. ACM, 2011.

[XYA+07] Yinglian Xie, Fang Yu, Kannan Achan, Eliot Gillum, Moises Goldszmidt, and Ted Wobber. How Dynamic Are IP Addresses? In *Proceedings of the 2007 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM)*, pages 301–312. ACM, 2007.

[YDAJ15] Kenji Yoshigoe, Wei Dai, Melissa Abramson, and Alexander Jacobs. Overcoming Invasion of Privacy in Smart Home Environment with Synthetic Packet Injection. In *Proceedings of the 2015 TRON Symposium (TRONSHOW)*, pages 1–7. IEEE, 2015.

[YL11] Jaewon Yang and Jure Leskovec. Patterns of Temporal Variation in Online Media. In *Proceedings of the Fourth ACM International Conference on Web Search and Data Mining (WSDM)*, pages 177–186. ACM, 2011.

[YN09] Attila Altay Yavuz and Peng Ning. BAF: An Efficient Publicly Verifiable Secure Audit Logging Scheme for Distributed Systems. In *Proceedings of the 2009 Annual Computer Security Applications Conference (ACSAC)*, pages 219–228. IEEE, 2009.

[YPLL14] Hui-Shyong Yeo, Xiao-Shen Phang, Hoon-Jae Lee, and Hyotaek Lim. Leveraging client-side storage techniques for enhanced use of multiple consumer cloud storage services on resource-constrained mobile devices. *Journal of Network and Computer Applications*, 43:142–156, 2014.

[YWRL10] Shucheng Yu, Cong Wang, Kui Ren, and Wenjing Lou. Achieving Secure, Scalable, and Fine-grained Data Access Control in Cloud Computing. In *Proceedings of the 29th IEEE International Conference on Computer Communications (INFOCOM)*, pages 534–542. IEEE, 2010.

[YYZ+13] Zhemin Yang, Min Yang, Yuan Zhang, Guofei Gu, Peng Ning, and X. Sean Wang. AppIntent: Analyzing Sensitive Data Transmission in Android for Privacy Leakage Detection. In *Proceedings of the 2013 ACM SIGSAC Conference on Computer and Communications Security (CCS)*, pages 1043–1054. ACM, 2013.

[ZB11] Shehnila Zardari and Rami Bahsoon. Cloud Adoption: A Goal-oriented Requirements Engineering Approach. In *Proceedings of the 2nd International Workshop on Software Engineering for Cloud Computing (SECLOUD)*, pages 29–35. ACM, 2011.

[ZDH13] Shams Zawoad, Amit Kumar Dutta, and Ragib Hasan. SecLaaS: Secure Logging-as-a-service for Cloud Forensics. In *Proceedings of the 8th ACM SIGSAC Symposium on Information, Computer and Communications Security (ASIACCS)*, pages 219–230. ACM, 2013.

[ZGS03] Shuheng Zhou, Gregory R Ganger, and Peter Alfons Steenkiste. Location-based Node IDs: Enabling Explicit Locality in DHTs. Technical Report CMU-CS-03-171, School of Computer Science, Carnegie Mellon University, 2003.

[ZGW14] Jan Henrik Ziegeldorf, Oscar Garcia Morchon, and Klaus Wehrle. Privacy in the Internet of Things: Threats and Challenges. *Security and Communication Networks*, 7(12):2728–2742, 2014.

[ZHHW15] Jan Henrik Ziegeldorf, Martin Henze, René Hummen, and Klaus Wehrle. Comparison-based Privacy: Nudging Privacy in Social Media (Position Paper). In *Proceedings of the 10th International Workshop on Data Privacy Management (DPM)*, pages 226–234. Springer, 2015.

[Zig12] ZigBee Alliance. ZigBee Specification. ZigBee Document 053474r20, 2012.

[Zig13] ZigBee Alliance. Smart Energy Profile 2 Application Protocol Standard. ZigBee Public Document 13-0200-00, 2013.

[Zim80] Hubert Zimmermann. OSI Reference Model – The ISO Model of Architecture for Open Systems Interconnection. *IEEE Transactions on Communications*, 28(4):425–432, 1980.

[ZMHW15] Jan Henrik Ziegeldorf, Jan Metzke, Martin Henze, and Klaus Wehrle. Choose Wisely: A Comparison of Secure Two-Party Computation Frameworks. In *Proceedings of the 2015 IEEE Security and Privacy Workshops (SPW)*, pages 198–205. IEEE, 2015.

[ZNP15] Guy Zyskind, Oz Nathan, and Alex 'Sandy' Pentland. Decentralizing Privacy: Using Blockchain to Protect Personal Data. In *Proceedings of the 2015 IEEE Security and Privacy Workshops (SPW)*, pages 180–184. IEEE, 2015.

[ZPH+17] Jan Henrik Ziegeldorf, Jan Pennekamp, David Hellmanns, Felix Schwinger, Ike Kunze, Martin Henze, Jens Hiller, Roman Matzutt, and Klaus Wehrle. BLOOM: BLoom filter based oblivious outsourced matchings. *BMC Medical Genomics*, 10(Suppl 2):29–42, 2017.

[ZSW13] Frances Zhang, Fuming Shih, and Daniel Weitzner. No Surprises: Measuring Intrusiveness of Smartphone Applications by Detecting Objective Context Deviations. In *Proceedings of the 12th ACM Workshop on Privacy in the Electronic Society (WPES)*, pages 291–296. ACM, 2013.

[ZVHW14] Jan Henrik Ziegeldorf, Nicolai Viol, Martin Henze, and Klaus Wehrle. POSTER: Privacy-preserving Indoor Localization. In *Poster Session of the 7th ACM Conference on Security and Privacy in Wireless and Mobile Networks (WiSec)*, 2014.

[ZZ11] Xiao Ming Zhang and Ning Zhang. An Open, Secure and Flexible Platform Based on Internet of Things and Cloud Computing for Ambient Aiding Living and Telemedicine. In *Proceedings of the 2011 International Conference on Computer and Management (CAMAN)*, pages 1–4. IEEE, 2011.