



**KTH Computer Science
and Communication**

Security and Privacy of Sensitive Data in Cloud Computing

ALI GHOLAMI

Doctoral Thesis
Stockholm, Sweden 2016

TRITA-CSC-A-2016:11 Department of Computational Science and Technology
ISSN 1653-5723 KTH School of Computer Science and Communication
ISRN KTH/CSC/A--16/11--SE SE-100 44 Stockholm
ISBN 978-91-7595-941-2 SWEDEN

Akademisk avhandling som med tillstånd av Kungl Tekniska högskolan framläggas till offentlig granskning för avläggande av teknologie doktorsexamen i datalogi onsdagen den 8 juni 2016 klockan 10.00 i Kollegiesalen, Administrationsbyggnaden, Kungl Tekniska högskolan, Valhallavägen 79, Stockholm.

© Ali Gholami, April 2016

Tryck: Universitetsservice US AB

Abstract

Cloud computing offers the prospect of on-demand, elastic computing, provided as a utility service, and it is revolutionizing many domains of computing. Compared with earlier methods of processing data, cloud computing environments provide significant benefits, such as the availability of automated tools to assemble, connect, configure and reconfigure virtualized resources on demand. These make it much easier to meet organizational goals as organizations can easily deploy cloud services. However, the shift in paradigm that accompanies the adoption of cloud computing is increasingly giving rise to security and privacy considerations relating to facets of cloud computing such as multi-tenancy, trust, loss of control and accountability. Consequently, cloud platforms that handle sensitive information are required to deploy technical measures and organizational safeguards to avoid data protection breakdowns that might result in enormous and costly damages.

Sensitive information in the context of cloud computing encompasses data from a wide range of different areas and domains. Data concerning health is a typical example of the type of sensitive information handled in cloud computing environments, and it is obvious that most individuals will want information related to their health to be secure. Hence, with the growth of cloud computing in recent times, privacy and data protection requirements have been evolving to protect individuals against surveillance and data disclosure. Some examples of such protective legislation are the EU Data Protection Directive (DPD) and the US Health Insurance Portability and Accountability Act (HIPAA), both of which demand privacy preservation for handling personally identifiable information.

There have been great efforts to employ a wide range of mechanisms to enhance the privacy of data and to make cloud platforms more secure. Techniques that have been used include: encryption, trusted platform module, secure multi-party computing, homomorphic encryption, anonymization, container and sandboxing technologies.

However, it is still an open problem about how to correctly build usable privacy-preserving cloud systems to handle sensitive data securely due to two research challenges. First, existing privacy and data protection legislation demand strong security, transparency and audibility of data usage. Second, lack of familiarity with a broad range of emerging or existing security solutions to build efficient cloud systems.

This dissertation focuses on the design and development of several systems and methodologies for handling sensitive data appropriately in cloud computing environments. The key idea behind the proposed solutions is enforcing the privacy requirements mandated by existing legislation that aims to protect the privacy of individuals in cloud-computing platforms.

We begin with an overview of the main concepts from cloud computing, followed by identifying the problems that need to be solved for secure data management in cloud environments. It then continues with a description of

background material in addition to reviewing existing security and privacy solutions that are being used in the area of cloud computing.

Our first main contribution is a new method for modeling threats to privacy in cloud environments which can be used to identify privacy requirements in accordance with data protection legislation. This method is then used to propose a framework that meets the privacy requirements for handling data in the area of genomics. That is, health data concerning the genome (DNA) of individuals. Our second contribution is a system for preserving privacy when publishing sample availability data. This system is noteworthy because it is capable of cross-linking over multiple datasets. The thesis continues by proposing a system called ScaBIA for privacy-preserving brain image analysis in the cloud. The final section of the dissertation describes a new approach for quantifying and minimizing the risk of operating system kernel exploitation, in addition to the development of a system call interposition reference monitor for Lind - a dual sandbox.

Sammanfattning

“Cloud computing”, eller “molntjänster” som blivit den vanligaste svenska översättningen, har stor potential. Molntjänster kan tillhandahålla exakt den datakraft som efterfrågas, nästan oavsett hur stor den är; dvs. molntjänster möjliggör vad som brukar kallas för “elastic computing”. Effekterna av molntjänster är revolutionerande inom många områden av datoranvändning. Jämfört med tidigare metoder för databehandling ger molntjänster många fördelar; exempelvis tillgänglighet av automatiserade verktyg för att montera, ansluta, konfigurera och re-konfigurera virtuella resurser “allt efter behov” (“on-demand”). Molntjänster gör det med andra ord mycket lättare för organisationer att uppfylla sina målsättningar. Men det paradigmskifte, som införandet av molntjänster innebär, skapar även säkerhetsproblem och förutsätter noggranna integritetsbedömningar. Hur bevaras det ömsesidiga förtroendet, hur hanteras ansvarsutkrävandet, vid minskade kontrollmöjligheter till följd av delad information? Följaktligen behövs molnplattformar som är så konstruerade att de kan hantera känslig information. Det krävs tekniska och organisatoriska hinder för att minimera risken för dataintrång, dataintrång som kan resultera i enormt kostsamma skador såväl ekonomiskt som policymässigt. Molntjänster kan innehålla känslig information från många olika områden och domäner. Hälso-data är ett typiskt exempel på sådan information. Det är uppenbart att de flesta människor vill att data relaterade till deras hälsa ska vara skyddad. Så den ökade användningen av molntjänster på senare år har medfört att kraven på integritets- och dataskydd har skärpts för att skydda individer mot övervakning och dataintrång. Exempel på skyddande lagstiftning är “EU Data Protection Directive” (DPD) och “US Health Insurance Portability and Accountability Act” (HIPAA), vilka båda kräver skydd av privatlivet och bevarandet av integritet vid hantering av information som kan identifiera individer. Det har gjorts stora insatser för att utveckla fler mekanismer för att öka dataintegriteten och därmed göra molntjänsterna säkrare. Exempel på detta är; kryptering, “trusted platform modules”, säker “multi-party computing”, homomorfisk kryptering, anonymisering, container- och “sandlåde”-tekniker.

Men hur man korrekt ska skapa användbara, integritetsbevarande molntjänster för helt säker behandling av känsliga data är fortfarande i väsentliga avseenden ett olöst problem på grund av två stora forskningsutmaningar. För det första: Existerande integritets- och dataskydds-lagar kräver transparens och noggrann granskning av dataanvändningen. För det andra: Bristande kännedom om en rad kommande och redan existerande säkerhetslösningar för att skapa effektiva molntjänster.

Denna avhandling fokuserar på utformning och utveckling av system och metoder för att hantera känsliga data i molntjänster på lämpligaste sätt. Målet med de framlagda lösningarna är att svara de integritetskrav som ställs i redan gällande lagstiftning, som har som uttalad målsättning att skydda individers integritet vid användning av molntjänster.

Vi börjar med att ge en överblick av de viktigaste begreppen i molntjänster, för att därefter identifiera problem som behöver lösas för säker databehandling vid användning av molntjänster. Avhandlingen fortsätter sedan med

en beskrivning av bakgrundsmaterial och en sammanfattning av befintliga säkerhets- och integritets-lösningar inom molntjänster.

Vårt främsta bidrag är en ny metod för att simulera integritetshot vid användning av molntjänster, en metod som kan användas till att identifiera de integritetskrav som överensstämmer med gällande dataskyddslagar. Vår metod används sedan för att föreslå ett ramverk som möter de integritetskrav som ställs för att hantera data inom området "genomik". Genomik handlar i korthet om hälsodata avseende arvsmassan (DNA) hos enskilda individer. Vårt andra större bidrag är ett system för att bevara integriteten vid publicering av biologiska provdata. Systemet har fördelen att kunna sammankoppla flera olika uppsättningar med data. Avhandlingen fortsätter med att föreslå och beskriva ett system kallat ScaBIA, ett integritetsbevarande system för hjärnbildsanalyser processade via molntjänster. Avhandlingens avslutande kapitel beskriver ett nytt sätt för kvantifiering och minimering av risk vid "kernel exploitation" ("utnyttjande av kärnan"). Denna nya ansats är även ett bidrag till utvecklingen av ett nytt system för (Call interposition reference monitor for Lind - the dual layer sandbox).

Acknowledgements

I would like to express my sincere gratitude to Prof. Erwin Laure, for supervising the thesis, helpful criticism and advice. Indeed, his incredible knowledge of computer systems and scientific approach to face research problems was always inspiring. I would also like to thank my co-advisor Prof. Seif Haridi, for letting me work with his excellent research group.

During my doctoral studies, I was fortunate to work with several brilliant people who I always admire. First, a special mention goes to Dr. Jim Dowling for his technical excellence and patience. Second, I would greatly appreciate Dr. Justin Cappos for providing me an internship at NYU and deepening my knowledge of cloud security. Third, I would like to thank all my co-authors and those who helped me to accomplish this thesis. Most notably, Prof. Jane Reichel for her invaluable comments, Prof. Jan-Eric Litton for his support, Prof. Ulf Leser for his feedback on the usability aspects of my research, Dr. Sonja Buchegger for her suggestions, Dr. Åke Edlund for always being helpful, Gert Svensson for his understanding and support, Gilbert Netzer for always providing good answers to my questions, Michael Schliephake for his helpfull suggestions, Genet Edmonson for improving my technical writing, and Laeeq Ahdmad for proof-reading of the thesis.

I would like to extend my gratitude to Prof. Schahram Dustdar for being my opponent. I am also grateful to Prof. Cecilia Magnusson Sjöberg, Dr. Rose-Mharie Åhlfeldt, Dr. Javid Taheri, Prof. Jeanette Hellgren-Kotaleski and Dr. Lars Arvestad to be the committee members of the thesis.

Financial support from the Swedish e-Science Research Center (SeRC), National Science Foundation (NSF) and the European FP7 framework is acknowledged.

Contents

Contents	viii
List of Figures	xiii
List of Tables	xv
I Prologue	1
1 Introduction	3
1.1 Motivation	3
1.2 Reference Platforms	6
1.2.1 Scalable Secure Storage BioBankCloud	6
1.2.2 VENUS-C	10
1.3 Research Questions and Contributions	13
1.4 Research Method	14
1.5 List of Scientific Papers	15
1.6 Thesis Outline	17
2 Background	19
2.1 Big Data Infrastructures	19
2.2 Cloud Computing	20
2.2.1 Concepts in Cloud Computing	22
2.2.2 Virtualization	24
2.2.3 Container Technology	26
2.3 Security Techniques to Ensure Privacy	31
2.3.1 The EU DPD Key Concepts	31
2.3.2 Authentication	32
2.3.3 Data Anonymization Techniques	34
2.3.4 Secret Sharing	37
2.4 Summary	39
3 Related Work	41

3.1	Identification of Research	41
3.2	Cloud Security	42
3.2.1	Authentication and Authorization	42
3.2.2	Identity and Access Management	44
3.2.3	Confidentiality, Integrity and Availability (CIA)	45
3.2.4	System Call Interposition:	49
3.2.5	Security Monitoring and Incident Response	50
3.2.6	Security Policy Management	50
3.3	Data Security and Privacy	51
3.3.1	Big Data Infrastructures and Programming Models	52
3.3.2	Privacy-Preserving Solutions in the Cloud	54
3.3.3	Privacy-Preservation Database Federation	56
3.4	Summary	57
 II Privacy by Design for Cloud Computing		59
4	Privacy Threat Modeling Methodology for Cloud Computing Environments	61
4.1	Introduction	61
4.2	Characteristics of a Privacy Threat Modeling Methodology for Cloud Computing	62
4.2.1	Privacy Legislation Support	62
4.2.2	Technical Deployment and Service Models	62
4.2.3	Customer Needs	62
4.2.4	Usability	63
4.2.5	Traceability	63
4.3	Methodology Steps and Their Products	63
4.3.1	Privacy Regulatory Compliance	64
4.3.2	Cloud Environment Specification	65
4.3.3	Privacy Threat Identification	66
4.3.4	Risk Evaluation	66
4.3.5	Threat Mitigation	67
4.4	Summary	67
5	Case Study: BioBankCloud Privacy Threat Modeling	69
5.1	Introduction	69
5.2	Scenario	70
5.3	Privacy Requirements	71
5.4	Cloud Environment Specification	74
5.5	Privacy Threat Identification	77
5.6	Risk Evaluation	80
5.7	Threat Mitigation	83
5.8	Summary	85

6	Design and Implementation of the Secure BioBankCloud	87
6.1	Introduction	87
6.2	Security Architecture	88
6.2.1	Comparison of Existing Solutions	88
6.2.2	Proposed Selection of Components	97
6.3	Design	98
6.3.1	Assumptions	98
6.3.2	Identity and Access Management	98
6.3.3	Authentication	99
6.3.4	Authorization	102
6.3.5	Auditing	103
6.4	Implementation	106
6.4.1	The Middleware and Libraries	106
6.4.2	Identity and Access Management	106
6.4.3	Custom Authentication Realm	110
6.4.4	Authorization	112
6.4.5	Privacy and Ethical Settings	113
6.4.6	Auditing	115
6.5	Verification and Validation	117
6.6	Discussion	118
6.7	Summary	120
 III Trustworthy Privacy-Preserving Cloud Models		121
7	Privacy-Preserving Data Publishing for Sample Availability Data	123
7.1	Introduction	123
7.2	Privacy-Preservation Mechanisms	124
7.3	Obscuring the Key Attributes	125
7.3.1	Hashing and Encryption	125
7.4	Threat Assumptions	126
7.4.1	Inference Attacks	126
7.4.2	Malicious Sample Publication	126
7.4.3	Audit and Control	127
7.4.4	Server Private Key Compromised	127
7.4.5	Ethical Constraints	127
7.4.6	Static Passwords	127
7.4.7	Query Reply Limitation	127
7.5	Design and Implementation	128
7.5.1	Scenario	128
7.5.2	Integration Service	130
7.5.3	Secure Data Management	131
7.5.4	Data Pseudonymization and Anonymization	132
7.5.5	Re-identification Risk	133

7.5.6	Auditing Process	134
7.6	Summary	135
8	Privacy-Preserving Brain Image Analysis in the Cloud	137
8.1	Introduction	137
8.2	Statistical Parametric Mapping (SPM)	138
8.3	Design	139
8.3.1	Security Management (SM)	140
8.3.2	Data Management (DM)	140
8.3.3	Job Management (JM)	140
8.3.4	Application Management (AM)	141
8.4	Security and Privacy	141
8.4.1	Authentication	142
8.4.2	Authorization	142
8.5	Implementation	143
8.5.1	Anonymization	143
8.5.2	Secure Deployment of the Generic Worker	143
8.5.3	Building the Application	144
8.5.4	Job Submission	144
8.5.5	Data Management	145
8.6	Summary	146
IV	Secure Multi-Tenancy in the Cloud	147
9	Quantifying and Minimizing the Risk of Kernel Exploitation	149
9.1	Introduction	149
9.2	Lind Dual-Layer Sandbox	150
9.2.1	Native Client (NaCl)	150
9.2.2	Seattle's Repty	152
9.3	Quantitative Evaluation	153
9.3.1	Hypothesis	153
9.3.2	Data Sources and Experiments	153
9.3.3	Kernel-Level Data Collection	156
9.3.4	Data Transformation	157
9.3.5	Kernel Traces Analysis and Evaluation	157
9.4	Summary	157
10	Lind Reference Monitor	159
10.1	Introduction	159
10.2	System Call Interposition Model	159
10.2.1	Policy Configurations	160
10.2.2	System Call Filtering	160
10.3	Implementation	161

10.4 Validation	162
10.5 Summary	162
V Epilogue	163
11 Discussion	165
11.1 Discussion on Formulating the Cloud Privacy Requirements	165
11.1.1 Cloud Privacy Threat Modeling	166
11.2 Discussion on Building Privacy-Preserving Cloud Solutions	166
11.3 Discussion on Quantifying and Minimizing the Risk of Kernel Exploits	167
12 Future Work	169
12.1 Privacy by Design for Cloud Computing	169
12.1.1 Applications of the CPTM in Other Domains	169
12.1.2 Emerging Data Protection Laws	170
12.1.3 Security and Usability of the BioBankCloud	170
12.2 Trustworthy Privacy-Preserving Cloud Models	170
12.3 Secure Multi-Tenancy in the Cloud	171
Bibliography	173
Appendices	193
A BioBankCloud	193
A.1 Identity and Access Management	193
A.2 Auditing Users Actions	198
B eCPC Toolkit	203
B.1 k-anonymity	203
B.2 l-diversity	204
B.3 Reidentification Risk	205
C Lind Dual Sandbox	207
C.1 Porting Applications in NaCl and RePy	207
C.2 Lind's Parser for Gcov	209
D Lind Reference Monitor	213
D.1 Policy Definition in Lind	213
D.2 System Call Filtering in Lind	216
D.3 System Call Forwarding in Lind	251
List of Abbreviations	255

List of Figures

1.1	Scalable, Secure Storage Biobank (BioBankCloud) Architecture	8
1.2	Study1 has John and Mary as users and includes DataSet1, while Study2 has only John as as a user and includes DataSet1, DataSet2, and DataSet3.	8
1.3	HopsFS and HopsYARN architectures.	9
1.4	The software stack of the scientific workflow management system SAAS-FEE, which comprises the functional workflow language Cuneiform as well as the Hi-WAY workflow scheduler for Hadoop. Cuneiform can execute foreign code written in languages like Python, Bash, and R. Besides Cuneiform, Hi-WAY can also interpret the workflow languages of the SWfMSs Pegasus and Galaxy. SAASFEE can be run both on Hadoop Optimized File System (HOPS) as well as Apache Hadoop. SAASFEE and HOPS can be interfaced and configured via the web interface provided by the Lab Information Management System (LIMS).	11
1.5	VENUS-C architecture	11
1.6	Simplified internal GW architecture	12
2.1	Big data ecosystem reference architecture (image courtesy of NIST [1])	19
2.2	Cloud computing reference architecture (image courtesy of NIST [2])	24
2.3	Comparison of Type-I and Type II virtualization architectures for Xen and KVM	26
2.4	Docker architecture (image courtesy of [3])	30
2.5	Linking to re-identify data (image courtesy of [4])	34
4.1	Privacy threat modeling in requirements engineering and design of a SDLC	64
4.2	The Cloud Privacy Threat Modeling (CPTM) methodology steps	64
5.1	BioBankCloud Physical architectures	76
5.2	Logical architecture	76
6.1	Security architecture of the BioBankCloud including various security modules	97
6.2	Identity lifecycle in the BioBankCloud	98

6.3	Custom authentication realm to support authentication for users with and without mobile devices.	100
6.4	Scanning the QRC using the Autneticator App in smartphones	101
6.5	Account registration in the AngularJS frontend	101
6.6	Yubikey accounts provisioning	101
6.7	BioBankCloud authorization system to enforce permissions to access study data.	103
6.8	Audit system	104
6.9	Account recovery options to be selected for resetting the users accounts .	109
6.10	Accounts management functinoalities to add/remove roles or changing the users status	109
6.11	Activation of new incoming user accounts requests	109
6.12	User's profile with the functionalities to change information, security credentials or terminate the account	110
6.13	Steps to change password, security question or terminate the account in the user' profile	110
6.14	Adding/removing roles or blocking/activating/deactivating user accounts	110
6.15	User authentication login page	111
6.16	Controlling privacy settings including uploading consent forms or updating the retention period of data by the data owner	114
6.17	Reviewing overall project status	114
6.18	Reviweing the new consents to be approved or rejected	114
6.19	Expired data sets to be removed by the administrator	115
6.20	Audit panel accessible for administrator and auditor roles	116
6.21	Role access and entitlement events audit panel	116
6.22	Auditing the login events of users	116
6.23	Auditing of account management activities	116
6.24	Auditing project information including based on several parameters such as the study name, date of access and username	117
7.1	PID pseudonymization through a two-level hashing mechanism to provide the functionality for joint queries over different data sources.	126
7.2	The eCPC toolkit design based on the privacy-preserving data publishing methods to upload the pseudonymized data to an external trusted third-party service.	129
7.3	Overview of the e-Science for Cancer Prevention and Control (eCPC) integration server that is protected with firewall to filter the ingoing/outgoing traffic.	130
7.4	Public key encryption of the large sensitive data sets using the TTP's private key.	132
7.5	Anonymization of the sensitive data using sdcMicro library.	133
7.6	Individual risk estimation of the pseudonymized data using sdcMicro library.	134

8.1 Resulting activation map of an experiment 138

8.2 A series of stages to do an fMRI data analysis over N subjects (S_1, S_2, \dots, S_N) each subject i containing n images ($IMG_{i,1}, IMG_{i,2}, \dots, IMG_{i,n}$) 139

8.3 Architectural view of the ScaBIA in the Cloud 139

8.4 Job execution on a GW instance 141

8.5 Installing the application requirements 141

8.6 Process of creating SPM scripts and making them compatible with GW 145

9.1 Architecture of Lind including various components such as NaCl, NaCl' glibc, and Rely Sandbox. User level applications will issue system calls that are dispatched through the Rely OS connector that bridges the Lind system to the OS Kernel. 151

9.2 Various activities performed to capture and analyze the kernel traces generated by legacy applications, system fuzzers, LTP, and CVE bug reports. The traces are collected using gcov and a Python-based program that transforms the gcov data to macrodata-level information of each traversed path for final data analysis. 153

9.3 Percentage of different kernel areas that were reached during LTP and Trinity system call fuzzing experiments to measure the reachable kernel surface 157

10.1 Reference monitor architecture 160

List of Tables

2.1	Evolution of Big Data from batch to real-time analytics processing [5]	21
2.2	Cloud computing characteristics [6]	22
2.3	Raw private patient dataset without anonymization	35
2.4	A sample patient dataset with k -anonymity, where $k=4$	36
2.5	k -anonymity description of attributes to prevent record linkage through Quasi Identifier (QID)	36
2.6	A sample patient dataset with ℓ -diversity, where $\ell=2$	37
3.1	Security and privacy factors of cloud providers [7]	42
4.1	Prioritization of the identified threats, L (Low), M (Moderate), H(High)	67
5.1	Correlating the domain actors to the cloud actors	75
5.2	Correlating the BioBankCloud actors with the DPD roles	77
5.3	Risk evaluation matrix for the identified threats. I indicates the likelihood of threat and E indicates the effect of exploiting the threat on the whole BioBankCloud.	83
6.1	Access control table to define the permissions for each role in the platform in regard to using the BioBankCloud services. For example, a researcher can <i>create</i> (C) a new study and will be assigned the data provider role afterwards. Then, as a data provider, the user will be able to <i>read</i> (R), <i>update</i> (U) and add new members or <i>delete</i> (D) or <i>execute</i> (X) the study.	104
6.2	Implementation of the BioBankCloud roles	112
8.1	Microsoft Azure basic tier general purpose compute	143
9.1	Repy sandbox kernel capabilities that supports NaCl functions, such as networking, file I/O operations and threading.	152
9.2	Exploitable CVEs that we triggered under VirtualBox, VMWare Workstation, Docker, LXC, QEMU, KVM and Graphene virtualization systems	156

List of Algorithms

1	The HMAC-based One-time Password (HOTP) algorithm	33
2	The Time-based One-time Password (TOTP) algorithm	33

Part I
Prologue

Chapter 1

Introduction

1.1 Motivation

Many organizations that handle sensitive information are considering using cloud computing as it provides resources that can be scaled easily, along with significant economic benefits in the form of reduced operational costs. However, it can be complicated to correctly handle sensitive data in cloud computing environments due to the range of privacy legislation and regulations that exist. Some examples of such legislation are the European Union (EU) Data Protection Directive (DPD) [8] and the US Health Insurance Portability and Accountability Act (HIPAA) [9], both demand privacy-preservation for handling personally identifiable information. This thesis discusses the challenges faced by such organizations and describes how cloud computing can be used to provide innovative solutions that ensure the safety of sensitive information.

The main focus of this thesis is on security and privacy issues concerning data produced by medical research, which requires particularly strict privacy-preserving solutions [10]. For example, a researcher that seeks to understand the human body and gain insights into disease processes by utilizing big data analytics and cloud computing technologies. However, when using data in the cloud, it is necessary to take into account the ethical and regulatory considerations that relate to data ownership. Such data must be processed transparently so that the identities of the individuals who “own” the data are not revealed. Consequently, cloud-based solutions must protect data privacy in an appropriate manner. Meanwhile, much of the existing privacy legislation hinders medical institutions from using cloud services - partly because of the way data management roles for medical data are defined at present and also due to restrictions imposed by the current rules for managing medical data.

Cloud computing has raised several security issues including multi-tenancy, loss of control and trust. Consequently the majority of cloud providers - including

Amazon Web Services (AWS)¹, the Google Compute Engine², HP³, Microsoft's Azure⁴, Citrix CloudPlatform⁵, and RackSpace⁶ do not guarantee specific levels of security and privacy in their Service Level Agreement (SLA)s, as part of the contractual terms and conditions between cloud providers and consumers.

Cloud computing providers virtualize and containerize their computing platforms to be able to share them between different users (or tenants). Multi-tenancy refers to sharing physical devices and virtualized resources between multiple independent users or organizations.

Loss of control is another potential breach of security that can occur where consumers' data, applications, and resources are hosted at the cloud provider's owned premises. As the users do not have explicit control over their data, this makes it possible for cloud providers to perform data mining on the users' data, which can lead to security issues. In addition, when the cloud providers backup data at different data centers, the consumers cannot be sure that their data is completely erased everywhere when they delete their data. This has the potential to lead to misuse of the unerased data. In these types of situations where the consumers lose control over their data, they see the cloud provider as a black-box where they cannot directly monitor the resources transparently.

Trust plays an important role in attracting more consumers by assuring on cloud providers. Due to loss of control (as discussed earlier), cloud users rely on the cloud providers using trust mechanisms as an alternative to giving users transparent control over their data and cloud resources. Therefore, cloud providers build confidence amongst their customers by assuring them that the provider's operations are certified in compliance with organizational safeguards and standards.

The security issues in cloud computing lead to a number of privacy concerns, because privacy is a complex topic that has different interpretations depending on contexts, cultures, and communities. In addition, privacy and security are two distinct topics although security is generally necessary for providing privacy [11, 12].

The right to privacy has been recognized as a fundamental human right by the United Nations [13]. Several efforts have been made to conceptualize privacy by jurists, philosophers, researchers, psychologists, and sociologists in order to give us a better understanding of privacy - for example, Alan Westin's research in 1960 is considered to be the first significant work on the problem of consumer data privacy and data protection. Westin [14] defined privacy as follow.

“Privacy is the claim of individuals, groups, or institutions to determine for themselves when, how, and to what extent information about them is communicated to others.”

¹<https://aws.amazon.com/s3/sla/>

²<https://cloud.google.com/compute/sla>

³<http://www.hpcloud.com/sla>

⁴<http://azure.microsoft.com/sv-se/support/legal/sla/>

⁵<https://www.citrix.se/products/cloudplatform/overview.html>

⁶<http://www.rackspace.com/information/legal/cloud/sla>

The International Association of Privacy Professionals (IAPP) glossary⁷ refers to privacy as appropriate use of information under the circumstances. The notion of what constitutes appropriate handling of data varies depending on several factors such as individual preferences, the context of the situation, law, collection, how the data would be used and what information would be disclosed.

In jurisdictions such as the US, “privacy” is the term that is used to encompass the relevant laws, policies and regulations, while in the EU the term “data protection” is more commonly used when referring to privacy laws and regulations.

Over the past 60 years privacy laws and data protection regulations have been introduced or have evolved, starting from the Universal Declaration of Human Rights in 1948, the European Conventions on Human Right in 1953, the first data protection law passed in Hesse, Germany, in 1970, the Swedish Data Act in 1973, the US Privacy Act in 1974, the OECD Fair Information Principles in 1980, the EU DPD in 1995, the EU e-Privacy Directive in 2002, the California Senate Bill 1386 , which introduces Breach Notification in 2013, and the proposed reform to the EU Directive in 2012.

Legislation that aim to protect privacy of individuals - such as the EU DPD [8], the Gramm-Leach-Bliley Act (GLBA) [15], the Right to Financial Privacy Act (RFPA) [16], the Telecommunications Act of 1996 [17], and the HIPAA [9] can become very complicated and have a variety of specific requirements. Organizations collecting and storing data in clouds that are subject to data protection regulations must ensure that the privacy of the data is preserved appropriately to lay the foundations for legal access to sensitive personal data.

This evolution of privacy legislation over time highlights the importance of privacy in societies that are embracing new technologies - most notably cloud computing and the emerging big data technologies - to cope with the huge amounts of sensitive personal data that are being generated. The resulting deluge of data poses risks for the privacy of individuals. For example, we are seeing sophisticated attacks leading to the theft of databases containing social security data, tax records, and credit card information from online shops. This results in privacy breaches, and the stolen information is often used for identity theft and fraud. Additionally, privacy policies that could be changed periodically in accordance with the preferences of the cloud providers (such as Microsoft, Amazon, and Google) often mean there are unexpected changes in their products privacy settings, which can threaten the privacy of individuals.

The development of a legal definition of cyber crime, the issue of jurisdiction (who is responsible for what information and where are they held responsible for it) and the regulation of data transfers to third-party countries [18] are among other challenging issues when it comes to security in cloud computing. For example, the DPD is the EU’s initial attempt at privacy protection and it contains 72 recitals and 34 articles to harmonize the regulations for information flow within the EU Member States.

⁷<https://iapp.org/resources/glossary>

There is an ongoing effort [19] to replace the EU DPD with a new data protection regulation containing 91 articles that aim to lay out a data protection framework in Europe. The proposed regulations expand the definition of personal data protection to cover any information related to the data subjects irrespective of whether the information is private, public or professional in nature. It also includes definitions of new roles (such as data transfer officers) and proposes restricting the transfer of data to third-party countries that do not guarantee adequate levels of protection. Currently, Argentina, Canada, Guernsey, Jersey, the Isle of Man, Israel, Switzerland, and the US Transfer of Air Passenger Name Data Record are considered to offer adequate protection according to the DPD. The new regulation considers imposing significant penalties for privacy breaches that result from violations of the regulations, for example, such a penalty could be 0.5% of the worldwide annual turnover of the offending enterprise.

In this thesis, we used two main reference platforms, BioBankCloud and Virtual Multidisciplinary Environments Using Cloud Infrastructures (VENUS-C), to build privacy-preserving clouds. The BioBankCloud provides scalable data infrastructure for storage and analysis of Next-Generation Sequencing (NGS) data using an optimized distribution of Apache Hadoop. The aim of VENUS-C was to provide infrastructure for e-Science community to build scalable cloud applications. In the following of this chapter (Section 1.2.1 and Section 1.2.2), we will present these two platforms.

1.2 Reference Platforms

This section presents an architectural view of the BioBankCloud components that has been used in Chapter 5 and Chapter 6. We introduce a new privacy threat model for biomedical clouds in Chapter 4 and as a proof of concept the BioBankCloud case study has been used to validate the proposed model in chapter 5. Chapter 6 describes the implementation of a security framework to build a working prototype of the BioBankCloud. Section 1.2.2 presents an overview of the VENUS-C infrastructure and its middleware Generic Worker (GW) that is used to build a privacy-preserving brain image analysis in Chapter 8.

1.2.1 Scalable Secure Storage BioBankCloud

1.2.1.1 Definition

The BioBankCloud [20] is a collaborative project bringing together computer scientists, bioinformaticians, pathologists, and biobankers. The system is designed as a Platform-as-a-Service (PaaS), i.e., it can be easily installed on cloud computing environments using Karamel and Chef⁸. Primary design goals are flexibility in terms of the analysis being performed, scalability up to very large data sets and

⁸<http://www.karamel.io>

very large cluster set-ups, ease of use and low maintenance cost, strong support for data security and data privacy, and direct usability for users.

The platform encompasses (a) a scientific workflow engine running on top of the popular Hadoop platform for distributed computing, (b) a scientific workflow language focusing on the easy integration of existing tools and simple rebuilding of existing pipelines, (c) support for automated installation, and (d) Role Based Access Control (RBAC). It also features (e) HopsFS, a new version of Hadoop Distributed File System (HDFS) with improved throughput, supported for extended metadata, and reduced storage requirements compared to HDFS, (f) Charon, which enables the federation of clouds at the file system level, and (g) a simple Laboratory Information Management Service with an integrated web interface for authenticating/authorizing users, managing data, designing and searching for metadata, and support for running workflows and analysis jobs on Hadoop. This web interface hides much of the complexity of the Hadoop backend, and supports multi-tenancy through first-class support for *Studies*, *SampleCollections* (DataSets), *Samples*, and *Users*.

As the BioBankCloud name implies, it aims to process biobanks' data within cloud computing environments. A biobank is a biorepository that stores and catalogs human biological material from identifiable individuals for both clinical and research purposes. Recent initiatives in personalized medicine created a steeply increasing demand for sequencing the human biological material stored in biobanks. As of 2015, such large-scale sequencing is under way in hundreds of projects around the world, with the largest single project sequencing up to 100.000 genomes⁹. Furthermore, sequencing also is becoming more and more routine in a clinical setting for improving diagnosis and therapy, especially in cancer [21]. However, software systems for biobanks traditionally managed only metadata associated with samples, such as pseudo-identifiers for patients, sample collection information, or study information. Such systems cannot cope with the current requirement to, alongside such metadata, also store and analyze genomic data, which might mean everything from a few Megabytes (e.g., genotype information from a Single Nucleotide Polymorphism (SNP) array) to hundreds of Gigabytes per sample (for whole genome sequencing with high coverage).

For a long time, such high-throughput sequencing and analysis were only available to large research centers that (a) could afford enough modern sequencing devices and (b) had the budget and expertise to manage high-performance computing clusters. This situation is changing. The cost of sequencing is falling rapidly, and more and more labs and hospitals depend on sequencing information for daily research and diagnosis/treatment. However, there is still a pressing need for flexible and open software systems to enable the computational analysis of large biomedical datasets at a reasonable price. Note that this trend is not restricted to genome sequencing; very similar developments are also happening in other medical areas,

⁹<http://www.genomicsengland.co.uk/>

such as molecular imaging [22], drug discovery [23], or data generated from patient-attached sensors [24].

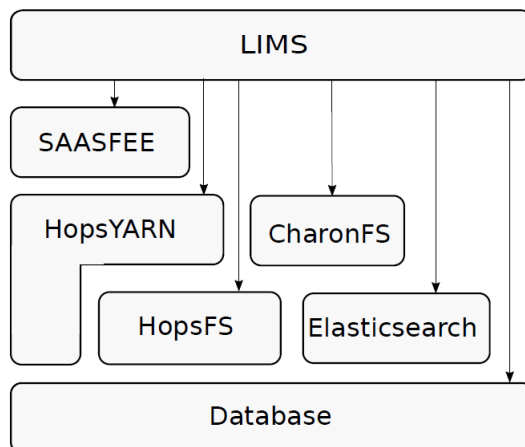


Figure 1.1: BioBankCloud Architecture

The platform has a layered architecture (see Figure 1.1). In a typical installation, users will access the system through the web interface after authentication. From there, she can access all services, such as the enhanced file system HopsFS (see Section 1.2.1.3), the workflow execution engine SAASFEE (see Section 1.2.1.5), the federated cloud service CharonFS, and an Elastic search instance to search through an entire installation. SAASFEE is built over YARN while CharonFS can use HopsFS as a backing store. HopsFS and Elastic search use a distributed, in-memory database for metadata management. Note that all services can also be accessed directly through command-line interfaces.

1.2.1.2 Data Sets for Hadoop

The web interface has integrated a LIMS to manage the typical data items inside a biobank, and to provide fine-grained access control to these items. These items are also reflected in the Hadoop installation. Specifically, BioBankCloud introduces **DataSets** as a new abstraction, where a DataSet consists of a related group of directories, files, and extended metadata. DataSets can be indexed and searched (through Elasticsearch) and are the basic unit of data management in BioBankCloud; all user-generated files or directories belong to a single DataSet. In biobanking, a sample collection would be a typical example of a DataSet. To allow for access control of users to DataSets, which is not inherent in the DataSet

concept, we introduce the notion of **Studies**. A Study is a grouping of researchers and DataSets (see Figure 1.2) and the basic unit of privacy protection (see below).

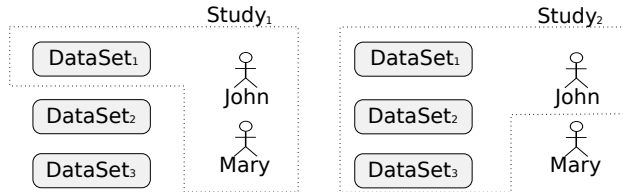


Figure 1.2: Study1 has John and Mary as users and includes DataSet1, while Study2 has only John as a user and includes DataSet1, DataSet2, and DataSet3.

1.2.1.3 Hadoop Open Platform-as-a-Service (Hops)

A full installation of the platform builds on an adapted distribution of the HDFS, called HopsFS, which builds on a new metadata management architecture based on a shared-nothing, in-memory distributed database (see Figure 1.3). Provided enough main memory in the nodes, metadata can grow to TBs in size with this approach (compared to 100GB in Apache HDFS [25]), which allows HopsFS to store 100s of millions of files. The HopsFS architecture includes multiple stateless NameNodes that manage the namespace metadata stored in the database. HopsFS' clients and DataNodes are aware of all NameNodes in the system. HopsFS is highly available: whenever a NameNode fails the failed operations are automatically retried by clients and the DataNodes by forwarding the failed requests to a different live NameNode. MySQL Cluster [26] is used as the database, as it has high throughput and is also highly available, although any distributed in-memory database that supports transactions and row level locking could be used. On database node failures, failed transactions are re-scheduled by NameNodes on surviving database nodes.

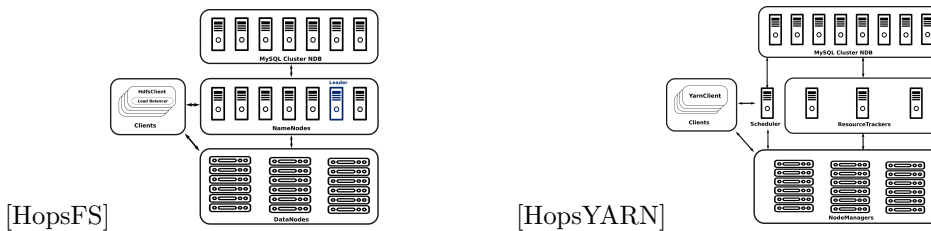


Figure 1.3: HopsFS and HopsYARN architectures.

The consistency of the file system metadata is ensured by implementing serialized transactions on well-ordered operations on metadata [27]. A leader NameNode

is responsible for file system maintenance tasks, and leader failure triggers our own leader-election service based on the database [28].

HopsFS can reduce the amount of storage space required to store genomic data while maintaining high availability by storing files using [29] erasure coding, instead of the traditional three-way replication used in HDFS. Erasure-coding can reduce disk space consumption by 44% compared to three-way replication. In HopsFS, an ErasureCodingManager runs on the leader NameNode, managing file encoding and file repair operations, as well as implementing a policy that places file blocks on DataNodes in such a way that ensures that, in the event of a DataNode failure, affected files can still be repaired.

1.2.1.4 HopsYARN

HopsYARN is our implementation of Apache YARN, in which we have (again) migrated the metadata to MySQL Cluster. YARN's ResourceManager is partitioned into (1) ResourceTracker nodes that process heartbeats from and send commands to NodeManagers, and (2) a single scheduler node that implements all other ResourceManager services, see Figure 1.3. If the scheduler node fails, our leader election service will elect a ResourceTracker node as the new scheduler that then loads the scheduler state from the database. HopsYARN scales to handle larger clusters than Apache YARN as resource tracking has been offloaded from the scheduler node to other nodes and resource tracking traffic grows linearly with cluster size. This will, in time, enable larger numbers of genomes to be analyzed in a single system.

1.2.1.5 SAASFEE

To process the vast amounts of genomic data stored in today's biobanks, researchers have a diverse ecosystem of tools at their disposal [30]. Depending on the research question at hand, these tools are often used in conjunction with one another, resulting in complex and intertwined analysis pipelines. Scientific workflow management systems (SWfMSs) facilitate the design, refinement, execution, monitoring, sharing, and maintenance of such analysis pipelines. SAASFEE [31] is a SWfMS that supports the scalable execution of arbitrarily complex workflows. It encompasses the functional workflow language Cuneiform as well as Hi-WAY, a higher-level scheduler for both Hadoop YARN and HopsYARN. See Figure 1.4 for the complete software stack of SAASFEE.

1.2.2 VENUS-C

The VENUS-C project was an initiative to develop, test and deploy an industry-quality, highly-scalable and flexible cloud infrastructure for e-Science [32]. The overall goal was to empower the many researchers who do not have access to super-computers or big grids, by making it easy to use cloud infrastructures. For this to

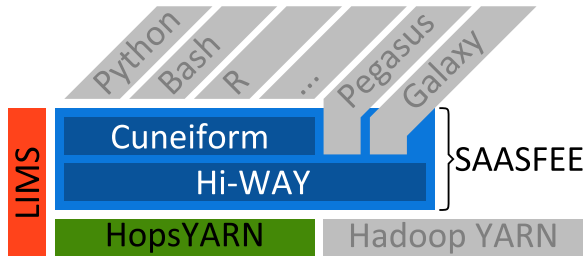


Figure 1.4: The software stack of the scientific workflow management system SAASFEE, which comprises the functional workflow language Cuneiform as well as the Hi-WAY workflow scheduler for Hadoop. Cuneiform can execute foreign code written in languages like Python, Bash, and R. Besides Cuneiform, Hi-WAY can also interpret the workflow languages of the SWfMSs Pegasus and Galaxy. SAASFEE can be run both on HOPS as well as Apache Hadoop. SAASFEE and HOPS can be interfaced and configured via the web interface provided by the LIMS.

be feasible, the project minimized the efforts that such researchers need to spend for development and deployment in order to do computations in the cloud. This has the added advantage of reducing the costs of operating the cloud.

Requirements from different scientific use cases were collected in the project and, as a result, the platform was designed with the capability of supporting multiple programming models, such as batch processing, workflow execution or even Map/Reduce (MR) [33] at the same time.

1.2.2.1 VENUS-C Architecture

Figure 1.5 illustrates the generalized VENUS-C architecture and shows the basic steps that a researcher must perform in order to use VENUS-C. These steps are independent of the programming model that is used. Firstly the researcher uploads the locally available data to the cloud storage. The next step is to submit a job. So-called dedicated Programming Model Enactment Services (PMES) are provided for this purpose. These services enable the researchers to perform tasks such as managing jobs or scaling the resources used in the cloud while simultaneously shielding the researchers from the underlying cloud infrastructure and the specific implementations of different infrastructures through Open Grid Service Architecture - Basic Execution Services (OGSA-BES) compliant interfaces [34]. OGSA-BES is an open standard for basic execution services and widely used in grid communities for submitting jobs. The third step involves carrying out the required computations. For this, the necessary application and job specific data are transferred to the compute node. After the computation has finished, the fourth step consists of transferring the resulting data to the cloud storage. In the fifth and final step, the researcher can download the results from the cloud to local facilities.

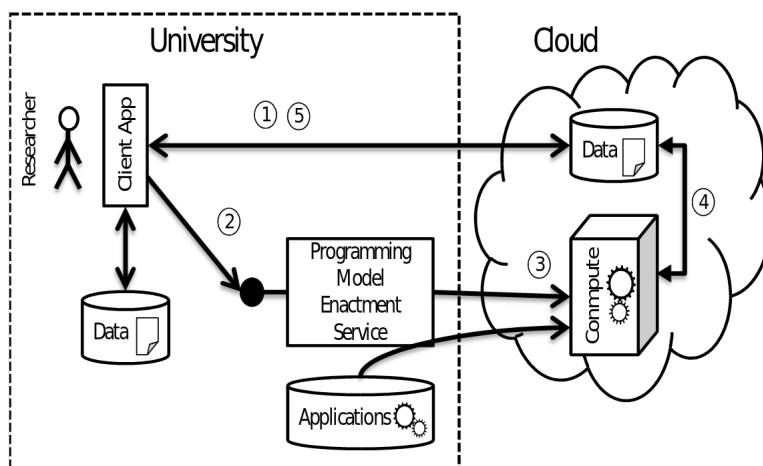


Figure 1.5: VENUS-C architecture

1.2.2.2 Generic Worker

The GW module has been developed in the VENUS-C project. Following the general VENUS-C architecture, the GW represents a reference implementation for a batch processing programming model and is available for public download.

The GW is basically a worker process (similar to Windows Service or UNIX daemon processes) that can be started on Virtual Machines (VM)s in the cloud. Being able to run many VM at the same time with a GW worker process provides great horizontal scaling capabilities and allows work items to be distributed across the machines according to the user's requirements.

Figure 1.6 shows how the GW is designed internally. Using this approach, researchers are able to upload applications and data to storage, which is connected to the Internet so that the GW can also access it. The GW design supports a broad selection of different protocols and storage services. In addition to the data and the application that should be run, the GW also needs a description of this application containing meta-data about it. This information allows the GW to understand parameters like input and output files enabling a proper execution of the application by the GW.

Jobs are submitted using the PMES. To make this safe, different security mechanism such as a Security Token Service (STS) to validate, issue and exchange security tokens based on the well-known WS-Trust protocol and username/password can be used. The PMES stores all the incoming jobs in an internal job queue based on a table (Job Index); an additional table is used for the job details. The GW driver processes continuously look for new jobs in this queue. As soon as a driver process finds a job in the queue, it will pull the job from the queue, and check the application and data storage to find out if everything that is needed is available, namely

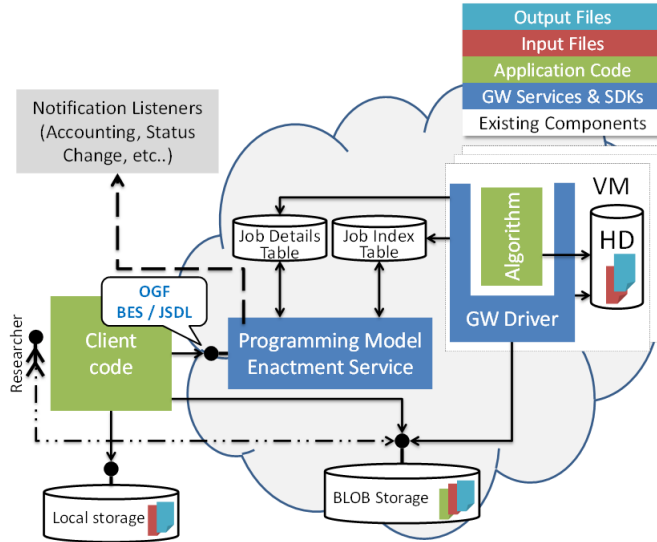


Figure 1.6: Simplified internal GW architecture

all the required input data and the relevant application binaries. If these are in place, the job can be executed. The driver process that found the job marks the job as being processed by that particular driver in the Job Details Table (JDT), and starts downloading the input data to the local hard disk of the VM. If application or data are not yet available, the job will be put back into the queue to wait for the missing files. The driver process also checks whether the application is already present on the VM and, if necessary, the application will be downloaded as well. Thus, the GW process follows a data-driven pull model, allowing simple workflows where jobs rely on the output of other jobs.

Once the application is available, the driver process retrieves information on how to call the application and then launches it. After the application terminates, the results are made persistent by uploading them to the data storage. Finally, the driver process uses the JDT to mark the job as either completed or failed, depending on the exit code of the application. Researchers who used the PMES client-side notification will be notified about this event. There are several notification-plugins available, e.g., sending emails or putting messages in a queue for every event. Researchers can also query the PMES to check the current state of a job.

1.3 Research Questions and Contributions

The main contributions (C1-C3) of this dissertation are aligned along three main research questions (Q1-Q3), which are introduced and briefly discussed in the fol-

lowing.

- **Q1:** Can we develop a methodology to formulate privacy requirements and threats to facilitate compliance with data protection regulations?
- **Q2:** How do we build privacy-preserving cloud-based systems from existing approaches in security and privacy?
- **Q3:** How do we increase the safety of an Operating System (OS) by reducing the risk of kernel exploits?
- **C1:** The first contribution of this thesis is to offer a better understanding of privacy requirements and corresponding threats. For this purpose, a specific methodology for modeling threats to privacy in relation to processing sensitive data in various cloud computing environments has been constructed. This is known as the CPTM methodology. This methodology involves applying Method Engineering (ME) [35] in order to specify the relevant characteristics of a cloud privacy threat modeling methodology, different steps in the proposed methodology and the corresponding products (Chapter 4). We applied the CPTM to a cloud computing software project that aims to provide a PaaS model for storing and processing sensitive medical data according to the EU DPD. This case study consisted of identifying the privacy requirements of the DPD to produce a privacy threat model that includes threat analysis, risk evaluation and threat mitigation measures (Chapter 5). The outcome publications of this part are [36, 37, 38].
- **C2:** We propose three usable privacy-preserving cloud-based architectures (Chapters 6, 7 and 8) to process genomics, clinical and brain imaging datasets. For this purpose, we studied an array of existing and state of the art research projects to identify the gap in the field and implemented three architectures. The data that is stored in these architectures includes information about population-scale genomic data, patient cancer data, and brain images and hence must conform to privacy requirements. These architectures ensure that all storage and processing of the sensitive data will be appropriate and will not involve risks to the privacy of the subjects. The outcome publications of this part are [39, 40, 41, 42].

In summary, the following proposed architectures contribute to building trustworthy cloud models with the capability of providing appropriate control according to privacy regulations over the users data in the cloud.

- The first architecture was implemented as a proof-of-concept for the CPTM. This included implementation of 8 key privacy requirements and implementing countermeasures for 26 critical threats to privacy of genomic data in a PaaS cloud (Chapter 6).

- The second architecture was implemented to demonstrate the feasibility of a privacy-preservation solution for aggregated queries over multiple datasets from different data sources. This solution provides a platform-independent and open-source anonymization toolkit that can be used for publishing the data about sample availability to a private cloud (Chapter 7).
- The third implementation was Scalable Brain Image Analysis (ScaBIA) architecture that incorporates a privacy-preserving implementation of a new model for running Statistical Parametric Mapping (SPM) jobs using Microsoft Azure. The proposed model enhances the methods for secure access and sharing of raw brain imaging data and improves scalability compared to the “single PC”-execution model (Chapter 8).
- **C3:** Finally, we introduce quantitative measuring and evaluation of the security of privileged code (such as in a hypervisor or kernel. This investigation included examining the kernel traces generated by running popular user applications and produced recommendations at the lines-of-code level. The proposed solution contributes to providing better isolation of user processes for the challenging issue of multi-tenancy. Our approach can be used to identify the risky portions of the OS kernel and secure them through a new concept called “safely-reimplement” (Chapter 9 and Chapter 10).

1.4 Research Method

This section lists the steps that have been undertaken in the research that is described in this thesis.

- A preliminary study of the appropriate literature was performed. This encompassed background theory for relevant topics including big data, cloud computing, virtualization, security protocols, privacy-enhancing technologies, personal data protection legislation, threat modeling, and software engineering.
- The background literature review was followed by a literature review of the state-of-the-art in security and privacy for cloud computing. This included classification of the cloud provider activities for the search strategy to identify relevant literature. The results of this review were then used to identify existing gaps in the current research on privacy-preservation in order to suggest areas for further investigation.
- Theoretical models consisting conceptual, logical, and physical architectures of the developing privacy-preserving systems were provided.
- Implemented and conducted experiments in several open source architectures using popular cloud programming and software environments such as Apache

Hadoop, Amazon EC2, Microsoft Azure, Vagrant, Docker, Java, Python, MATLAB and R. We verified our implementations by various test scenarios to identify any potential defects.

- Presented papers at workshops and conferences and publication of proceedings for reviews, comments, and valuable feedback.
- Visited external research groups in the field, in addition to participating in various summer schools and tutorials.

1.5 List of Scientific Papers

Publications that have directly stemmed from this work are:

I **A. Gholami**, A.-S. Lind, J. Reichel, J.-E. Litton, A. Edlund, and E. Laure, “Design and implementation of the advanced cloud privacy threat modeling,” *International Journal of Network Security & Its Applications*, Vol. 8, No. 2, March 2015.

Author’s contributions: I am the main author, identified the privacy threats according to the DPD and developed a proof-of-concept for the Advanced CPTM methodology.

II **A. Gholami** and E. Laure, “Big data security and privacy issues in the cloud,” *International Journal of Network Security & Its Applications*, Vol. 8, No. 1, January 2016.

Author’s contributions: I am the main author, identified the related research and state-of-the-art in the area of big data security and privacy.

III **A. Gholami** and E. Laure, “Advanced cloud privacy threat modeling,” *The Fourth International Conference on Software Engineering and Applications, CCSIT, SIPP, AISC, CMCA, SEAS, CSITEC, DaKM, PDCTA, NetCoM*, pp. 229–239, 2016.

Author’s contributions: I am the main author, performed the requirements analysis, devised the design and implemented the methodology.

IV **A. Gholami** and E. Laure, “Security and privacy of sensitive data in cloud computing: a survey of recent developments,” *The Seventh International Conference on Network and Communication Security (NCS), Wireless & Mobile Networks (WiMoNe-2015)*, pp. 131–150, 2015.

Author’s contributions: I am the main author, classified the related research and state-of-the-art according to the cloud provider activities.

V A. Bessani, J. Brandt, M. Bux, V. Cogo, L. Dimitrova, J. Dowling, **A. Gholami**, K. Hakimzadeh, M. Hummel, M. Ismail, E. Laure, U. Leser, J.-E. Litton, R. Martinez, S. Niazi, J. Reichel, and K. Zimmermann, “Biobankcloud:

a platform for the secure storage, sharing, and processing of large biomedical data sets,” in The First International Workshop on Data Management and Analytics for Medicine and Healthcare (DMAH 2015), September 2015.

Author’s contributions: All authors contributed equally to this work. I wrote the Security Model Section and also revised other parts of the paper.

- VI **A. Gholami**, J. Dowling, and E. Laure, “A security framework for population-scale genomics analysis,” in 2015 International Conference on High Performance Computing & Simulation, HPCS 2015, Amsterdam, Netherlands, July 20-24, 2015, pp. 106–114, IEEE, DOI: 10.1109/HPCSim.2015.7237028.

Author’s contributions: I am the main author, proposed the security architecture, devised the design, built the components and validated the proposed framework.

- VII **A. Gholami**, A.-S. Lind, J. Reichel, J.-E. Litton, A. Edlund, and E. Laure, “Privacy threat modeling for emerging biobankclouds,” *Procedia Computer Science*, vol. 37, no. 0, pp. 489 – 496, 2014. The 5th International Conference on Emerging Ubiquitous Systems and Pervasive Networks (EUSPN-2014)/The 4th International Conference on Current and Future Trends of Information and Communication Technologies in Healthcare (ICTH 2014)/ Affiliated Workshops.

Author’s contributions: I am the main author, contributed extensively to formulate the privacy requirements, conducted risk analysis for the identified threats and defined the methodology.

- VIII **A. Gholami**, E. Laure, P. Somogyi, O. Spjuth, S. Niazi, and J. Dowling, “Privacy-preservation for publishing sample availability data with personal identifiers,” *Journal of Medical and Bioengineering*, vol. 4, pp. 117–125, April 2014.

Author’s contributions: I am the main author, devised the design, provided the pilot implementation of the anonymization toolkit and partial implementation of the integration server.

- IX **A. Gholami**, G. Svensson, E. Laure, M. Eickhoff, and G. Brasche, “Scabia: Scalable brain image analysis in the cloud,” in CLOSER 2013 - Proceedings of the 3rd International Conference on Cloud Computing and Services Science, Aachen, Germany, 8-10 May, 2013 (F. Desprez, D. Ferguson, E. Hadar, F. Leymann, M. Jarke, and M. Helfert, eds.), pp. 329–336, SciTePress, 2013, DOI=10.5220/0004358003290336, ISBN=978-989-8565-52-5.

Author’s contributions: I am the main author, devised the design, provided the pilot implementation and performed the experiments in the Microsoft Azure Cloud.

Other scientific papers during my pre-doctoral studies that are not included in this thesis:

- D. Cameron, **A. Gholami**, D. Karpenko, and A. Konstantinov, “Adaptive Data Management in the Arc Grid Middleware,” *Journal of Physics: Conference Series*, vol. 331, no. 6, p. 062006, 2011.
- F. Hedman, M. Riedel, P. Mucci, G. Netzer, **A. Gholami**, M. Memon, A. Memon, and Z. Shah, “Benchmarking of Integrated OGSA-BES with the Grid Middleware,” in *Euro-Par 2008 Workshops - Parallel Processing*, vol. 5415 of *Lecture Notes in Computer Science*, pp. 113–122, Springer Berlin Heidelberg, 2009.

1.6 Thesis Outline

The remainder of this dissertation is organized as follows: Chapter 2 presents the background material. Chapter 3 presents the previous research in the field. Chapter 4 describes the CPTM methodology. Chapter 5 describes a case study to be implemented using CPTM for compliance with the EU DPD. Chapter 6 describes the design and implementation of a security framework according to the CPTM. Chapter 7 presents a privacy-preserving solution for publishing the sample availability data with personal identifiers. Chapter 8 describes ScaBIA to securely process the brain imaging data using statistical parametric approach. Chapter 9 presents a novel approach for quantifying and minimizing the risk of kernel exploitation. Chapter 10 presents a reference monitor for Lind. Chapter 11 summarizes our findings and conclusions. Finally, Chapter 12 discusses the future work.

Chapter 2

Background

2.1 Big Data Infrastructures

Computers produce soaring rates of data [43, 44, 5] that is primarily generated by Internet of Things (IoT), telescopes, NGS machines, scientific simulations and other high throughput instruments which demand efficient architectures for handling the new datasets. In order to cope with this huge amount of information, “Big Data” solutions such as the Google File System (GFS) [45], MR [33], Apache Hadoop and the HDFS [46, 47] have been proposed both as commercial or open-source.

Key vendors in the IT industry such as IBM [48], Oracle [49], Microsoft [50], HP [51], Cisco [52] and SAP [53] have customized these big data solutions. There have been different definitions and claims relating to “Big Data” that have been put forward as the concept has emerged in recent times. Over the past few years, the National Institute of Standards (NIST) has formed a big data working group. This is a community with joint members from industry, academia and government that aims to develop consensus definitions, taxonomies, secure reference architectures, and a technology roadmap [54]. This group has characterized big data as extensive datasets that are diverse; that include structured, semi-structured, and unstructured data from different domains (variety); that are of large orders of magnitude (volume); that arrive at a fast rate (velocity); and that change their other characteristics (variability) [1].

Figure 2.1 shows the four main parts of the big data ecosystem: data sources, data transformation processes, the data storage and retrieval infrastructure and the users of the data [1]. In addition, there are supporting subsystems for ensuring the security of the big data and for managing the big data - these subsystems provide services to the other components of the big data ecosystem.

The data sources part of the ecosystem contains the big data to be served for a specific purpose that can transform in different ways. When sets of big data are initially collected, the datasets with similar source structures are combined.

Then metadata is created to facilitate lookup methods for the combined data.

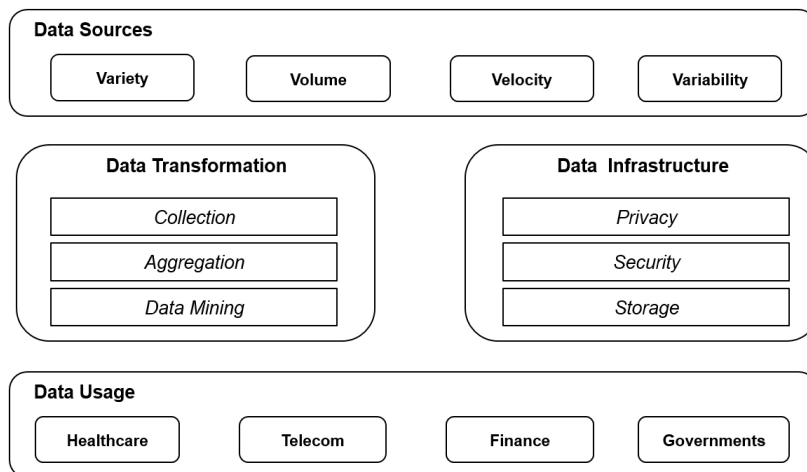


Figure 2.1: Big data ecosystem reference architecture (image courtesy of NIST [1])

Datasets with dissimilar metadata are also aggregated into a larger collection for matching purposes, for example, by correlating the aggregated data with identifiers after applying security policies.

Data mining may then be used for analyzing the resulting aggregated data from different perspectives or to extract specific information from the data. The result of the data mining process will be a summary of the information that identifies relationships within the data - this could either be descriptive (for example, information about the existing data) or predictive (such as forecasts based on data).

The big data infrastructure (which is on the right-hand side in Figure 2.1) consists of data storage systems, servers, and networking to support the data transformation functions and for storing data in Structured Query Language (SQL) and NoSQL databases on-demand. The storage component of the infrastructure supports the efficient processing of big data by providing computing and storage technologies that are appropriate for the transformation and usage scenarios where conditioning (de-identification, sampling and fuzzing) may be required.

Table 2.1 summarizes the big data technologies from batch processing to real-time streaming analytics with most significant stages and products [5]. The concept of big data and related technologies are used in Chapter 6 to build a secure NGS data analytic engine.

2.2 Cloud Computing

When considering cloud computing, we need to be aware of the types of services that are offered, the way those services are delivered to those using the services,

Stage/Year	Characteristics	Examples
Batch Processing / 2003 - 2008	Big amount of data is collected, entered, processed and then the batch results are produced. Distributed file systems are used for fault-tolerant and scalability. Parallel programming models such as MR are used for efficient processing of data.	GFS, MR, HDFS, Apache Hadoop
Ad-hoc (NoSQL) / 2005 - 2010	Support random read/write access to overcome shortcomings of distributed file systems that are appropriate for sequential data access. NoSQL databases solve this issue by offering column based or key-value stores, in addition, to support for storage of large unstructured datasets such as documents or graphs.	CoachDB, Redis, Amazon DynamoDB, Google Big Table, HBase, Cassandra, MongoDB
SQL-like / 2008 - 2010	Simple programming interfaces to query and access the data stores. This approach provides functionalities similar to the traditional data warehousing mechanisms.	Apache Hive/Pig, PrestoDB, HStore, Google Planner
Stream Processing / 2010 - 2013	Data are pushed continuously as streams to servers for processing before storing them. Streaming data usually have unpredictable incoming patterns. Such data streams are processed using fast, fault-tolerant, and high availability solutions.	Hadoop Streaming, Google Big Query, Google Dremel, Apache Drill, Samza Apache Flume/Hbase, Apache Kafka/Storm
Real-time Analytical Processing / 2010 - 2015	Automated decision making for streams that are generated from the machine-to-machine applications or other live channels. This architecture helps to apply real-time rules for the incoming events and existing events within a domain.	Apache Spark, Amazon Kinesis, Google Dataflow

Table 2.1: Evolution of Big Data from batch to real-time analytics processing [5]

and the different types of people and groups that are involved with cloud services.

Many formal definitions of cloud computing exist. In 2011, the NIST defined cloud computing and its characteristics as follows [6].

“Cloud computing is 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. This cloud model is composed of five essential characteristics, three service models, and four deployment models.”

The five essential characteristics of cloud computing are briefly described in Table 2.2.

Characteristic	Description	Application
On-demand Self-service	For automatically providing a consumer with provisioning capabilities as needed	Server, Time, Network and Storage
Broad Network Access	For heterogeneous thin or thick client platforms	Smartphones, tablets, PCs, wide range of locations
Resource Pooling	The provider’s computing resources are pooled to serve multiple consumers using a multi-tenant model	Physical and virtual resources with dynamic provisioning
Rapid Elasticity	Capabilities can be elastically provisioned and released, in some cases automatically, to scale rapidly outward and inward with demand	Adding or removing nodes, servers, resource or instances
Measured Service	Automated control and optimization of a resource through measuring or monitoring services for various reasons, including billing, effective use of resources, or predictive planning	Storage, billing, processing, bandwidth, and active user accounts

Table 2.2: Cloud computing characteristics [6]

2.2.1 Concepts in Cloud Computing

Cloud computing delivers software, platforms and infrastructure as services based on pay-as-you go models. Cloud service models can be deployed for on-demand storage and computing power as Software-as-a-Service (SaaS), PaaS and Infrastructure-as-a-Service (IaaS), as shown in Figure 2.2 (service layer). Cloud service models can be summarized as follows [6].

- **SaaS** allows consumers to run applications by virtualizing hardware on cloud providers, e.g., Salesforce Customer Relationship Management (CRM)¹ or Oracle Sales Cloud.
- **PaaS** makes it possible to deploy custom applications with their dependencies within an environment called a container. Containers deliver isolation and resource management in Linux environments. An OS container isolates a process from the rest of the system. The Google App Engine², Oracle Java Cloud³, Heroku⁴, OpenShift⁵, dotCloud⁶, and Cloud Foundry⁷ are examples of PaaS clouds.
- **IaaS** provides a hardware platform (including, for example, virtual machines, processing, storage, networks and database services) as a service, such as with the Amazon Elastic Compute Cloud (EC2)⁸, Google Compute Engine or Oracle Compute Service.

Cloud services can be delivered to consumers using different cloud deployment models: private cloud, community cloud, public cloud, and hybrid cloud.

- **Private Cloud:** In a private cloud, the cloud infrastructure is provided within the Intranet of an organization. It can be owned and operated by the organization or by third-party partners within the premises of the organization.
- **Community Cloud:** The cloud infrastructure of a community cloud is shared among a community of organizations with common concerns, such as their missions, security requirements, policies and regulatory compliance.
- **Public Cloud:** For a public cloud, the cloud infrastructure is built for open use over the public Internet and it is delivered to consumers through subscription. A public cloud can be operated by a business, or by an academic or government organization, or by a combination of these.
- **Hybrid Cloud:** The cloud infrastructure of a hybrid cloud is built from two or more other types of clouds (that is, private, community, or public clouds). Hybrid clouds make it possible to share resources and support data and application portability.

¹<https://www.salesforce.com/crm/>

²<https://appengine.google.com/>

³<https://cloud.oracle.com/java/>

⁴<https://www.heroku.com>

⁵<https://www.openshift.com/>

⁶<https://www.dotcloud.com/>

⁷<https://www.cloudfoundry.org/>

⁸<https://aws.amazon.com/ec2/>

Such cloud deployment models are usually built within data centers for cloud computing that belong to one or multiple organization, where depending on the deployment model can have relatively homogeneous or heterogeneous software and hardware platforms. A data center or computer center is a facility used to house computer systems and associated components, such as storage and network systems. It generally includes redundant or backup power units, redundant network connections, air conditioning, and fire safety controls. Cloud data centers run a small number of very large applications, where cloud computing workloads are designed to gracefully tolerate large numbers of component faults with little or no impact on service level performance and availability.

The NIST cloud computing reference architecture in Figure 2.2 defines five major actors in the cloud arena: cloud consumers, cloud providers, cloud carriers, cloud auditors and cloud brokers. Each of these actors is an entity (either a person or an organization) that participates in a transaction or process, and/or performs tasks in cloud computing [2].

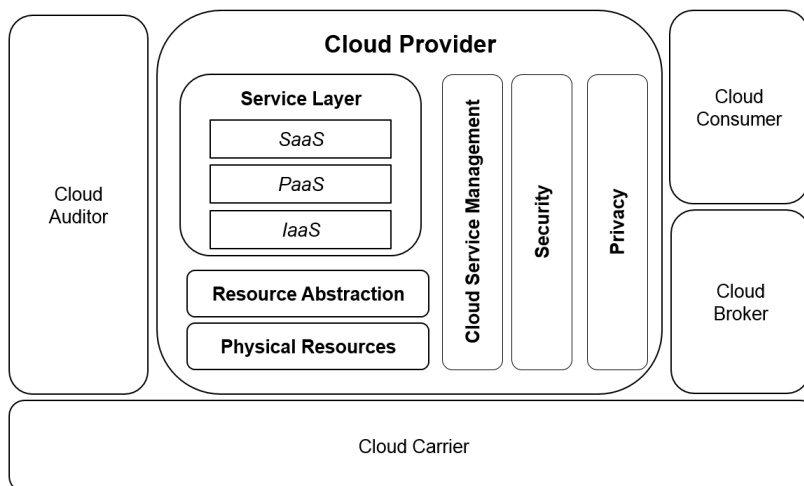


Figure 2.2: Cloud computing reference architecture (image courtesy of NIST [2])

A cloud consumer is a person or organization that uses services from cloud providers in the context of a business relationship. A cloud provider makes cloud services available to interested users. A cloud auditor conducts independent assessments of cloud services, operations, performance and security in relation to the cloud deployment. A cloud broker manages the use, performance and delivery of cloud services and establishes relationships between cloud providers and cloud consumers. A cloud carrier provides connectivity and transport of cloud services from cloud providers to cloud consumers through the underlying network.

While it is practical and cost effective to use cloud computing, there can be issues with security when using systems that are not provided in-house. To look into these and find appropriate solutions, there are several key concepts and technologies that are widely used in cloud computing that need to be understood, such as virtualization mechanisms, varieties of cloud services, and “container” technologies.

2.2.2 Virtualization

The use of virtualization technology as an engine of cloud computing has been growing over the past few decades. Historically, virtualization technology appeared concurrently with time-sharing systems around the late 1960s when IBM developed Control Program (CP)/Cambridge Monitor System (CMS) [55]. The CMS was an existing single-user OS machine. Later in 1970, IBM introduced VM370 [56], where a CP acted as the piece of software to bridge between an application and the relevant hardware.

After few years, in 1974, Gerald J. Popek and Robert P. Goldberg [57] formalized the properties of virtual machines. They defined a VM as “an efficient, isolated duplicate of the real machine.” *Efficiency* ensures that the speed of the VM is close to the the real hardware speed. *Isolating* the VM ensures several instances of a particular VM to be run without interfering with each other. *Duplicating* the real machine means that the VM behaves identically to the real machine, so programs that run inside a VM cannot tell whether they are running on the real machine or a VM.

During the 1990s, other organizations, such as VMWare, introduced more sophisticated products such as Elastic Sky X (ESX)⁹ servers, VMWare workstation and a Personal Computer (PC)¹⁰ for Macintosh systems (as Macs were then called) to run Microsoft Windows.

Over the past few years, major vendors (such as Amazon, Microsoft, and Google) have provided VMs, via their clouds, that customers could rent. These clouds utilize the hardware resources and support live migration of VMs in addition to dynamic load-balancing and on-demand provisioning. This means that, by renting VMs via a cloud, the entire datacenter footprint of a modern enterprise could be reduced from thousands of physical servers to a few hundred (or even dozens) of virtualization hosts.

Virtualization Mechanisms: A Virtual Machine Monitor (VMM) or hypervisor is a key component in virtual computer systems - it resides between the VMs and the real machine hardware and is used to control the virtualized resources [58]. The VMM or hypervisor makes it possible to run several isolated virtual machines on the same physical host. Hypervisors can be divided into two groups [59] as follows.

⁹<http://www.vmware.com/products/es-xi-and-esx/overview>

¹⁰<https://support.microsoft.com/en-us/kb/958559>

- **Type I:** Here the hypervisor runs directly on the real system hardware, and there is no OS under it. This approach is efficient as it eliminates any intermediary layers. Another benefit with this type of hypervisor is that security levels can be improved by isolating the guest VMs. That way, if a VM is compromised, it can only affect itself and will not interfere with the hypervisor or other guest VMs.
- **Type II:** The second type of hypervisor runs on a hosted OS that provides virtualization services, such as Input/Output (I/O) device support and memory management. All VM interactions, such as I/O requests, network operations, and interrupts, are handled by the hypervisor.
- **Trap/Emulate:** In the “trap and emulate” approach, a guest VM will attempt to access a physical resource. This will trigger a trap (exception) to invoke the hypervisor. The hypervisor will, in turn, emulate the instruction, for example, updating the state of the Central Processing Unit (CPU).
- **Binary Translation:** This approach translates the instructions before they are executed, and sensitive instructions are replaced with traps and other code.
- **Paravirtualization:** In this approach the processor instruction set architecture is augmented with hyper calls and the sensitive instructions are removed or the number of traps is reduced. A hypercall is a trap from a domain to the hypervisor (similar to a system call from an application to the kernel). Hypercalls are used by domains to request privileged operations such as updating page tables.
- **Hardware Support:** In the period 2005-2006, AMD and Intel introduced two new architectures with similar functionalities to support virtualization through AMD-V [60] and Intel Virtualization Technology (VT) [61].

Figure 2.3 shows a comparison of two popular open source hypervisors Xen¹¹ and Kernel Virtual Machine (KVM)¹² (respectively of *Type I* and *Type II*). As it is depicted, Xen runs directly on the underlying hardware system and inserts a virtualization layer between the system hardware and the virtual machines. The OSs running in the VM interact with the virtual resources as if they were actually physical resources. While KVM is a virtualization feature in the Linux Kernel that makes it possible to safely execute guest code directly on the host CPU. As illustrated in Figure 2.3, KVM provides full native virtualization solution for Linux on x86 hardware containing virtualization extensions (Intel VT or AMD-V). A modified version of Quick Emulator (QEMU)¹³ provides emulation for Basic Input/Output System

¹¹<http://xen.org/products/xenhyp.html>

¹²<http://www.linux-kvm.org/>

¹³<http://wiki.qemu.org/>

(BIOS), Peripheral Component Interconnect (PCI) bus, Universal Serial Bus (USB) bus, Small Computer System Interface (SCSI) disk controllers, network cards, file systems and serial I/O. KVM provides strong isolation between VMs through the default use of mandatory access control policies. VM instances inside KVM run as processes that are confined using SELinux policies.

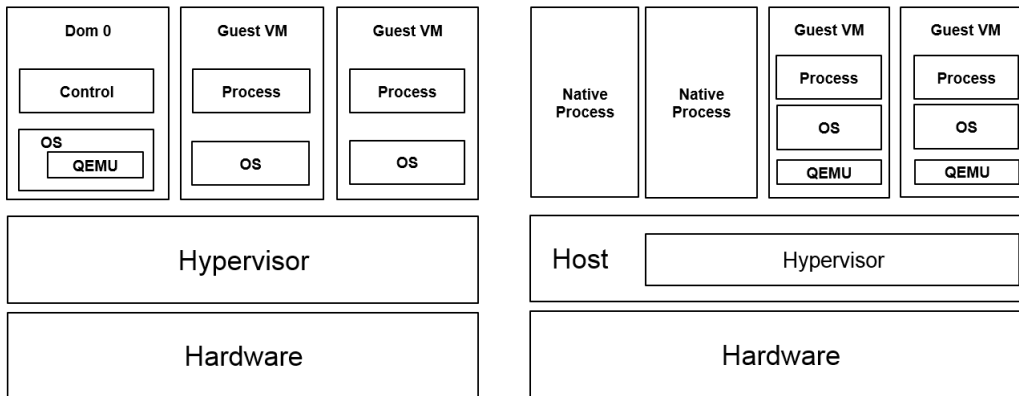


Figure 2.3: Comparison of Type-I and Type II virtualization architectures for Xen and KVM

2.2.3 Container Technology

Linux Container (LXC) technology is considered as a next-generation cloud and it has become an important part of the cloud computing infrastructure because of its ability to run several OS-level isolated VMs within a host with a very low overhead. LXC is built on modern kernel features. A LXC resembles a light-weight execution environment within a host system that runs instructions native to the core CPU while eliminating the need for instruction level emulation or just-in-time compilation [62]. LXC contains applications, configurations, and the required storage dependencies, in a manner similar to the “just enough OS”.

Using containers, several applications can share an OS, binaries or libraries, which results in significant increases in efficiency compared to using hypervisors. For example, the portability of applications and the provisioning time of VMs is very low with container technologies [63]. LXC technologies were introduced in the 1980s, starting with the `chroot` (change root) command, and evolving into popular container managers such as Docker.

- **Chroot:** The Unix `chroot` system call, which was introduced as part of Unix version 7 in 1979, can be considered as the first step in the evolution

of containerization. The `chroot` call changes the root directory of the calling process to a specified path, where the root directory is known by all children of the calling process. This feature is used by some containers for isolation and sharing the underlying file system. The Unix `chroot` is often used when building system images by changing root to a temporary directory, downloading and installing packages in `chroot` or compressing `chroot` as a system root file system.

- **FreeBSD Jail** [64] extended `chroot` in 1998 to provide enhanced security. FreeBSD jail settings can explicitly restrict access outside the sandbox environment by files, processes, and user accounts (including accounts created by the Jail definition specifically for that purpose). Jail can, therefore, define a new root user, which has full control inside the sandbox, but that cannot reach anything outside.
- **Namespaces** were introduced in 1992 [65] for process-based resource isolation. Namespaces provide tools for isolating each Control Groups (cgroups) view of global resources such as details about file systems, processes, network interfaces, Inter Process Communication (IPC), host names, and user identifiers. Processes in a particular namespaces are invisible to other processes because they think that they are the only processes on the system and because “connectivity” is only permitted with the parent namespaces. The Unix namespaces are categorized as follows.
 - **mnt**: Processes in different mount namespaces can have different views of the file system hierarchy. One use of mount namespace is to create environments that are similar to `chroot` Jail. However, mount namespaces are more secure and flexible than `chroot`.
 - **pid**: Each process has a `/proc/[pid]/ns/` subdirectory containing one entry for each namespace. The `pid` namespaces also allow each container to have its own `init` with `PID = 1` as the ancestor of all other processes to manage various system initialization tasks and to terminate child processes. The `init` (initialization) is the first process that starts when an operating system is booted and it continues running until the system is shut down. The `init` process can be considered as the parent of all the processes in the system and it is executed and responsible for all other processes.
 - **net**: Network namespace provides isolation for network devices, IPv4 and IPv6 stacks, port numbers (sockets), routing, and firewalls. Each container can have its own (virtual) network device and its own applications that bind to the per-namespace port number, where routing policies in the host can forward network packets to the network device bound to a specific container. For example, a host can run multiple separate containers with web servers running on port 80. Network namespace also

supports bridge filtering, stateful firewalls and Virtual LAN (VLAN)s through the exclusive use of virtual network devices.

- **ipc**: AT&T’s Unix System V introduced three forms of IPC facilities: message queues, semaphores, and shared memory. Objects created in an IPC namespace are visible to all other processes that are members of that namespace, but are not visible to processes in other IPC namespaces. This feature makes it possible for a process to think it is the only process running on the server. Each IPC object has a unique IPC identifier associated with it. The container can run its own **init** process with *PID* = 1 while the host sees that process with a different process identifier.
- **uts**: A process has a separate copy of the **hostname** and a domain name, so the process can set it to something else without affecting the rest of the system. For example, in a container, the Unix Time Sharing (UTS) **namespaces** feature allows each container to have its own **cgroups** and domain name.
- **user**: Isolation of identifiers and attributes are achieved in the user namespace. The user’s process and group identifiers can be different inside and outside a user namespace. This permits a process with full privileges for operations inside the user namespace but makes it unprivileged for operations outside the namespace. For example, a root user within a container might have a *UID* = 0 but on the host system might have a non-privileged user identifier.
- **Control Groups cgroups**¹⁴ are kernel mechanisms introduced by Google in 2007 to provide fine-grained control by grouping processes and their children into a tree structure for resource management. Each group can be assigned a task for CPU, memory, disk and network. For example, to isolate two groups such as applications resources and OS resources, two groups (group 1 and 2) can be created to assign resource profiles individually.

A **cgroup** assigns a set of tasks with a set of parameters for subsystems. A subsystem can be considered to be a module that makes use of the task grouping facilities provided by **cgroup** to handle groups of tasks in the desired manner. A subsystem acts as a “resource controller” within the process hierarchy. This feature provides access (which devices can be used per **cgroup**), limits (**memory**, CPU, device accessibility, block I/O, prioritization (who gets more of the CPU, memory), accounting (resource usage per **cgroup**) and control (freezing and checkpointing).

- **Linux Security Module (LSM)** kernel modules provide a framework for mandatory access control security implementations, where the administra-

¹⁴<https://www.kernel.org/doc/Documentation/cgroups/>

tor (user or process) assigns access controls to subject/initiator. In Discretionary Access Control (DAC), the resource owner (user) assigns access controls to individual resources. Existing LSM implementations include AppArmor, SELinux, Grsecurity¹⁵, and so forth to prevent virtual machines from attacking other virtual machines or the host. For this purpose, policies are used to define what actions a process can perform on a particular system.

- **Containers** are built on the hardware and OS but they make use of kernel features called `chroots`, `cgroups` and `namespaces` to construct a contained environment without the need for a hypervisor. The most recent container technologies are Solaris Zones, OpenVZ, and LXC.

In 2004, Solaris 10 used *zones* as facilities to provide protected virtualized environments within a single host. Every Solaris system includes a global zone for both system and system-wide administrative control, and may have one or more non-global zones. All processes run in the global zone if there is no non-global zone. The global zone is aware of all devices and all file systems while non-global zones are not aware of the existence of any other zones. Zone-based containers provide isolation, security, and virtualization. Zones are similar to jails with additional features such as snapshots and cloning that make it possible to clone efficiently or to duplicate a current zone into a new zone.

In 2005 OpenVZ¹⁶ containers were introduced using a modified Linux kernel with a set of extensions. OpenVZ is based on the `namespaces` and `cgroup` concepts in contrast to Jails, which were used in FreeBSD¹⁷.

Later in 2008, LXC¹⁸ emerged as a container management tool and it combined `namespaces` and control groups to create a fully isolated environment. It provides libraries and command-line support to enable administrators to create new containers. LXC containers can be used in either privileged (as a root user) or unprivileged (as a non-root user) modes to easily customize kernel capabilities or configure `cgroups` to satisfy the particular requirements.

Docker is another container management tool - it was introduced in 2013 and is based on `namespaces`, `cgroups` and SELinux. Docker provides automation for the deployment of containers through remote Application Programming Interface (API)s and has additional features that make it possible to create standardized environments for developing applications. This has made Docker a popular technology. Creating the standardized environments is achieved using a layered image format that enables users to add or remove applications

¹⁵<https://en.wikibooks.org/wiki/Grsecurity>

¹⁶<https://openvz.org/>

¹⁷<http://www.cybera.ca/news-and-events/tech-radar/contain-your-enthusiasm-part-two-jails-zones-openvz-and-lxc/>

¹⁸<https://linuxcontainers.org/>

and their dependencies within a trusted image. Figure 2.4 represents a layered view of Docker to run applications through different APIs.

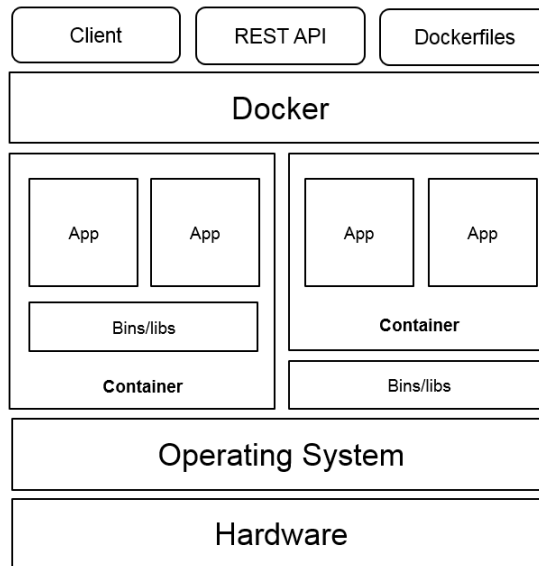


Figure 2.4: Docker architecture (image courtesy of [3])

Docker adds portable deployment of LXC's across different machines. In cloud terms, one can think of LXC as the hypervisor and Docker as both the open virtualization appliance and the provision engine [3]. Docker images can run unchanged on any platform that supports Docker. In Docker, containers can be created from "build" files such as Chef/Puppet¹⁹, Maven²⁰ and there are both command-line or Representational State Transfer (REST) API available for interacting with Docker.

Rocket²¹ is another alternative to Docker that was introduced in 2014. It utilizes *systemd* for creating containers and also has an image management mechanism. Rocket is used to serve in environments with demands for layered security, composability, speed and production requirements. The Rocket software consists of two components (*Actool* and *Rkt*), each with stand-alone command-line support. *Actool* is used for launching containers and includes facilities for container validation and discovery. *Rkt* deals with fetching and running container images.

¹⁹www.chef.io

²⁰<https://maven.apache.org/>

²¹<https://coreos.com/blog/rocket/>

The momentum of container technology with the capability of application portability by encapsulating application within containers has become very interesting for cloud providers. However, integrity and isolation of the container instances within a host still are major security concerns.

The concepts of cloud computing, virtualization, containers are used to design and implement secure systems such as proposed solutions throughout this thesis. We mainly use virtualization and container technologies in chapters 9 and 10.

2.3 Security Techniques to Ensure Privacy

This section describes the EU DPD as one of the most prominent data protection directives that are widely used by the EU Member States. We build a cloud platform that processes genomic data according to the EU DPD requirements to ensure the privacy of sensitive information, as described in Chapter 6. This platform utilizes two-factor authentication and PolyPasswordHasher (PPH) in the BioBankCloud security framework. In Chapter 7 and Chapter 8 we use X.509 certificates and anonymization to ensure confidentiality and integrity of sensitive data through mutual and strong authentication.

2.3.1 The EU DPD Key Concepts

We use the DPD as one of the most widely adopted privacy regulations that have been deployed in many countries, to build proof-of-concept for privacy-preserving cloud systems.

The term “sensitive data” that is used in the DPD may fit into various classifications based on the legal requirements and use case. Some personal data elements are considered more sensitive than others. The definition of what is considered sensitive may vary depending on jurisdiction and particular regulations. Sensitive data require more privacy and security when it comes to collection, processing and disclosure.

The *Personal Data* refers to any Personally identifiable information (PII) directly or indirectly, in particular by reference to an identification number or to one or more factors specific to his physical, physiological, mental, economic, cultural or social identity (Article 2.a). The DPD aims to protect the rights and freedoms of persons with respect to the processing of personal data by laying down the key criteria for making processing lawful and the principles of data quality.

The DPD applies to data processed by automated tools such as a digital database of patients and data stored in or intended to be part of non-automated filing systems such as simple paper documents. It does not apply to the processing of data by a natural person in the context of personal or household activities or in the context of an activity which relates out of the scope of Community law, such as issues concerning State or public security and defense.

The main entities (participants) of the DPD are Data Subject, Controller, and Processor:

- **Data Subject:** Article 2.a of the DPD defines data subject as an identifiable individual associated with the personal data.
- **Controller:** In Article 2.d, the controller is defined as the natural or legal person, public authority, agency or any other body which alone or jointly with others determines the purpose and means of the processing of personal data.
- **Processor:** The processor acts as the natural or legal person, public authority, agency or any other body which processes personal data on behalf of the controller, as defined in Article 2.e. [8].

The obligation of the Controller of personal data to ensure that personal data are adequately secured is one of the fundamental principles of EU DPD (Article 17 of the DPD). Data Controller is the responsible authority to decide about the transmission of data to non-Member States. Transfers of personal data from a Member State to a third country with an adequate level of protection are authorized. There are scenarios where transfers may not have an adequate level of protection. There are exceptions to this rule, i.e., the data subject agrees to the transfer, in the event of deploying a contract or if Binding Corporate Rules (BCR) or Standard Contractual Clauses have been approved by the Member States.

2.3.2 Authentication

The proposed authentication systems in the next chapters are implemented based on the One Time Password (OTP) and X.509 public key certificates.

OTP is a password that can be only used once for login or performing a transaction. Compared to regular passwords that are static, OTPs provides additional security due to avoiding the vulnerabilities of regular passwords caused by replay attacks or credentials theft. OTPs are considered as two-factor authentication where one factor is what user knows and another factor is what user possesses. Since OTPs change after usage, it is difficult to remember them.

The OTP-based authentication can be divided into two subtypes: HMAC-based One-time Password (HOTP) that relies on a counting events and Time-based One-time Password (TOTP) using timing events. Both these approaches are governed by the Open Authentication (OATH) Initiative. In the followings, we will describe the differences between these algorithms that are utilized in the BioBankCloud two-factor authentication system.

2.3.2.1 HMAC-based One-time Password (HOTP)

HOTP is an event-based algorithm that relies on the transformation of a shared secret and an event count (the moving factor) that is synchronized between the server and the client. HOTP was published by Internet Engineering Task Force (IETF) Request for Comments (RFC) 4226 [66]. Keyed-Hashing for Message Authentication (HMAC) includes a hash function internally (HMAC-SHA-1) to hash

the moving counter factor using the secret key. Then, these bytes are taken modulo 10^n , where n is the number of digits in the passcode. The counter factor value increases monolithically. Each client has a unique shared secret, typically 128-bit or 160-bit in length that is generated randomly. Algorithm 1 describes the steps to generate an OTP.

Algorithm 1 The HMAC-based One-time Password (HOTP) algorithm

Require: K be a shared secret between client and server

Require: C be a 8-byte counter value (the moving factor)

Require: T be throttling parameter:

Ensure: $\text{HOTP}(K,C) = \text{Truncate}(\text{HMAC-SHA-1}(K,C))$

Let $HS = \text{HMAC-SHA-1}(K,C)$ // Generate an HMAC-SHA-1 value

Let $Sbits = \text{DT}(HS)$ // Generate a 4-byte string (Dynamic Truncation)

Let $Snum = \text{StToNum}(Sbits)$ // Convert S to a number $0 \dots 2^{31} - 1$

Let $D = Snum \bmod 10^{Digit}$ // D is a number in the range $0 \dots 10^{Digit} - 1$

2.3.2.2 Time-based One-time Password (TOTP)

TOTP is a time-based algorithm that relies on the transmission of a shared secret and a time value that is synchronized between the server and the client. TOTP is adopted as IETF RFC 6238 [67]. TOTP also uses HMAC to combine a secret key with the current timestamp using a cryptographic hash function to generate a one-time password. The timestamp typically increases in 30-second intervals, so passwords generated close together in time from the same secret key will be equal. Algorithm 2 describes the steps to generate an OTP.

Algorithm 2 The Time-based One-time Password (TOTP) algorithm

Require: K be a shared secret between client and server

Require: T_0 is the Unix time to start counting time steps

Require: $X = 30$ //represents the time step in seconds

Ensure: $\text{TOTP} = \text{HOTP}(K, T/X)$

$T = (\text{Current Unix time} - T_0)$

Let $HS = \text{HMAC-SHA-1}(K,T/X)$ // Generate an HMAC-SHA-1 value

Let $Sbits = \text{DT}(HS)$ // Generate a 4-byte string (Dynamic Truncation)

Let $Snum = \text{StToNum}(Sbits)$ // Convert S to a number $0 \dots 2^{31} - 1$

Let $D = Snum \bmod 10^{Digit}$ // D is a number in the range $0 \dots 10^{Digit} - 1$

2.3.2.3 X.509 Public Key Certificates

The X.509 public key certificate provides scalable secure communication in open networks based on an asymmetric pair of keys known as public (shared with all

parties) and private (owned only by the user). It provides two main features: a digital signature to sign a document and data encryption between two participants. The usage of the keys is defined through the certificate policy along the validation of the private key and by the public key and certificate revocation list. A certification authority distributes such policy. There are several public certificate types that contain additional information such as attributes but the most common are X.509 version 3.0 which is built on trust between the issuing certification authority of the certificate and the public key users.

2.3.3 Data Anonymization Techniques

Researchers need to access valuable medical microdata that is accumulated over time to perform experiments and analysis to get insight into diseases. Microdata is a series of unaggregated records containing information of an individual such as a person or an institution. The final recipients of such specific data are able to perform analytical tasks compared to macrodata that is in precomputed statistical formats. Organizations that have collected such data are subject to legal requirements to prevent the identification of data subjects while allowing data to be analyzed and used.

Data anonymization as a sanitation technique can help to build privacy-reserving cloud solutions to increase security, either by encrypting or removing PII such as name, social security number, telephone number and email of microdata. In contrast, with access control mechanisms that protect the information from unauthorized access, data anonymization controls disclosure of private data to publish useful information. This will reduce the risk of re-identification in published data through removing attributes that clearly identify data subjects by removing or replacing them with random values.

Despite removing the direct identifiers from the disclosed information that produce de-identified micro datasets, there are scenarios where an adversary can join the published databases through background knowledge with external databases on attributes such as gender, date of birth, and Zone Improvement Plan (ZIP) code to re-identify individuals who are required by data protection legislation to remain private. This re-identification attack is pointed out by Sweeney [68] that naive removing of PII such as name and Social Security Number (SSN) leaves possibility of open attacks by combining the microdata with other publicly available information.

A well-known example is provided by the report [4] that describes the combination of only on {5-digit ZIP code, gender, date of birth} can result in unique re-identification of approximately 87% (216 million of 248 million) of the population in the United States. Figure 2.5 shows how Sweeney could re-identify individuals through QIDs over two public databases **data source I** and *data source II*. Data source I represents the voters lists including name, address, ZIP code, date of birth (DoB), and gender of each voter which is publicly available. Data source II represents the de-identified medical data with no explicit identifier such as name or address.

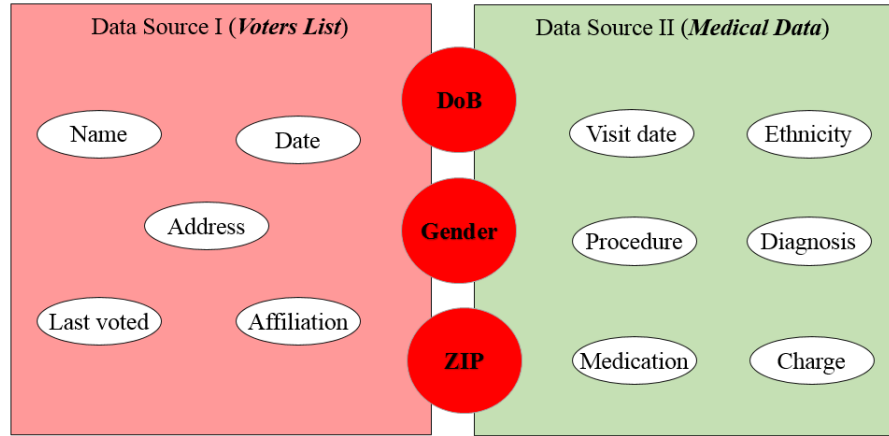


Figure 2.5: Linking to re-identify data (image courtesy of [4])

In literature, variable values or combination of such attributes that are not structurally unique but can result in re-identification of a population unit (a data subject) is called as a QID [69]. To avoid such linking attacks via QIDs, k -Anonymity was proposed by Sweeney and Samarati [68, 70, 71] as a formal model of privacy. The goal is to make each record in the microdata sets indistinguishable from a (k) number of other records, i.e., in scenarios where an adversary attempts to reidentify the data subjects.

2.3.3.1 k -Anonymity

Assume table \mathbf{T} with tuples $\{t_1, t_2, \dots, t_n\}$ over an attribute set $\mathbf{A} \{a_1, a_2, \dots, a_n\}$. Table \mathbf{T} satisfies k -anonymity, if, for any tuple with a given set of attributes in \mathbf{A} , there are at least $(k-1)$ other tuples that equal those attributes. Each tuple is associated with a key that represents a unique identifier which is required to be removed or obscured.

For example, table 2.3 shows a private patient dataset that is recorded in a hospital. The table contains ZIP code and age as QID attributes. The disease attribute is the actual sensitive attribute that must remain private despite disclosing the dataset.

Table 2.4 shows a 4-anonymous dataset over the QID (ZIP code, age) in the original patient data (table 2.3). The 4-anonymity property is satisfied because the minimum equivalence group size on the QID for is at least 4 for the anonymized groups in the table (two groups with sizes of 4 and 5).

To achieve higher levels of privacy, the value of (k) should be increased. Because the probability of linking a data subject to a specific tuple through QID is at most

ZIP code	Age	Disease
17601	31	Cancer
17601	32	BRCA Mutation
17605	33	Cancer
17605	34	Alzheimer
13059	36	Cancer
13056	38	Cancer
13054	37	Viral Infection
13055	38	Viral Infection
13059	39	Viral Infection

Table 2.3: Raw private patient dataset without anonymization

ZIP code	Age	Disease
176**	[30,35)	Cancer
176**	[30,35)	BRCA Mutation
176**	[30,35)	Cancer
176**	[30,35)	Alzheimer
1305*	[35,40)	Cancer
1305*	[35,40)	Cancer
1305*	[35,40)	Viral Infection
1305*	[35,40)	Viral Infection
1305*	[35,40)	Viral Infection

Table 2.4: A sample patient dataset with k -anonymity, where $k=4$

$1/k$. The properties of k -anonymity for various attributes are summarized in Table 2.5.

Attribute	Description	Example
Key	Uniquely identifiers of data subjects	Name, SSN and address
QID	Combination of attributes that can be linked with external information to re-identify data subjects	ZIP code, birthday and gender
Sensitive	Data that a data subject is sensitive about revealing	Disease, ethnicity and salary

Table 2.5: k -anonymity description of attributes to prevent record linkage through QID

Even k -anonymity provides an effective approach for anonymization due to its simplicity and algorithmic supports, it is still vulnerable mainly using homogeneity or background knowledge attacks. Because in k -anonymity model sensitive at-

tributes that exist in the equivalence class lack diversity, which leaves it open to adversarial attacks.

A homogeneity attack can be launched due to the fact that the values for a sensitive attribute within a set of the k tuple are identical. Despite k -anonymization over a dataset, values of sensitive attributes might be predictable. For example in the 4-anonymous table, Eve (an attacker) can make a prediction over Alice (a patient) that lives in a specific area and her age that is known to Eve. Since Eve knows Alice’s age and that Alice is present in the dataset, then she will know Alice has cancer.

Background knowledge attack is another approach that an attacker is able to make the association between one or more QID with the sensitive attributes through reducing the sensitive attribute value range. Machanavajjhala et al. [72] in 2007 showed that the knowledge of common diseases in a specific geographical area could be used to narrow the range of values for a sensitive attribute of a patient dataset. For example, if Eve the attacker knows Alice (the patient) is in the dataset and since young women are not likely to have cancer at the age of 30 then Eve can conclude Alice has a viral infection.

2.3.3.2 ℓ -Diversity

ℓ -diversity is another privacy model proposed by Machanavajjhala et al. [72] to diversify the sensitive attributes in k -anonymity model. The goal of ℓ -diversity is to prevent attribute linkage through ensuring that each QID group contains at least ℓ “well-represented” sensitive values. ℓ -diversity improves anonymization beyond k -anonymity and it makes it possible to counter both of homogeneity and background knowledge attacks.

Table 2.6 shows a ℓ -diversity, where $\ell=2$ wherein the first group a factious value has been replacing with the Alzheimer disease to satisfy the diversity constraint. This replacement causes in the loss of quality of data and it should be compensated during analysis.

ZIP code	Age	Disease
176**	[30,35)	Cancer
176**	[30,35)	BRCA Mutation
176**	[30,35)	Cancer
176**	[30,35)	BRCA Mutation
1305*	[35,40)	Cancer
1305*	[35,40)	Cancer
1305*	[35,40)	Viral Infection
1305*	[35,40)	Viral Infection
1305*	[35,40)	Viral Infection

Table 2.6: A sample patient dataset with ℓ -diversity, where $\ell=2$

Similar to k -anonymity that prevents record linkage, ℓ -diversity also assumes the attacker's knowledge of the presence of a data subject in the dataset. But ℓ -diversity ensures the attribute linkage over sensitive values.

2.3.4 Secret Sharing

Secret sharing is one of the main building blocks in the modern cryptography that provides efficient mechanisms for many applications and protocols for distributing a secret among n parties. The original secret can be reconstructed only if a specific number of shares are present. Secret sharing schemes can be utilized for many sensitive operations and applications that require high levels of security. The idea of secret sharing originates from the Liu's book on combinatorial mathematics [73] that discussed the following problem:

"Eleven scientists are working on a secret project. They wish to lockup the documents in a cabinet so that the cabinet can be opened if and only if six or more of the scientists are present. What is the smallest number of locks needed? What is the smallest number of keys to the locks each scientist must carry?"

Prior to formalizing the definition of threshold secret sharing, there have been several informal definitions. Most notably, t -out-of- n threshold scheme, where the need for having trusted parties to access the shared secret is relaxed. This scheme has the following properties.

- At least, t (the threshold) parties are needed to recover the secret (privacy level).
- There are n parties in total, where $(n - t)$ is the redundancy level.

The t -out-of- n threshold scheme can provide maximum privacy in terms of perfect privacy or perfect secret sharing. For this purpose, $(t - 1)$ parties have no knowledge of anything related to the secret from their $(t - 1)$ shares. This is equivalent to storing n shares in a safe over different banks. The value of t can be increased for scenarios with little trust to the bank managers (key repositories).

Secret sharing can be verifiable in the presence of active adversaries. For example, if a dealer (distributor) sends inaccurate shares instead of a correct share. In such scenario, reconstruction of the original secret collapses. To solve this issue, participants can verify that they received shares of the same secret, to guarantee, that any honest party can recompute the same secret k .

Secret sharing can be used by several real-life applications that suffer from weaknesses originating from vulnerabilities in computer systems. For example untrusted communication, key escrow and secure multi-party computation can benefit from the secret sharing scheme. Communication network vulnerabilities include channel destruction, eavesdropping, and altering the contents in transit. Key escrow is a special case of secret sharing through trusted government or industry agencies that still demands a trusted third party. Secure multi-party computation is based on

the idea of zero knowledge proof that allows n parties to compute a function of private inputs without leaking more than required knowledge about the inputs.

Threshold cryptography is another application of secret sharing that requires decrypting an encrypted message by k parties, where k is larger than the number of participants. For example, a message can be encrypted using a public key and the corresponding private key is shared among the different parties.

In 1979, Shamir [74] proposed a secret sharing solution based on the threshold cryptography and Lagrange interpolation polynomial semantics. Shamir proposed this approach to solving the issue of reconstructing a share where the presence of all required shares is impractical. Therefore, the (k, n) -threshold scheme is used where any k (threshold) of the n parts are sufficient to reconstruct the original secret, otherwise no information about the secret is provided.

Suppose S is a secret that is required to be hidden using the (k, n) -threshold scheme where $k < n$. Shamir Secret Sharing produces a $k - 1$ random coefficients for a $k - 1$ degree polynomial $f(x)$. The k th element is the secret that is supposed to remain hidden.

A value between 1 and $k - 1$ is chosen to compute a share. The polynomial is constructed with x equal to the share value, where x and $f(x)$ represents the share. To reconstruct the secret from at least k shares, a shareholder can interpolate the values to compute the secret.

The following example illustrates the algorithm. Suppose that a secret, “1234” is required to be hidden. The secret “1234” can only be reconstructed if three shares are presented by different parties. Because the threshold is three, two random values (94 and 166) are generated as the first step to represent a polynomial, such as $f(x) = 94x^2 + 166x + 1234$. Then other shares are generated by computing x and $f(x)$. For example (1, 1494), (2, 1942), (3, 2598), (4, 3402), (5, 4414), and (6, 5614). A party that has, at least, three shares can interpolate to reconstruct the full polynomial of $f(x)$ and knows the secret “1234”.

PPH [75] is a password protection solution based on the Shamir secret sharing and it provides additional levels of security to make cracking users passwords very hard in a database. PPH proposes a hybrid solution including cryptographic hashing and threshold cryptography to combine password hash data with shares so that users credentials are used to protect the database safely. It also combines a share that is derived using a threshold cryptography system with a salted password hash. The result then will be stored as a combined value in the password database. The key idea in PPH is to not storing the share nor the password hash on disk. This way, an attacker cannot recover either piece with only the password database information. To crack a password that is created by the PPH, an adversary requires knowing a threshold of passwords.

The difference between PPH and salted hash is that a salted hash database stores a username, salt and a salted hash, while a PPH database, stores the secure hash XORed with the related share. The resulting PPH database also stores an extra field called the share number that demonstrates which share was XORed with which salted hash. Upon supplying validate passwords, the server XORs the salted

hash with the stored data and concludes whether the result is a valid share of the threshold cryptography system.

2.4 Summary

This chapter provided a brief overview of big data and cloud computing concepts that are used throughout this dissertation. It introduced the big data infrastructure technologies such as Hadoop ecosystem and its evolving building blocks. Summarized definition of cloud computing concepts and virtualization mechanisms that are used in chapters 4, 5, 6, 8, and 9 were presented. We also discussed the widely used EU DPD privacy regulation to be enforced through security measures such as authentication, anonymization and secret sharing in chapters 5, 6, 7. We will identify the state-of-the-art related to this research in the next chapter.

Chapter 3

Related Work

This chapter is mainly based on publications II and IV. It reviews related research on security and privacy for the cloud within two aspects. First, it identifies the related research on different elements of cloud provider activities. Second, it reviews the existing developments in the area of privacy-preservation for sensitive data in cloud computing.

3.1 Identification of Research

There has been numerous research on security and privacy of sensitive data in cloud computing in both industry and academia [76, 77, 78, 79, 80, 81]. Most notably developing protocols and tools for anonymization or encryption of data for confidentiality purposes.

To identify the related research on security and privacy of sensitive data in the cloud, we categorized the activities of cloud providers into five main categories: service deployment, resource abstraction, physical resources, service management, security, and privacy [2]. Service deployment consists of delivering services to cloud consumers according to one of the service models (SaaS, PaaS, IaaS). Resource abstraction refers to providing interfaces for interacting with networking, storage and compute resources. The physical resources layer includes the physical hardware and facilities that are accessible via the resource abstraction layer. Service management includes providing business support, resource provisioning, configuration management, portability and interoperability to other cloud providers or brokers. The security and privacy responsibilities of cloud providers include integrating solutions to ensure legitimate delivery of cloud services to the cloud consumers. The security and privacy features that are necessary for the activities of cloud providers are described in Table 3.1 [7].

Security Context	Description
Authentication and Authorization	Authentication and authorization of cloud consumers using pre-defined identification schemes
Identity and Access Management	Cloud consumer provisioning and provisioning via heterogeneous cloud service providers
Confidentiality, Integrity and Availability	Assuring the confidentiality of the data objects, authorizing data modifications and ensuring that resources are available when needed
Security Monitoring and Incident Response	Continuous monitoring of the cloud infrastructure to ensure compliance with consumer security policies and auditing requirements
Security Policy Management	Defining and enforcing rules to enforce certain actions such as auditing and proof of compliance
Privacy	Protect PII within the cloud from adversarial attacks that aim to find out the identity of the person that the PII relates to

Table 3.1: Security and privacy factors of cloud providers [7]

3.2 Cloud Security

This section reviews the research on security solution such as authentication, authorization, and identity management that were identified in Table 3.1 [2] as being necessary so that the activities of cloud providers are sufficiently secure.

3.2.1 Authentication and Authorization

Multifactor authentication is a mechanism that requires more than one factor to verify the identity of users to achieve strong authentication. For example, smart cards and tokens [82] or two-factor authentication [83] through popular services such as Short Message Service (SMS) or direct phone calls are multi-factor authentication mechanisms that effectively protect classified information. We use two-factor authentication using mobile devices and Yubikey tokens because setting up the infrastructure for smart cards, phone calls or SMS based authentication can become costly to maintain.

Public Key Infrastructure (PKI) is another strong authentication approach that provides scalable secure communication solutions in open networks based on an asymmetric pair of keys known as public (shared with all the parties) and private (owned only by the user) keys [84], [85]. There is a usability issue with the PKI certificate authentication, since non-IT users might have difficulties understanding how to keep public/private keys secret and yet available on a computer for login. FermiCloud [86] uses this approach for authentication and authorization - it utilizes PKI X.509 certificates for user identification and authentication. FermiCloud is built

in OpenNebula¹ and it develops both X.509 authentication in Sunstone OpenNebula - a Web interface intended for user management - and X.509 authentication via command-line interfaces. To avoid the limitations of OpenNebula access control lists that are used for authorization after successful authentication of users, authors integrated an existing local credential mapping service. This solution has also been extended in cloud federations to authorize users across different cloud providers that have established trust relationships through trusted certification authorities.

Biometrics authentication is another evolving field for secure password authentication [87], [88]. This method offers authentication based on the measurement of unique physiological characteristics of a user, such as fingerprints (to replace passwords), face recognition, iris codes and behavioral characteristics.

In [89], Gonzalez et al. propose a credential classification and a framework for analyzing and developing solutions for credential management that include strategies to evaluate the complexity of cloud ecosystems. This study identifies a set of categories relevant for authentication and authorization for the cloud focusing on the infrastructural aspects which include classifications for credentials and adapt those categories to the cloud context. The study also summarizes important factors that need to be taken into consideration when adopting or developing a solution for authentication and authorization - for example, identifying the appropriate requirements, categories, services, deployment models, lifecycle, and entities. In other work, a design model for multi-factor authentication in cloud computing environments is proposed in [90] by Banyal et al., and this model includes an analysis of the potential security threats in the proposed model. Another authentication solution is seen with MiLAMob [91], which provides a SaaS authentication middleware for mobile consumers of SaaS cloud applications. MiLAMob is a middleware layer that handles the real-time authentication events on behalf of consumer devices with minimal Hypertext Transfer Protocol (HTTP) traffic. The middleware currently supports mobile consumption of data on IaaS clouds such as Amazon's Simple Storage Service (S3). Different to these approaches, we provide authentication in IaaS and SaaS layers.

Tang et al. [92] introduce collaborative access control properties such as centralized facilities, agility, homogeneity, and outsourcing trust. They have introduced an authorization-as-a-service (AaaS) an approach using a formalized multi-tenancy authorization system, and providing administrative control over enhanced fine-grained trust models. Integrating trust with cryptographic RBAC is another solution [93] by Ferraiolo et al. that ensures trust for secure sharing of data in the cloud. The authors propose using cryptographic RBAC to enforce authorization policies regarding the trustworthiness of roles that are evaluated by the data owner [94]. Another feature of the authorization system in this solution is that it develops a new concept using role inheritance for evaluating the trustworthiness of the system. In another study, Sendo et al. [95] propose a user-centric approach for platform-level authorization of cloud services using the OAuth2 protocol to allow services to

¹<http://opennebula.org/>

act on behalf of users when interacting with other services in order to avoid sharing usernames and passwords across services.

Similar to these approaches, we use PKI X.509 certificates for mutual authentication in the proposed architectures in Chapter 7 and Chapter 8, in addition for fine-grained encryption of clinical data in Chapter 7. However, the above authentication and authorization mechanisms have been designed for specific use cases which make them hard to fit other models such as BioBankCloud that aims to satisfy slightly different architectural goals. Or other technical limitation such as flexibility of web servers to offer several authentication methods in parallel to provide a higher degree of usability.

3.2.2 Identity and Access Management

The important functionalities of identity management systems for the success of clouds in relation to consumer satisfaction are discussed by Leonardo et al. in [96]. The authors also present an authorization system for cloud federation using Shibboleth - an open source implementation of the Security Assertion Markup Language (SAML) for single sign-on with different cloud providers. This solution demonstrates how organizations can outsource authentication and authorization to third-party clouds using an identity management system. Stihler et al. [97] also propose an integral federated identity management for cloud computing. A trust relationship between a given user and SaaS domains is required so that SaaS users can access the application and resources that are provided. In a PaaS domain, there is an interceptor that acts as a proxy to accept the user's requests and execute them. The interceptor interacts with the STS, and requests the security token using the WS-Trust specification.

IBHMCC [98] introduced by Li et al. provides Identity-Based Encryption (IBE) and Identity-Based Signature (IBS) schemes. Based on the IBE and IBS schemes, an identity-based authentication for cloud computing has been proposed. The idea is based on the identity-based hierarchical model for cloud computing along with the corresponding encryption and signature schemes without using certificates for simplified key management.

Contrail [99] is another approach that aims to enhance integration among heterogeneous clouds both vertically and horizontally. Vertical integration provides a unified platform for the different kinds of resources while horizontal integration abstracts the interaction models of different cloud providers. In Contrail, Carlini et al. [99] develop a horizontal federation scheme as a requirement for vertical integration. The proposed federation architecture contains several layers, such as users' identities, business logic, and a federation manager to support APIs for resources, storage, and networking across different providers.

E-ID authentication and uniform access to cloud storage service providers [100] is an effort by Gouveia et al. to build identity management systems for authenticating Portuguese citizens using national e-identification cards for cloud storage systems. In this approach, the OAuth protocol is integrated for authorizing the

cloud users. The e-ID cards contain PKI certificates that are signed by several levels of governmental departments. A certification authority is responsible for issuing the e-ID cards and verifying them. The e-ID cards enable users for identity-based encryption of data in cloud storage.

Hummer et al. introduce a model-driven specification and enforcement of task-based entailment constraints in distributed service-based business processes [101], where the authors describe challenges of secure sharing of data using RBAC. Identity and access management tasks are enforced using Web services where SAML tokens are used for authentication users across various identity providers.

In [102], Dreo et al. consider the issues related to the inter-cloud federation and the proposed ICEMAN identity management architecture. ICEMAN discusses identity life cycle, self-service, key management, provisioning and deprovisioning functionalities that need to be included in an appropriate intercloud identity management system.

In [103], Sipos et al. discuss delivery of a hybrid federated cloud as a collaboration of communities developing, innovating, operating and using clouds for research and education that is called European Grid Infrastructure (EGI). The EGI federated cloud provides IaaS, persistent block storage attached to VMs, and object-level storage for transparent data sharing. The EGI controls access to resources using X.509 certificates and the concept of Virtual Organization (VO). VO refers to a dynamic set of users or institutions using resource-sharing rules and conditions. The authorization attributes are issued through a VO management system that can be integrated with SAML for federation.

However, these solutions mainly use Shibboleth (an open source implementation of SAML) to outsource the authentication to a third-party or use PKI for authentication. Both these approaches have usability issues in our solutions. First, Shibboleth does not support Glassfish integration [104]. Second, PKI as we will discuss in the later chapters, PKI non-IT users have difficulty to use PKI infrastructure in addition to its maintainability.

3.2.3 Confidentiality, Integrity and Availability (CIA)

This section summarizes other virtualization approaches related to our work in Chapters 9 and Chapter 10 that aim to ensure the safety of privileged code in userspace and kernelspace through Confidentiality, Integrity, Availability (CIA) properties.

Santos et al. [105] extend the Terra [106] design that enables users to verify the integrity of VMs in the cloud. The proposed solution is called the Trusted Cloud Computing Platform (TCCP), and the whole IaaS is considered to be a single system instead of granular hosts in Terra. In this approach, all nodes run a trusted virtual machine monitor to isolate and protect virtual machines. Users are given access to cloud services through the cloud manager component. The External Trusted Entity (ETE) is another component that provides a trust coordinator service in order to keep track of the trusted VMs in a cluster. The ETE can be

used to attest the security of the VMs. A TCCP guarantees confidentiality and integrity in data and computation and it also enables users to attest to the cloud service provider to ensure whether the services are secure prior to setting up their VMs. These features are based on the Trusted Platform Module (TPM) chip. The TPM contains a private endorsement key that uniquely identifies the TPM and some cryptographic functions that cannot be altered.

In 2011, Popa et al. [107] proposed CloudProof as a secure storage system to guarantee confidentiality, integrity and write-serializability using verifiable proofs of violation by external third parties. Confidentiality is ensured by private keys that are known only to the owner of the data that is to be encrypted. The main idea behind CloudProof is the use of the attestation mechanism. Attestations provide proof of sanity of users, data owners, and cloud service providers. Data owners use a block identifier to acquire the content of a block. This mechanism enables users to store data by putting a block identifier and the contents of the block in the cloud. The attestation structure implements a solution called “block hash” for performing integrity checks through signature verification. The block hash provides proof for write-serializable using a forked sequence of the attestations while a chain hash is used for a broken chain of attestations which are not sequenced correctly.

Fuzzy authorization for cloud storage [108] is another flexible and scalable approach to enable data to be shared securely among cloud participants. Fuzzy authorization ensures confidentiality, integrity, and secure access control by utilizing secret sharing schemes for users with smartphones who are using the cloud services.

In [109], Perez-Botero et al. define threats to the cloud server hypervisors by analyzing the codebase of two popular open-source hypervisors: Xen and KVM. In addition, they discuss the vulnerabilities reports associated with them. As a result, a model is proposed for characterization of hypervisor vulnerabilities in three dimensions: the trigger source, the attack vector, and the attack target. The attack vector consists of the Hypervisor functionality that makes security breaches possible - for example, virtual CPUs, symmetric multiprocessing, soft memory management units, interrupt and timer mechanisms, I/O and networking, para-virtualized IO, VM exits, hyper call, VM management (configure, start, pause and stop VMs), remote management, and software hypervisor add-ons. Successful exploitation of a vulnerability in these functionalities enables an attacker to compromise the confidentiality, integrity, or availability of the Hypervisor or one of its guest VMs.

The vulnerability reports in [109] show 59 vulnerability cases for Xen and 38 cases for KVM. Approximately, 50 percent of all vulnerabilities are common regarding confidentiality, integrity, and availability. The remote management software of Xen contributes to 15.3 percent of the vulnerabilities that demonstrates the increase attack surface by non-essential services. VM management component contains 11.9 percent of the vulnerabilities in Xen compared to 5.3 percent in KVM. The lower vulnerability rate in KVM is due to the *libvirt* toolkit inside the hypervisor, where Xen allocates an entire privileged area in Dom0. Other dimensions that have been studied in [109] are trigger sources and attack targets, including network, guest VM’s userspace, guest VM’s kernel-space, Dom0/host OS, and hypervisor. The

most common trigger source is the guest VM userspace contributing to 39.0 percent of Xen's and 34.2 percent of KVM's vulnerabilities. This makes it possible for any user-space guest VM to be a threat to the hypervisor. The guest VM kernel-space is in the second place with around 32 percent of the total in both cases. Finally, the results show that Dom0 to be a more common target than the hypervisor in Xen. While in KVM the host OS is less common target compared to the hypervisor. The location of the I/O device emulation backend drivers plays an important factor in this difference. The I/O and network device emulation functionalities cause one-third of the 15 vulnerabilities in both Xen and KVM hypervisors.

In [110] Brassler et al. propose Swap and Play as a new approach for live updating of hypervisors without the need to reboot the VM for high availability. The proposed design is scalable, usable and applicable in cloud environments and it has been implemented in Xen as one of the most popular hypervisors. Swap and Play provide methods for transferring the in-memory state of the running hypervisor to the updating state, in addition to updating the underlying host. Swap and Play consists of three independent phases: preparation, distribution and update. In the preparation phase, information for the later state transfer is collected. The distribution phase deploys the update package on the target host for updating. In the last step, the update package is patched to individual hosts in the cloud. Each host applies the update package independently of the others and does not require any network resources. The Xen implementation of the Swap and Play solution is called SwapVisor. SwapVisor introduces a new hyper call in the Xen architecture. A hyper call is a trap from a domain to the hypervisor (similar to a system call from an application to the kernel). Hypercalls are used by domains to request privileged operations such as updating page tables. The experiments show that updating from Xen version 4.2.0 to version 4.2.1 is fulfilled within approximately 45 ms which seems to be intangible and have almost zero effect on the network performance.

Klein et al. [111] improve cloud service resilience using a load-balancing mechanism called brownout. The idea behind this solution is to maximize the optional contents to provide a mechanism that is resilient to volatility in terms of flash crowds and capacity shortages (through load-balancing over replicas) when compared to other approaches that are implemented using response-time or queue length. In another effort [112], Lakew et al. proposed a synchronization mechanism for cloud accounting systems that are distributed. The runtime resource usage generated from different clusters is synchronized to maintain a single cloud-wide view of the data so that a single bill can be created. The authors also proposed a set of accounting system requirements and an evaluation method which verifies that the solution fulfills these requirements.

Language-based virtualization provide safety through virtualization such as Java, JavaScript, Lua [113], and Silverlight [114] are commonly used in application-level sandboxing. These safe languages provide virtualized environments to check the safety of the running code by a monitor process. They combine untrusted application code with an interpreter and standard libraries that consolidate routines to

perform I/O, network communication, and other sensitive functions. For example, the Java Virtual Machine (JVM) [115] functions as an application-level sandbox to separate untrusted code from the OS in addition to performing safety checks for avoiding unauthorized branching in memory.

There are also sandboxing solutions based on type-safety of programming languages, i.e. validating through a type-checker [116] or enforcing security policies on an untrusted system through a reference monitor [117]. Compared to these approaches, Lind (see chapter 9) provides more portability than just executing the code in a safe sandbox browser.

Though many sandboxes implement the bulk of standard libraries in a memory-safe language like Java or C#, flaws in memory-safe code can still pose a threat. In fact, many security critical bugs can be found in the standard libraries [118, 119]. Any bug or failure in a programming language virtual machine is usually fatal. In contrast, Lind with a very small Trusted Code Base (TCB) (approximately 8,000 Line of Code (LOC)) enhances security compared to the above virtual machines.

OS virtualization techniques can be divided into two categories: bare-metal hardware virtualization such as VMware ESX Server, Xen, LXC [120], BSD's jail, Solaris zones, and Hyper-V, and hosted hypervisor virtualization such as VMware Workstation, VMware Server, VirtualPC and the open-source counterpart VirtualBox. Security by isolation [121, 122, 123, 124] is a feature of OS virtualization to provide safe executing environments through containment for multiple user-level virtual environments that share the same hardware. This approach relies on the VMM to confine untrusted applications within guest OSs. However, there are limitations due to the large attack vectors against the hypervisors including vulnerabilities of software and configuration risk. Lind deals with these concerns by using a much smaller and safer TCB.

Library OSeS are useful for applications to efficiently obtain the benefits of virtual machines, including security isolation, host platform compatibility, and migration. Drawbridge [125] uses lightweight processes and a library OS to present a Windows persona to a wide variety of Windows applications. This is accomplished by moving a large portion of the OS into the process, and presenting a simplified system virtual machine-like interface to each process. Bascule [126], an architecture for library OS extensions based on Drawbridge, allows application behavior to be customized by extensions loaded at runtime. Graphene [127] is a recent library OS system that seamlessly and efficiently executes both single OS and multi-process applications, with low memory and performance overheads. Haven [128] uses a library OS to implement shielded execution of unmodified server applications in an untrusted cloud host.

Its library OS technique is similar to Lind but differs in the fact that existing library OS systems rely heavily on the underlying kernels to perform system functions while Lind only relies on a very limited set of system functions, and reconstructs most OS functions with its own safe Repry code.

3.2.4 System Call Interposition:

System call interposition offers a number of properties that make it attractive for building sandboxes though it can be error prone [129]. Approaches for delegation and filtering have been extensively studied. For example,

Janus Version 2 (J2) [130, 131] uses filtering and sandboxing, and Ostia [129] uses a delegation. Ostia also provides a hybrid interposition architecture, which allows for kernel level enforcement and user policies.

However, system call interposition has many problems. OS semantics are very difficult to replicate correctly. Indirect paths to resources are often overlooked, and there are side effects to denying system calls [132].

Nevertheless, this technique is very useful and has inspired many new techniques, such as library OSes. The concepts behind system call interposition have evolved into other modern techniques and has benefited many security systems, including Lind.

Software Fault Isolation (SFI) is an alternative to hardware memory protection for running two applications in one address space by instruction rewriting. SFI provides sandboxing in which native instructions can only be executed if they do not violate the sandbox's constraints [133]. This goal is achieved through machine-level code analysis to enforce security policies. In this approach, memory writes are protected and code jumps cannot access predefined memory of other programs or execute other programs codes in memory.

The preliminary design of SFI approach built on RISC architectures. The authors of PittSFIeld [134] optimized and extended the original SFI to support CISC architectures. For this purpose, the source instructions are padded with no-ops to fit, i.e. in the 16-byte x86 byte chunk alignment, where a call instruction is appended. A sequence of instructions forms instruction streams that ensure execution order of the sequence. The final code before execution will be checked by a verifier component to ensure safety. The authors used machine-checked proof for increased assurance that verifier only approves safe operations.

SFI has been also used in MisFIT [135] to ensure kernel modules integrity for x86. The authors emphasized the extendability of object-oriented programming languages to eliminate the need for remote calls to achieve high throughput sandboxes.

Nooks [136] is another SFI-based solution that provides protected environment for running device drivers by isolating kernel modules and device derives mainly for reliability and fault resistance. Nooks runtime environment is located within the kernel and it includes the majority of drivers that needs to be protected from each other.

Recently, Google provided Google Native Client (NaCl) [137] for Chrome browser to allow native executable code to be run directly in a browser using the PittSFIeld semantics. NaCl prevents suspicious code from memory corruption or direct access to the underlying system resources. For this purpose, NaCl loads untrusted modules from the trusted modules into two different address spaces, wherein most SFI

approaches both untrusted and trusted codes are loaded into a common address space.

3.2.5 Security Monitoring and Incident Response

Anand [138] present a centralized monitoring solution for cloud applications consisting of monitoring the server, monitors, agents, configuration files and notification components. Redundancy, automatic healing, and multi-level notifications are other benefits of the proposed solution which are designed to avoid the typical drawbacks of a centralized monitoring system, such as limited scalability, low performance and single point of failure.

Brinkmann et al. [139] present a scalable distributed monitoring system for clouds using a distributed management tree that covers all the protocol-specific parameters for data collection. Data acquisition is done through specific handler implementations for each infrastructure-level data supplier. Data suppliers provide interoperability with cloud software, virtualization libraries, and OS-level monitoring tools. The authors review the limitations of existing intrusion detection systems and discuss VM-level intrusion detection as an emerging area for securing VMs in cloud environments. The requirements for an efficient intrusion detection system for cloud infrastructures - including multi-tenancy, scalability, and availability - are identified and a VM introspection detection mechanism via a hypervisor is proposed.

Hypervisor-based cloud intrusion detection systems are a new approach (compared to existing host-based and network-based intrusion detection systems) that is discussed in [140]. The idea is to use hypervisor capabilities to improve performance over data residing in a VM. Performance metrics are defined as networking transmitted and received data, read/write over data blocks, and CPU utilization. These metrics are retrieved in near real-time intervals by endpoint agents that are connected directly to a controller that analyzes the collected data using signatures to find any malicious activity. The controller component sends an alert to a notification service in case there is any potential attack.

3.2.6 Security Policy Management

In [141], Basescu et al. propose a generic security management framework allowing providers of cloud data management systems to define and enforce complex security policies through a policy management module. The user activities are stored and monitored for each storage system and are made available to the policy management module. Users' actions are evaluated by a trust management module based on their past activities and are grouped as "fair" or "malicious". An appropriate architecture for security management which satisfies the requirements of policy definitions (such as flexibility, expressiveness, extendibility, and correctness) has been implemented. The authors evaluated the proposed system on a data management system that is built for data storage.

Takabi et al. [142] introduce policy management as a service to provide users with a unified control point for managing access policies in order to control access to cloud resources independently of the physical location of cloud providers. The proposed solution is designed specifically to solve the issue of having multiple access control authorization mechanisms employed by cloud service providers that restrict the flexibility of applying custom access control to a particular service. For this purpose, the architecture includes a policy management service provider that is the entry point for cloud users to define and manage the policies. The cloud service provider imports the user-defined policies and acts a policy decision point to enforce the user policies.

The challenges associated with policy enforcement in heterogeneous distributed environments are discussed in [143]. Hamlen et al. propose a framework to support flexible policy enforcement and a feedback system using rule- and context-based access control to inform cloud users about the effect of defined policies. There are three main requirements for building a general policy enforcement framework. First it must support various data types such as image, structured and textual data. Secondly, in a distributed environment there need to be several compute engines such as Map/Reduce, relational database management systems or clusters. Finally, access policy requirements in terms of access control policies, data sharing policies, and privacy policies need to be integrated with the general policy management framework. Several policy enforcement mechanisms (such as extensible access control markup language or inline reference monitors to enforce user-centric policies in accord with cloud provider approval) were also discussed.

In [144], Pearson et al. describe A4Cloud with the aim of developing solutions to ensure accountability and transparency in cloud environments. Users need to be able to track their data usage to know how the cloud provider satisfies their expectations for data protection. For this purpose cloud providers must employ solutions that provide users with appropriate control and transparency over their data, e.g. tools to define policies for compliance with regulatory frameworks. In another effort [145], Hansen et al. discuss the issue of usable transparent data processing in cloud computing and also consider how to enable users to define transparency policies over their data. They identify the requirements for transparent policy management in the cloud based on two aspects: user demands and legal aspects of transparent data processing.

In this dissertation, we develop transparency mechanisms from both cloud provider and cloud consumer perspectives. Because, we believe to enforce transparent processing of data, it is necessary to involve diverse participants in the development lifecycle.

3.3 Data Security and Privacy

This section outlines several efforts and projects on big data security and privacy including big data infrastructures and programming models. It focuses on the

Apache Hadoop that is a widely-used infrastructure for big data projects such as HDFS and Hive, HBase, Flume, Pig, Kafka, and Storm. We also summarize the state-of-the-art for privacy-preserving data-insensitive solutions in cloud computing environments.

3.3.1 Big Data Infrastructures and Programming Models

Many data infrastructures have been deployed based on the Apache Hadoop without demand for strong security [146]. Only a few companies have deployed secure Hadoop environments such as Yahoo!. Therefore, Hadoop built-in security requires tailoring for different security requirements. Hadoop operates in two modes: normal (non-secure) and secure modes.

Hadoop normal mode configurations are in non-secure mode. The default mode has no authentication enforcement. It relies on client-side libraries to send the credentials from the user machine operating system in the context of the protocol [146]. Clusters are usually deployed onto private clouds with restricted access to authorized users.

In this model, all users and programs have similar access rights to all data in HDFS. Any user that submits a job could access any data in the cluster and reads any data belonging to other users. Also MR framework does not authenticate or authorize submitted tasks. An adversary is able to tamper with the priorities of other Hadoop jobs in order to make his job complete faster or terminate other jobs [147].

Data confidentiality and key management are also missing in the Hadoop default mode. There is no encryption mechanism deployed to keep data confidential in HDFS and MR clusters. For scenarios where confidentiality is a requirement, other distribution of Hadoop can be utilized to achieve security.

Hadoop secure 's mode consist of authentication, service level authorization and authentication for web consoles. By configuring Hadoop in secure mode, each user and service require authentication by Kerberos in order to use Hadoop services. Since Hadoop requires a user identifier string to identify users, a Portable Operating System Interface (POSIX)-compliant username can be used for authentication purposes. The usernames can also be used during authorization to check the access control lists. Additionally, Hadoop supports the notion of POSIX groups to allow a group of users to access HDFS resources. Authorization checks through access control lists and file permissions are still performed against the client supplied user identifiers.

There is a Remote Procedure Call (RPC) library that is used to provide clients secure access to Hadoop services through sending username over Simple Authentication and Security Layer (SASL). SASL is built on Kerberos or DIGEST-MD5. In Kerberos mode, users acquire a ticket for authentication using SASL for mutual authentication. The digest Message-Digest algorithm 5 (MD5) mechanism uses shared symmetric keys for user authentication with servers to avoid overheads of using a Key Distribution Center (KDC) as a third party for authentication. RPC also

provides data transmission confidentiality between Hadoop services clients through encryption in contrast to the Web console that utilized Hypertext Transfer Protocol Secure (HTTPS).

Kerberos can be used for user authentication in Hadoop secure deployments over encrypted channels. For organizations that require other security solutions not involving Kerberos, this demands setting up a separate authentication system. Hadoop implements SASL/Generic Security Services Application Program Interface (GSS-API) for mutual authentication of users with Kerberos, running processes, and Hadoop services on RPC connections [147]. A secure deployment requires Kerberos settings where each service reads authentication information saved in a key tab file with appropriate permission. A key tab is a file that contains pairs of Kerberos principals and encrypted keys. Keytabs are used by the Hadoop services to avoid entering a password for authentication.

There are several efforts from the industry to enhance the Hadoop security [148] such as Apache Rhino², Apache Knox³, Apache Ranger⁴ and Apache Sentry⁵.

Apache Rhino is an initiative started by Intel at the beginning of 2013 to remarkably enhance the Hadoop ecosystem security. It aims at providing a framework for Hadoop key management, authorization, audit, and logging. Rhino provides a framework support for encryption and key management, a common authorization framework for the Hadoop ecosystem, a token based authentication and single sign-on, and it improves audit logging.

Apache Knox (Gateway) is another effort that aims to provide perimeter security for confidential access to Hadoop clusters through organizational policies within enterprises. Apache Knox enhances the Hadoop security through simplifying users' access to the cluster data and job execution. Client interactions are performed through REST Web services over HTTP. Knox also aims to provide easy integration with existing identity providers and abstracting Kerberos authentication. This is done through encapsulating Kerberos to eliminate the need for client software or client configuration of Kerberos by clients. In addition, it provides integration with SAML, open authorization (OAuth) and OpenID.

Apache Ranger proposes a framework for data security across the Hadoop platforms to enable enterprises to run multiple workloads in a multi-tenant environment. Ranger aims to provide centralized security administration to manage all security related tasks in a central user interface or using REST APIs. Fine grained authorization for specific operations through a central user interface is another goal of Ranger. Support for RBAC and Attribute-Based Access Control (ABAC), in addition to centralized auditing services are among the functionalities of this software.

Enforcing fine-grained role-based authorization for data and metadata located in a Hadoop cluster is provided by Apache Sentry. Sentry implements a policy

²<https://github.com/intelhadop/projectrhino>

³<https://knox.apache.org/>

⁴<http://ranger.incubator.apache.org/>

⁵https://blogs.apache.org/sentry/entry/getting_started

provider to define the access control. This is done by defining a single global policy file that can be applied to enforce access control.

There have been also efforts from academia [149], [150] to formulate the security and privacy issues of big data and also to enhance the security of existing Hadoop distributions. For example, Yu et al. [149] proposed SEHadoop to enhance the Hadoop security in public clouds by increasing the isolation level among the Hadoop components and enforcing the least access privilege for various Hadoop processes. The SEHadoop implements optimized Block Token and Delegation Tokens to avoid authentication key vulnerability and ensuring fine-grained access control. In [150], Dowling et al. implement a solution to isolate multiple studies in HOPS to restrict cross-linking over unauthorized data sources in the Hadoop environment.

Recently in [151], Bertino discuss the specific challenges for big data security and privacy. Similar to traditional information security models, big data solutions must ensure the CIA properties to ensure secure computing. For example, to ensure integrity, big data platform must enforce authorization mechanisms to restrict execution of arbitrary MR jobs over multiple datasets from different owners. Data trustworthiness is another property that ensures accurate analysis for effective decision making. This work also highlights the privacy challenges including efficiency of existing cryptographic techniques to ensure the privacy of data due to scalability issues within big data. The trade-off between security and privacy, data ownership and privacy-aware data lifecycle framework are other privacy challenges that have been discussed.

However, none of these solutions offer flexible Identity and Access Management (IAM), strong authentication mechanisms such as two-factor authentication or provide auditing report for various customized contexts. Our security framework supports the development of multi-tenant Hadoop environment [150].

3.3.2 Privacy-Preserving Solutions in the Cloud

Over the time, organizations have collected valuable information about the individuals in our societies that contain sensitive information, e.g. medical data. Researchers need to access and analyze such data using big data technologies [152, 153] in cloud computing, while organizations are required to enforce data protection compliance.

There has been considerable progress in privacy preservation for sensitive data in both industry and academia, e.g., solutions that develop protocols and tools for anonymization or encryption of data to ensure confidentiality. This section categorizes work related to this area according to different privacy protection requirements.

Pearson [11] discusses a range of security and privacy challenges that are raised by cloud computing. Lack of user control, lack of training and expertise, unauthorized secondary usage, the complexity of regulatory compliance, transborder data flow restrictions and litigation are among the challenges faced in cloud computing environments. In [154], Dove et al. describe the privacy challenges of genomic

data in the cloud including Terms of Service (ToS) of cloud providers that are not developed with a privacy mindset. Awareness of patient to upload their data into the cloud without their consent, multi-tenancy, data monitoring, data security and accountability are among other issues that have been discussed. The authors also provide recommendations for data owners when aiming to use cloud provider services.

In [155], Ayday et al. discussed several privacy issues associated with genomic sequencing. This study also described several open research problems (such as outsourcing to cloud providers, genomic data encryption, replication, integrity, and removal of genomic data) along with giving suggestions to improve privacy through collaboration between different entities and organizations. In another effort by Ayday et al. [156], raw genomic data storage through encrypted short reads is proposed.

Outsourcing privacy is another topic that is discussed by Huang et al. in [157]. The authors define the concept of “outsourcing privacy” where a database owner updates the database over time on untrusted servers. This definition assumes that database clients and the untrusted servers are not able to learn anything about the contents of the databases without authorized access. The authors implement a server-side indexing structure to produce a system that allows a single database owner to privately and efficiently write data to, and multiple database clients to privately read data from, an outsourced database.

Privacy preserving workflow (data, tasks, and structures) scheduling under SLA requirements such as deadline or budget is another effort conducted by Sharif et al. [158, 159]. The authors propose Multiterminal Cut for Privacy in Hybrid Clouds (MPHC) - a multiterminal cut algorithm to partition a workflow with regard to multiple levels of privacy in hybrid cloud environments. This approach ensures private tasks to be accessible only for users with adequate privileges through implementing three levels of privacy in hybrid and private clouds.

Homomorphic encryption is another privacy-preserving solution that is based on the idea of computing with encrypted data without knowing the keys belonging to different parties. To ensure confidentiality, the data owner may encrypt data with a public key and store data in the cloud. When the process engine reads the data, there is no need to have the DP’s private key to decrypt the data. In private computation on encrypted genomic data [160], Lauter et al. proposed a privacy-preserving model for genomic data processing using homomorphic encryption on genome-wide association studies.

Blockchain is a decentralized cryptography system that is used in Bitcoin [161] to ensure anonymity. Each blockchain collects information about the history of spending and it acts as a ledger. Each user has a wallet identifier that is public to the world and a private key that is known only to the owner of the bitcoins. In other words, the ownership of bitcoins is determined by the ownership of private keys.

BaseSpace⁶ is an industrial product of Illumina as NGS cloud platform for the biologists in collaboration with AWS. BaseSpace redirects the produced genetic data from the NGS machines to the amazon data centers and biologists are able to perform specific analysis tasks. Data are secured using Advanced Encryption Standard (AES) 256-bit encryption and transferred through Secure Socket Layer (SSL) channels. Data centers are also compliant with several regulations such as service organization auditing standard No. 70 (SAS70) [162], ISO 2700⁷, payment card industry, and the Federal Information Security Management Act (FISMA) [163]. Less is known about the implementation and efficiency of BaseSpace internals as it's a proprietary commercial product.

However, majority of these solutions to a large extent rely on the trust between various participants and there is no appropriate privacy threat modeling to efficiently enforce the privacy regulations requirements. Additionally, some solutions such as different flavors of homomorphic encryption are still not applicable for complicated data analysis over large datasets. As the first effort in the neuroimaging community, propose a privacy-preserving brain image analysis framework for cloud computing in Chapter 8.

3.3.3 Privacy-Preservation Database Federation

A longtime confidentiality protection strategy is to dilute data by degrading the precision of given data records in a controlled process, so that the database can still satisfy the intended purpose, but is not specific enough to allow for easy re-identification. This challenge can be expressed as a task of managing re-identification risks, based on the identifying level of the attributes while taking into consideration the background knowledge available. The approach relies on an iterative optimization process without providing hard guarantees, mirroring risk management in other aspects of life such as being hit by an accident. A multilingual terminology for talking about privacy by data minimization including anonymity, unlinkability, undetectability, unobservability, pseudonymity, and identity management can be found in [164]. A recording or observational data set is called microdata. Every recording or observation has a set of variables. This set of variables needs to be categorized and may need to be modified in order to apply privacy-preserving data publishing measures. Microdata is expected to be safe when its deliberate or accidental disclosure does not do any harm to the population involved.

To produce safe microdata, variables are categorized into at least three, not necessarily distinct groups: variables that are explicitly and directly identifying, such as personal numbers, social security numbers, serial numbers etc. Key variables (also called pseudo keys, QID or non-sensitive attributes) are a group of variables that are identifying when used together. Linking based on key variables is applied when archived records are processed that have no explicit identifiers, e.g., [165] or when

⁶<https://basespace.illumina.com/home/sequence>

⁷<http://www.27000.org/>

linking attacks are performed such as [4, 166] for the purpose of re-identification. Choosing is often based on mandatory items set forth by law (EU DPD [8], HIPAA [9] and such) or by managing the risk of being fined [167].

And last but not least, by using common sense: as a rule of thumb, non-sensitive attributes are the ones that are likely to appear in other databases, whether publicly accessible or not, therefore, which can potentially be used for linking. A canonical example is a seemingly innocuous gender, date of birth and ZIP code triplet which is highly identifying to the majority of a population and can be found in a vast number of databases. Remaining variables (also called sensitive or non-confidential variables) that are not in the two groups mentioned above. They are either not expected to appear in any other database and, therefore, cannot be used for linking or are not identifying by nature.

Preset software settings for acceptable risk levels may be set by legal requirements to be licensed as public use files or microdata files under contract for research purposes. As an indicator of the current state of affairs, data protected by HIPAA and Safe Harbor regulations result in a re-identification risk measure of approximately 0.04% (that is 4/10.000), ranging between 0.01% to 0.25% and being 10% to 60% in case of restricted data sets under non-disclosure agreements according to [168]. Further experimental measurements can be found in [169].

There are many existing toolkits to help produce safe microdata, providing commonly used anonymization algorithms such as k -anonymity [68] and ℓ -diversity [72]. Argus [170], sdcMicro [171], and University of Texas at Dallas (UTD) anonymization toolbox [172], based on Incognito [173], are examples of open-source toolkits that provide workflow support for anonymizing sensitive data.

There have been several other attempts to provide support for federated queries over different data sources through database federation [174, 175, 176], where there are also two levels (local Personal Identifier (PID) and global PID, collection PID and analysis PID hashes – using PKI keys). The identifier (global PID number, analysis PID etc.) is used to join different study data, not unlike in this case [176]. The Clinical E-Science framework [177] is another attempt to provide data privacy protection using pseudonymization.

There have been also industrial efforts such as Custodix [178] to offer federated queries through implementing Trusted Third Party (TTP) approach, however, such TTP can not be deployed in some jurisdictions because of the existing restrictions.

Different to these solutions we implement a platform-independent anonymization toolkit using R, in Chapter 7, and also propose a two-layer encryption mechanism to enable researchers to issue cross-linking queries over multiple clinical data sources.

3.4 Summary

This chapter reviewed several security and privacy issues on big data in the cloud. It described several key concepts such as virtualization, and containers. We also

discussed several security challenges that are raised by existing or forthcoming privacy legislation, such as the EU DPD and the HIPAA.

The presented results in the area of cloud security and privacy are based on cloud provider activities, such as providing orchestration, resource abstraction, physical resource and cloud service management layers. Security and privacy factors that affect the activities of cloud providers in relation to the legal processing of consumer data were identified and a review of existing research was conducted to summarize the state-of-the-art in the field.

The results of our survey demonstrate that currently there is no developed methodology to properly identify privacy requirements according to privacy legislation for processing sensitive data in cloud computing. There are comprehensive security threat models but privacy is not emphasized. Also, there is not a common protection solution or a simple combination of existing mechanisms to build usable cloud systems while ensuring security and privacy of sensitive data. In the remainder of this thesis, we explore this gap through proposing several usable solutions that can be used to ensure security and privacy in cloud computing environments.

Part II

Privacy by Design for Cloud Computing

Chapter 4

Privacy Threat Modeling Methodology for Cloud Computing Environments

This chapter is mainly based on publications I, III and VII. It describes a new methodology for privacy threat modeling to process sensitive data in cloud computing environments.

4.1 Introduction

Threat modeling is an important part of the process of developing secure software - it provides a structured approach that can be used to identify attacks and to propose countermeasures to prevent vulnerabilities in a system from being exploited [179].

As described earlier, the issues of security and privacy are really two distinct topics [11] as security is a core privacy concept, and the current focus of the existing threat modeling methodologies is not on privacy in cloud computing, which makes it difficult to apply these methodologies to developing privacy-preserving software in the context of cloud computing environments.

We introduce a cloud privacy threat modeling methodology according to the principles of ME [35]. The method that has been applied is one known as “Extension-based”, which is used for enhancing the process of identifying privacy threats by applying meta-models/patterns and predefined requirements. This new methodology that is being proposed provides strong methodological support for privacy legislation and regulation in cloud computing environments. We describe a top-down approach to identify the requirements for an ideal privacy threat modeling methodology in cloud computing and build a new methodology by applying the requirements that were identified.

The rest of this chapter is organized as follows. Section 4.2 describes the characteristics that are desirable in privacy threat modeling for cloud computing en-

vironments. Section 4.3 describes the steps and products for the proposed new methodology. Section 4.4 summarizes the conclusions from this research.

4.2 Characteristics of a Privacy Threat Modeling Methodology for Cloud Computing

This section describes the features that we believe a privacy threat model should have in order to be used for developing privacy-preserving software in clouds in an efficient manner. Based on the properties that are identified, we then apply the Extension-based methodology design approach to constructing an extension of the CPTM for supporting various privacy legislation.

4.2.1 Privacy Legislation Support

Methodological support for the regulatory frameworks that define privacy requirements for processing personal or sensitive data is a key concern. Privacy legislation and regulations can become complicated for cloud customers and software engineering teams, particularly because of the different terminologies in use in the IT and legal fields. In addition, privacy threat modeling are not emphasized in existing threat modeling methodologies, which causes ambiguity for privacy threat identification.

4.2.2 Technical Deployment and Service Models

Cloud computing delivers computing software, platforms, and infrastructures as services based on pay-as-you-go models. Cloud service models can be deployed for on-demand storage and computing power SaaS, PaaS and IaaS [6]. As described earlier, cloud services can be delivered to consumers using different cloud deployment models: private cloud, community cloud, public cloud, and hybrid cloud.

4.2.3 Customer Needs

The actual needs of the cloud consumers must be taken into consideration throughout the whole life-cycle of a project. Additionally, during the course of a project, requests for changes often arise and these may affect the design of the final system. Consequently, it is important to identify any privacy threats arising from the customer needs that result from such change requests. Customer satisfaction can be achieved through engaging customers from the early stages of threat modeling so that the resulting system satisfies the customer's needs while maintaining adequate levels of privacy.

4.2.4 Usability

Cloud-based tools aim at reducing IT costs and supporting faster release cycles of high-quality software. Threat modeling mechanisms for cloud environments should, therefore, be compatible with the typical fast pace of software development in clouds based projects. However producing easy-to-use products with an appropriate balance between maintaining the required levels of privacy while satisfying the consumer's demands can be challenging when it comes to cloud environments.

4.2.5 Traceability

Each potential threat that is identified should be documented accurately and be traceable in conjunction with the associated privacy requirements. If threats can be traced in this manner, it means that threat modeling activities are efficient in the tracing of the original privacy requirements that are included in the contextual information and changes over the post requirement steps such as design, implementation, verification, and validation.

4.3 Methodology Steps and Their Products

Motivated by the facts that privacy and security are two distinct topics and that no single methodology could fit all possible software development activities, we apply ME that aims to construct methodologies to satisfy the demands of specific organizations or projects [180]. In [35], ME is defined as “the engineering discipline to design, construct, and adapt methods, techniques, and tools for the development of information systems”.

There are several approaches to ME [180, 181] such as a fundamentally “ad-hoc” approach where a new method is constructed from scratch, “paradigm-based” approaches where an existing meta-model is instantiated, abstracted or adapted to achieve the target methodology, “Extension-based” approaches that aim to enhance an existing methodology with new concepts and features, and “assembly-based” approaches where a methodology is constructed by assembling method fragments within a repository.

Figure 4.1 represents different phases in a common Secure Development Life Cycle (SDLC). Initial security requirements are collected and managed in the requirements engineering phase (A). This includes identifying the quality attributes of the project and assessing the risk associated with achieving them. A design is composed of the architectural solution, attack surface analysis, and the privacy threat model. Potential privacy threats against the software that is being developed are identified and solutions are proposed to mitigate for adversarial attacks (B). The proposed solution from the design phase is implemented through a technical solution and deployment (C). This includes performing static analysis on the source code for software comprehension without actually executing programs. The verification process (D) includes extensive testing, dynamic analysis on the execut-

ing programs on virtual resources and fuzzing as a black-box testing approach to discover coding errors and security loopholes in the cloud system. Finally, in the Validation phase, the end-users participate to assess the actual results versus their expectations, and may put forth further change requests if needed.

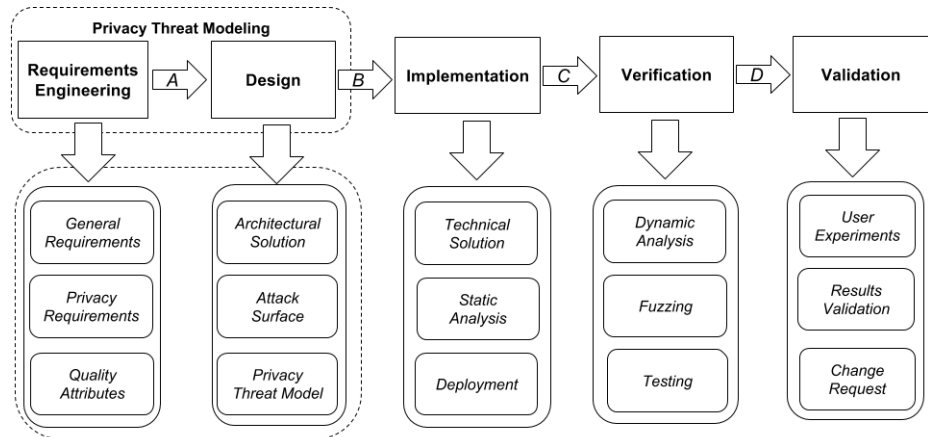


Figure 4.1: Privacy threat modeling in requirements engineering and design of a SDLC

Our proposed methodology identifies the privacy requirements in the Requirements Engineering step, as shown in Figure 4.2. The results from the Requirements Engineering, which include specifications for privacy regulatory compliance, are fed into the Design step, where activities such as specifying the appropriate cloud environment, identifying privacy threats, evaluating risks and mitigating threats are conducted. Then the produced privacy threat model would be used in the implementation step finally it would be verified and validated in the subsequent steps.

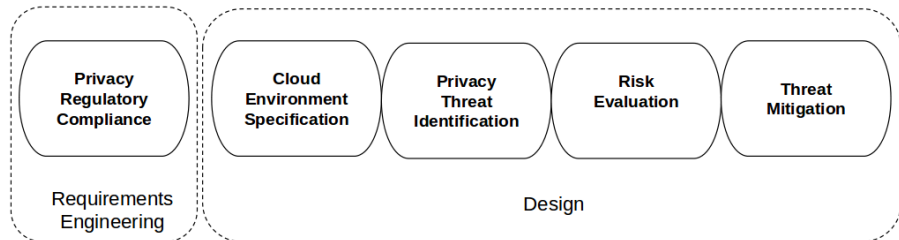


Figure 4.2: The CPTM methodology steps

Cloud stakeholders and participants such as cloud users, software engineering team and legal experts will engage in the activities shown in Figure 4.2 to implement the threat model in the context of steps A and B in Figure 4.1. Cloud software architect as a member of the software engineering team initiates a learning session to clarify the methodology steps and their products, privacy requirements (introducing the law title that is needed to be enforced in the cloud environment), and quality attributes such as performance, usability. The legal experts will identify the definitive requirements that ensure the privacy of data in the platform. In the Design step, the cloud software architect presents the architecture of the developing cloud environment for various participants. This will result in a unified terminology to be used in the privacy threat model.

The rest of this section outlines the implementation model of the steps that are represented in Figure 4.2.

4.3.1 Privacy Regulatory Compliance

Interpreting privacy regulatory frameworks can be often complex for software engineering teams. In the privacy regulatory compliance step, learning sessions with privacy experts, end-users and requirements engineers facilitates the elicitation of privacy requirements (PR). For example, in the EU DPD some of the privacy requirements are lawfulness, informed consent, purpose binding, transparency, data minimization, data accuracy, data security, and accountability [36]. Each of the requirements that are identified will be labeled with an identifier, e.g., (PRi), name and description to be used in later stages.

4.3.2 Cloud Environment Specification

To ensure that the final cloud software will comply with the relevant legal and regulatory framework, several of the key characteristics that are affected by cloud computing services (including virtualization, outsourcing, off-shoring, and autonomic technologies) must be specified. For this purpose, the physical/logical architectures of the deployment and service model can be developed according to the following steps.

- **Step A:** Define the cloud actors (such as Cloud Consumer, Cloud Provider, Cloud Auditor, Cloud Broker, and Cloud Carrier) [2]. Cloud consumer is a person or organization uses service from cloud providers in the context of a business relationship. Cloud provider makes service available to interested users. Cloud auditor conducts an independent assessment of cloud services, operations, performance and security of the deployment. Cloud broker manages the use, performance and delivery of cloud services and establishes relationships between cloud providers and cloud consumers. Cloud carrier provides connectivity and transport of cloud services from cloud providers to cloud consumers through the network.

- **Step B:** Describe a detailed model of the cloud deployment physical architecture where the components will be deployed across the cloud infrastructure. This should give details of where the components will be deployed and run, for example, the operating system version, the database version, the virtual machine location, and where the database server will run.
- **Step C:** Describe the logical architecture of the cloud services model where the major cloud services, along with and the relationships between them that are necessary to fulfill the project requirements, are recorded. This should include the data flow and connections between the relevant cloud services and actors. Note that in this context, an entity is a cloud service with a set of properties that meet a specific functional requirement.
- **Step D:** Describe the assets that need to be protected, the boundaries of the cloud and any potential attackers that might endanger either the cloud environment or the assets that have been identified as being associated with that particular cloud.

The cloud environment specification step consists of composing an architectural report including assets that are subject to privacy protection, cloud actors, physical architecture of the deployment model, and logical architecture of the service model.

4.3.3 Privacy Threat Identification

In this step, privacy threats against the PRs that were established in privacy regulatory compliance definition phase will be identified and analyzed. To achieve this, the system designers will undertake the following steps.

- **Step A:** Select a privacy requirement from the PR list for threat analysis, e.g., (PR2).
- **Step B:** Correlate identified cloud actors (Step A from Section 3.2) with the actor roles that are defined in the project's privacy law. For example, correlating the Data Controller role as a Cloud Consumer, or the Data Processor role as a Cloud Provider in the DPD.
- **Step C:** Identify all the technical threats that can be launched by an adversary to privacy and label them in the specified cloud environment. Each identified threat can be named as a $T_{i,j}$, where i indicates that threat T that corresponds to PR_i and j indicates the actual threat number. For example, in $T_{2.5}$, digit 2 indicates the relevance of the threat to PR_2 and digit 5 is the actual threat number.
- **Step D:** Repeat the previous steps until all PRs are processed.

The threat identification step consists of composing an analysis report including a list of threats including id, name, date, author, threat scenario for each class of the PRs.

4.3.4 Risk Evaluation

In this step, all actors participate to rank the threats that have been identified in previous step (Section 4.3.3) with regard to their estimated level of importance and the expected severity of their effect on the overall privacy of the cloud environment. The *Importance* indicates the likelihood of a particular threat occurring and the level of the *Effect* indicates the likely severity of the damage if that threat against the cloud environment were carried out.

Assume there are three identified PRs (PR1, PR2, PR3) in addition to related privacy threats T1.4, T2.1, and T3.3 from previous steps for an imaginary cloud system. In this imagined cloud, various participants in the project such as Alice (Cloud Consumer), Bob (Cloud Provider), Dennis (Software Architect), Tom (Lawyer) and Rosa (Cloud Carrier) evaluate the corresponding risk of each identified threats, as illustrated in Table 4.1.

ID	Name	Exploit Scenario	I	E	Participants
T1.4	Data Accumulation over Time	The cloud system stores a huge amount of data from Cloud Consumers over the time. This can be done through extensive analysis of collected data from different sources.	H	M	Alice, Bob, Dennis, Tom
T2.1	Linkability of Records	A record owner can be linked through the adversarial background knowledge for the published data to the Cloud Provider.	H	H	Alice, Bob, Dennis, Tom
T1.4	Cross-linking of data processing	A Cloud Consumer is able to run cross-linking queries over multiple data sets from different data sources.	M	H	Alice, Bob, Dennis, Tom

Table 4.1: Prioritization of the identified threats, L (Low), M (Moderate), H(High)

4.3.5 Threat Mitigation

In this step, the threat modeling team proposes countermeasures to the threats that were identified in the previous step as having the highest likelihood of occurrence and the worst potential effects on the cloud environment. Each countermeasure should clearly describe a solution that reduces the probability of the threat occur-

ring and that also reduces the negative effects of the cloud if the threat was carried out.

Finally, the recommended countermeasures from this step should be documented and fed into the implementation step to be realized through coding and for their effectiveness to be assessed by static analysis. In the later stages of verification and validation, each such countermeasure will be evaluated and approved by the participants.

4.4 Summary

In this chapter, we identified the requirements to build a privacy threat modeling methodology for cloud computing environments using an Extension-based ME approach. For this purpose, we introduced Cloud Privacy Threat Modeling (CPTM) for compliance with various legal and regulatory frameworks, in addition to improving the threat identification process.

As proof of concept, we will apply the proposed methodology within the BioBankCloud that aims to process the sensitive data, as discussed in next Chapter 5. This will be a first step to validate our proposed methodology for providing customized privacy threat modeling in bioinformatics cloud computing environments.

Chapter 5

Case Study: BioBankCloud Privacy Threat Modeling

This chapter is mainly based on publications I, III and VII. It describes the implementation of our proposed methodology in Chapter 4 for a PaaS cloud. This includes a high-level definition of the BioBankCloud and an exemplary workflow.

5.1 Introduction

BioBankClouds are gaining popularity due to flexibility and scalability in processing big genomic data. BioBankClouds are cost-efficient for biobanks as commodity hardware ownership is no longer required. However, it can be complicated to correctly identify the relevant privacy requirements for processing sensitive data in cloud computing environments due to the range of privacy legislation and regulations that exist.

Some examples of such legislation are the EU DPD [8] and the US HIPAA [9], both of which demand privacy-preservation for handling personally identifiable information. In addition, there is no consensus within International law whether specific requirements should be applicable to genetic information. There are several documents at the regional and international level that include some guidelines, for example, UNESCO International Declaration on Human Genetic Data (2003) and the Guidelines on Human Biobanks and Genetic Research Databases (2009). The Council of Europe has enacted an Additional Protocol to the Convention on Human Rights and Biomedicine, concerning Genetic Testing for Health Purposes (2008), which is legally binding to the states that have ratified it.

Also national law might contain specific regulation entailing further conditions and criteria for the handling to be legal and thereby legitimate. In Sweden for example, the Act on Genetic Integrity¹ lays down specific requirements when per-

¹Lag (2006:351) om genetic integritet m.m.

forming genetic testing or taking part of the genetic information. These types of requirements also exist in the other Member States.

The EU DPD is considered as a general super-national law that specifies objectives and requirements that must be adopted by the Member States. Recent biobank legislation in several jurisdictions in Europe (such as Finland [182] and Sweden [183]) has led to the definition of a role for managing sensitive medical data, such as Data Access Controllers, Data Processors, and Auditors as the adoption of the EU DPD.

Privacy threat modeling, an essential stage of secure software engineering, encourages communication of privacy requirements among stakeholders of the biobank when developing privacy-preserving cloud services. There has been considerable progress in development of information security threat modeling frameworks and tools, for example, OCTAVE [184] and STRIDE [179]. Unfortunately, the complexity of such frameworks makes them difficult to be used for a project that demands agile methods with limited resources. It is also worth noting that privacy-preservation is not emphasized in the existing security threat modeling frameworks.

To this end, we aim to formulate the privacy requirements and identify the privacy threats of a BioBankCloud according to the EU DPD [8]. As a result of applying our methodology, a set of 8 key privacy requirements of the EU DPD, in addition, to 26 critical privacy threats have been identified. The threats have been evaluated and their exploit paths by an adversary have been described. Finally, we have proposed countermeasures to mitigate effects of exploiting those threats in the BioBankCloud through adversarial attacks.

This chapter is organized as follows. Section 5.2, describes a generic scenario of the BioBankCloud. Section 5.3, outlines the privacy requirements of the EU DPD. Section 5.4 specifies the design of the BioBankCloud environment. Section 5.5, describes the threat identification process. Section 5.6 risk evaluation. Section 5.7, describes the countermeasures for the identified threats. Section 5.8, summarizes our findings and conclusions.

5.2 Scenario

As described earlier, the BioBankCloud platform is a collaborative project bringing together computer scientists, bioinformaticians, pathologists, and biobankers [39]. BioBankCloud aims to provide the capability of deploying sequencing applications with their dependencies within an environment called a container within a cloud computing environment.

Assume the BioBankCloud participants are Alice (Researcher) as Cloud Consumer representative, Bob (BioBankCloud) as Cloud Provider, Dennis as Auditor, Tom as Lawyer and Ove as Cloud Software Architect both evaluate the corresponding risk of each identified threats. A typical scenario to use the BioBankCloud platform by Alice (Researcher) is as following steps.

1. Alice registers an account in the platform and by default a guest account will be created for her.
2. Bob enables Alice and entitles appropriate access control levels.
3. Alice opens the login page and enters the authentication credentials.
4. Bob authenticates the user according to the presented credentials.
5. If the Alice is authenticated, Bob will permit her login to the platform.
6. Alice uploads genomics data and tries to run a workflow on the platform.
7. Bob ensures that Alice has enough permission to run the workflow.
8. Depending on the permission check:
 - a) If Alice is not authorized to run the workflow she will be denied by Bob.
 - b) If Alice is granted to access the resources demanded by the workflow, she will be permitted
9. Bob authorizes Alice to run the workflow and access the genomic data in the platform through the platform execution cluster. This cluster stores the results in the platform and presents the results to Alice.
10. The platform stores the log information from the above steps for auditing purposes by Dennis (Auditor).

This usage scenario will be used in Sections 5.4 for the actor-role correlations step.

5.3 Privacy Requirements

The BioBankCloud platform demands to enforce the DPD requirements. As described earlier, the EU DPD is the EU's initial attempt at privacy protection to harmonize the regulations for information flow within the EU Member States. The identified privacy requirements include the main roles of the EU DPD and the fundamental PRs are lawfulness, informed consent, purpose binding, data minimization, data accuracy, transparency, data security, and accountability.

The DPD also highlights the demand for cross-border transfer of data through non-legislative measures and self-control. One example of where these types of privacy principles are being used is the Safe Harbor Agreement which made data transfer possible to US-based cloud providers that are assumed to have appropriate data protection mechanisms. However, as seen above, the European Court of Justice declared the Safe Harbor Agreement invalid in a ruling in October 2016 [185].

There is an ongoing effort [19, 186] to replace the EU DPD with a new General Data Protection Regulation containing more than 130 recitals and 91 articles that

aim to lay out a data protection framework in Europe [187]. The proposed regulation will have a wider scope of application, covering the processing of data from the third state that is directed to the EU, such as to offer goods and services. Also, the understanding of what is personal data is expanded in one specific context, namely through recital 25a of the proposal of the Regulation where it is stated that genetic data should be defined as personal data relating to the genetic characteristics of an individual which have been inherited or acquired as they result from an analysis of a biological sample from the individual in question, in particular, Deoxyribo Nucleic Acid (DNA) or Ribo Nucleic Acid (RNA). This will restrict the processing of genetic data since it could never be considered to fall outside the Regulation. The regulation also includes definitions of new roles related to handling data (such as data transfer officers) and considerably strengthens the role and functions of national data authorities [186]. For example, the regulation confers the power to the authorities to impose significant penalties for privacy breaches that result from violations of the regulations, for example, such a penalty could be 0.5 percent of the worldwide annual turnover of the offending enterprise [188].

In the following, we describe the key requirements of the EU DPD to handle genomic data.

- **PR1 Lawfulness**² sets out the basic premises for the legitimate processing of data, that all processing must be conducted within the regulatory framework of the DPD. Data processing can be allowed on the basis of for example statutory permissions (such as legislation), or with data subject consent, if necessary for the performance of a contract or on the basis of statutory permission such as legislation. In regards to sensitive data such as health data, the DPD holds that the EU Member States must provide further safeguards, for example through the involvement of a research ethics committee. Procedures and processes to disclose the sensitive genomic data are governed strictly by the lawfulness requirement. It can also be seen as an umbrella principle, reflecting other requirements and being the point of departure for true data protection.
- **PR2 Informed Consent**³ informed consent justifies processing of genomic data in the BioBankCloud. The genomic data may have been provided with informed consent through the controller which constitutes the main justification for processing. In cases where data has been collected a long time ago, where the data subject is deceased, or in the case of data on anonymous cell lines, the data may no longer be able to connect to an individual person and thereby fall outside the scope of the DPD. In such case, even non-consented use of data may be permitted. Further, there might be some room to re-use previously consented data if it conforms to the law applicable to the controller and this legislative act indicates the purpose for processing and the purpose is

²Paras 18, 23, 28 of the Preamble, Article 6 of the DPD.

³Para 30 of the Preamble, Article 7 of the DPD.

of substantial public interest. This law or the decision must include necessary safeguards so that the interests of the data subjects are effectively protected.

- **PR3 Purpose Binding**⁴ ensures that personal data processing is performed according to predetermined purposes. The collected genetic data in the BioBankCloud will only be processed according to the purposes covered by the informed consent given by the subject or, if the law applicable to the DP so admits, according to further purposes within the legal framework.
- **PR4 Data Minimization**⁵ restricts extra and unnecessary disclosure of information to third parties, such as a processor, to reduce the risk of information leakage that leads to privacy breaches. This requirement of the DPD demands a retention period of the published genetic data to be monitored closely. Storage over time can only be permitted if in accordance with the law applicable to the controller.
- **PR5 Data Accuracy**⁶ describes the necessity to keep data accurate and to be updated by the controller. A controller holding personal information shall not use that information without taking steps to ensure with reasonable certainty that the data are accurate and up to date. The obligation to ensure the accuracy of data must be seen in the context of the purpose of data processing. In line with the principle of accuracy, data subjects must have the right under national law to obtain from the controller the rectification, erasure or blocking of their data if they think that their processing does not comply with the provision of the directive, in particular because of the inaccurate or incomplete nature of the data. Data accuracy requirement is closely linked with Transparency (PR6), as described in the following.
- **PR6 Transparency**⁷ entitles the data subjects to have information about the processing of their data and thereby a means to learn of the processing operation of their data. Transparency thus functions as a prerequisite for the data subjects to monitor that the data is accurate, in accordance to PR5. Transparent data processing is required to be implemented in the BioBankCloud with a clear description of technical, physical and organizational measures that processor has in place to infer if data are processed appropriately.
- **PR7 Data Security**⁸ proposes implementing technical measures to provide legitimate access and organizational safeguards. The controller shall ensure

⁴Paras 28-31 of the Preamble, Articles 6 and 7 of the DPD.

⁵Paras 59-61 of the Preamble, Articles 16-17 of the DPD.

⁶Paras 28 and 41 of the Preamble.

⁷Paras 38-40 of the Preamble, Articles 10-15 of the DPD.

⁸Para 46 of the Preamble, Articles 6, 16-17 of the DPD.

that whoever processes the data on his behalf, e.g., the processor provides adequate levels of security against unlawful data processing. It is stated in Article 17 DPD that Member States shall provide that the controller (data provider) must implement appropriate technical and organizational measures to protect personal data against accidental or unlawful destruction or accidental loss, alteration, unauthorized disclosure or access, in particular where the processing involves the transmission of data over a network and against all other unlawful forms of processing. Data security covers the equipment (hardware, software, etc.) but also organizational aspects such as internal rules on how communication with and from the staff of the processor is dealt with, how responsibilities are handled internally etc., and how access to facilities of data storage is regulated.

- **PR8 Accountability**⁹ mandates internal, external auditing and control for various assurance reasons. The data provider is responsible for ensuring compliant of supplied genomic data usage to the processors. For instance, to ensure that confidentiality and integrity of data have been preserved by the acting processors. The processors shall put in place measures which would under normal circumstances guarantee that data protection rules are adhered to in the context of processing operations; and have documentation ready which proves to the data provider and to supervisory authorities what measures have been taken to achieve adherence to the data protection rules. This criterion requires the controller to act in a proactive manner, to actively demonstrate compliance and not merely wait for data subjects or supervisory authorities to point out shortcomings.

5.4 Cloud Environment Specification

- **Step A:** The BioBankCloud actors are identified through the general use cases. For example, from the usage scenario, Researcher, Guest, Data Provider, Administrator, and Auditor actors are identified as described in Table 5.1 .

- **Step B:** BioBankCloud Physical Architecture

The BioBankCloud is designed as a PaaS to be easily installed on a private cloud using Karamel and Chef. The Chef recipes parametrize Vagrant to create the virtualized clusters and services over 14.04 images using HOPS, as shown in Figure fig:physical. HOPS utilize the Hadoop version 2.x within a Hadoop Cluster (Hops-YARN) and HOPS infrastructure containing MySQL 7.x cluster and HOPS file system (Hops-FS).

GlassFish server version 4.0.1 over JDK 7.0 is used to run the big data lab LIMS. This server is confined between external and internal firewalls to separate it from the trusted private network behind the internal firewall. The

⁹Paras 55-64 of the Preamble, Articles 22-24 of the DPD.

Table 5.1: Correlating the domain actors to the cloud actors

Domain Actor	Cloud Actor	Description
<ol style="list-style-type: none"> 1. Researchers 2. Guest 3. Data Provider 	Cloud Consumer: A person or organization that uses service from the BioBankCloud.	<ol style="list-style-type: none"> 1. Researchers who are affiliated with the institutions that hold the genomic data. The researcher acts under the responsibility of the Data Provider, as described in this section. The guest researcher conducts experiments on subjects genomic data. 2. The Guest is able to log into the BioBankCloud and browse the public content of the platform. 3. Data Provider is the person responsible for data sharing. The Guest can only access the resources and data in the BioBankCloud after permission has been granted by the data provider.
Administrator	Cloud Provider: The BioBankCloud platform that makes service available to interested users.	The administrator enables the users of the platform and installs new libraries. This actor can also assign/revoke roles from the users.
Auditor	Cloud Auditor: Conducts independent assessment of cloud services, operations, performance and security of the deployment.	The Auditor is able to see major identity and access management events, in addition to user authentication, authorization events to ensure compliance with the privacy legislation.

platform users are able to access the web pages from the browsers on their

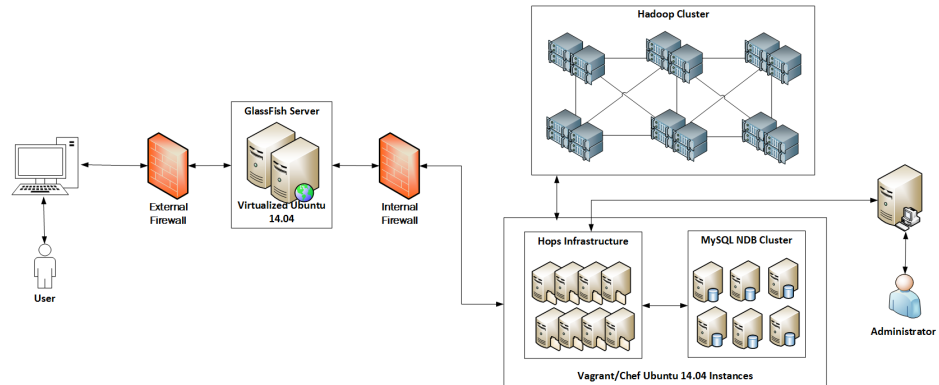


Figure 5.1: BioBankCloud Physical architectures

computers through an external firewall while platform administrator can only access the platform within the trusted network.

- **Step C:** BioBankCloud Logical Architecture

The BioBankCloud LIMS consists of several components for secure access, workflow execution, data sharing, as shown in Figure 5.2. Hops-FS as the next-generation of the HDFS supports multiple stateless Name Nodes but it keeps the metadata in a MySQL Cluster. Hops-YARN is another component that provides distributed stateless resource management with storing states in the MySQL Cluster for fault-tolerant.

The BioBankCloud services will be deployed in a PaaS private cloud. The user requests will be dispatched to a multi-tenant Hadoop cluster that is managed by the Hops-YARN. The Hadoop cluster runs submitted jobs and will notify the user through the LIMS interfaces. The Node Manager is an agent that is responsible for containers, monitoring their resource usage such as CPU, memory, network and reporting to the Hops-YARN resource manager. A DataNode manages storage of each container node using keeping a set of blocks related to files. When a workflow is executed, its algorithmic dependencies will be fetched into the container as a data-staging phase.

- **Step D:** Assets, Boundaries and Attackers

The BioBankCloud platform stores genomic data which constitute as the main asset. The platform services that are provided to the external users can also be considered as assets. The security boundaries consist of firewalls that control the incoming/outgoing traffic through the BioBankCloud along with the physical means to deny access to the computing platform for unauthorized persons. Eavesdroppers and malicious users are known as attackers or adversaries that are able to exploit the possible vulnerabilities in the platform.

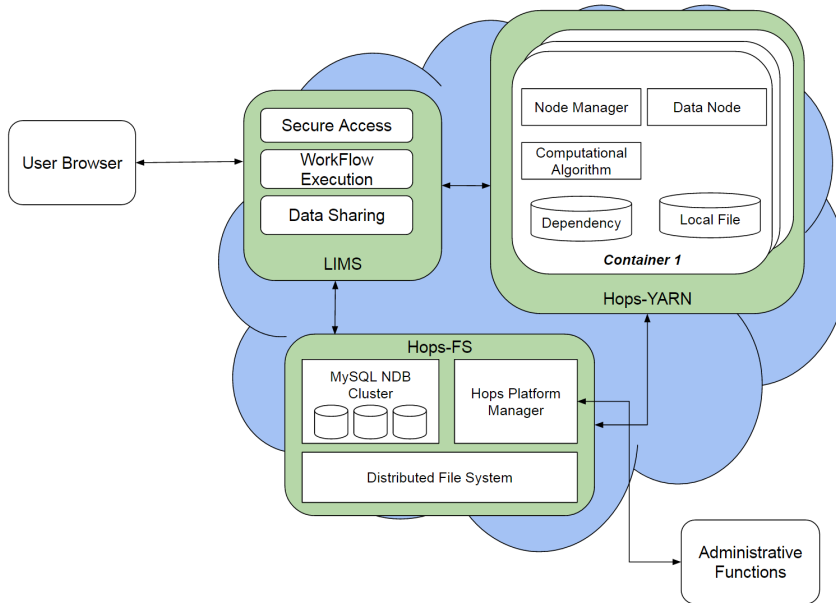


Figure 5.2: Logical architecture

5.5 Privacy Threat Identification

- **Step A:** Select a privacy requirement from the PR list for threat analysis from PR1, PR2, . . . , PR8.
- **Step B:** Correlate the domain actors to the DPD roles, as shown in Table 5.2.
- **Step C:** Identify the technical threats that can be launched by an adversary to privacy and label them in the specified cloud environment. Each identified threat can be named as a $T_{i,j}$, where i indicates that threat T that corresponds to PR_i and j indicates the actual threat number. For example, in $T_{2.5}$ value, 2 indicates the relevance of the threat to PR_2 and value 5 is the actual threat number.

T1: Lawfulness

Lawfulness can be considered an umbrella principle. In this context, any violation the PR's may amount to a lawless threat.

- T1.1: Lack of relevant information on legal rights and duties to allow data subject and other interested parties to use effective means for accessing the data.

Table 5.2: Correlating the BioBankCloud actors with the DPD roles

Cloud Actor	Domain	DPD Role	Role Description
Cloud Consumer		Data Subject or Data Controller	Data Subject: The individual person from which the genomic data has been extracted Data Controller: The entity that owns or provides the genomic data to the platform and decides on the processing of the data.
Cloud Provider		Data Processor	The BioBankCloud platform and all the administrative resources that provide the computing services to the Cloud Consumers. The Data Processor may only act on behalf of the Controller.
Cloud Auditor		Data Processor or Data Controller	Auditor inspects compliance with legal and accountable processing and storage of the genomic data according to the DPD rules. If the auditor simply processes the data on behalf of the controller (e.g, internal control) then she is a processor. Otherwise, she can act as a controller on her own right.

- T1.2: Amendments to legal requirements or unawareness of new rules. Incorrect interpretation or application of legal concepts leading to unlawful processing of (complex) data.
- T1.3: Lack of agreement between all entities regarding the processing the genomic data.
- T1.4: If data are not obtained in accordance with the law, or without approval (informed consent or ethics board).

T2: Informed Consent

- T2.1: Excessive ToS containing too much specific information that is not clear enough, e.g., legal terms that are not easily understandable to a layman.

- T2.2: Lack of possibility to give consent dynamically to a specific subset of genomic data.

T3: Purpose Binding

- T3.1: Researcher does not use the data according to the initial purpose or the Data Provider does not use the data according to the purposes.
- T3.2: Researcher who has access to multiple data studies makes cross-link analysis to the genomic data that does not consent and, hence illegal, according to the PR1 and PR2.

T4: Data Minimization

- T4.1: The requested sensitive data to be used by a cloud provider or guest researcher is not certain or well defined in advance.
- T4.2: If the Data Provider does not define the retention period of the sensitive (genomic) data, there is a threat of accumulating more and more sensitive data over time, that can be used for inference and linking attacks.

T5: Data Accuracy

- T5.1: If the Data Provider cannot or will not update data when having wrong information has been found to be incorrect.
- T5.2: Data Provider uploads data to the cloud provider but data source validity is not affirmed.

T6: Transparency

- T6.1: Lack of communication and information between entities. The threat is even more severe when the lack of information or openness is harmful to weaker parties, the sample donor/data subject.
- T6.2: Researchers or Data Provider cannot get access to modify or erase data, due to unclear data processing procedures.

T7: Data Security

- T7.1: Theft of authentication credentials by an adversary through phishing attacks or network eavesdropping, brute force attacks to guess authentication credentials or identities of users.
- T7.2: Injecting code such as SQL injection, Identity-Based Encryption (XSS) format, type or length of input data.
- T7.3: Wide access to data by a large group of people.

- T7.4: Elevation of privileges by an attacker to change the access rights to higher privileges.
- T7.5: Unavailability of data due to Denial of Service (DoS) attacks.
- T7.6: Session replies through message theft by eavesdropping to steal a session.
- T7.7: Inference attacks through guessing or background knowledge to the minimized data, required by PR4.
- T7.8: Storing passwords, credentials, database connections, keys in plain text or within the source code.
- T7.9: Existing bugs in the OS kernel that can be a target for an attacker to exploit them.

T8: Accountability

- T8.1: Lack of mechanism for secure auditing to provide evidence of confidentiality and integrity.
- T8.2: Logging information or audit trails contain sensitive information about the subjects.
- T8.3: Excessive information in the audit logs to make audit and inspection about the usage of the genomic data by an auditor.

5.6 Risk Evaluation

The BioBankCloud participants such as analyst, architects and lawyers participate and evaluate the important of the threats as summarized in the Table 5.3.

ID	Name	Exploit Scenario	I	E	Participants
T1.1	Lack of Lawfulness	An adversary uses the BioBankCloud to process or store genomics information due to lack of lawfulness information.	L	H	Bob, Dennis, Tom
T1.2	Amendments of Regulations	An adversary uses the BioBankCloud to avoid compliance with new regulations that may have strict privacy requirements.	L	M	Alice, Bob, Tom, Ove

T1.3	Lack of Agreement	An adversary is able to process and achieve results on the platform without agreeing to the BioBankCloud ToS.	M	H	Alice, Bob, Tom, Ove, Dennis
T1.4	Illegally Collected Data	An adversary unlawfully process data without ethical and legal permissions.	M	M	Alice, Bob, Tom, Ove, Dennis
T2.1	Excessive ToS	The BioBankCloud ToS contains many complicated legal terms that clients are not able to understand.	H	H	Alice, Bob, Tom, Ove
T2.2	Dynamic Consents	The BioBankCloud platform restricts user to update the consent and ethical information.	H	M	Alice, Bob, Tom, Ove
T3.1	Misuse of Data	Researcher does not use the data according to the initial purpose or the DP does not use the data according to the purposes. Also the BioBankCloud should not share the genomics data or users data to other external entities.	L	H	Alice, Bob, Ove, Dennis
T3.2	Cross-linking over Datasets	Researcher who has access to multiple data studies makes cross-link analysis to the genomic data that does not consent.	M	H	Alice, Bob, Ove, Dennis
T4.1	Excessive Information	A Researcher uploads unnecessary information of data subjects that is not required by the platform.	M	L	Alice, Bob, Ove
T4.2	Data Minimization	The BioBankCloud platform accumulates huge amount of information over time from users who have not set data retention period.	M	M	Alice, Bob, Tom, Ove, Dennis

T5.1	Data Modification	The BioBankCloud is not able to update the user information or genomics data when Data Provider or Cloud Consumer decides to update the information.	M	M	Alice, Bob, Tom, Ove, Dennis
T5.2	Data Source Validity	An adversary uploads compromised genomics data without valid data source references.	L	L	Alice, Bob, Tom, Dennis
T6.1	Unclear Processing of Data	The BioBankCloud restricts other participants to access the information about the data usage in the platform and how genomic data is processed.	H	M	Alice, Bob, Tom, Ove
T6.2	Transparent Data Access	Researchers or Data Provider cannot get access to modify or erase data, due to unclear data processing procedures.	L	M	Alice, Bob, Tom, Ove, Dennis
T7.1	Weak Authentication	An adversary is able to steal user credentials through network eavesdropping or password theft and make reply attacks.	H	H	Alice, Bob, Tom, Ove, Dennis
T7.2	Input Validation	An adversary is able to issue SQL injection or XSS over web pages.	H	H	Alice, Bob, Ove
T7.3	Misusing the Roles	Wide access to data by a large group of people.	M	H	Alice, Bob, Tom, Ove, Dennis
T7.4	Unauthorized Access	An adversary is able to change her privileges in order to get permissions from other roles that she is not entitled.	M	H	Alice, Bob, Ove

T7.5	Unavailable Services	The BioBankCloud platform is not accessible due to technical or adversarial attacks.	M	M	Alice, Bob, Ove
T7.6	Session Theft	An adversary is able to steal other users sessions to compromise the integrity of data.	M	M	Alice, Bob, Ove
T7.7	Re-identification of Individuals	An adversary is able to re-identify individuals from the data sets with prior background knowledge.	M	H	Alice, Bob, Ove
T7.8	Theft of Credentials in Source Code	An adversary is able to steal the platform master keys and credentials that are written in the source code or unsecured text storage.	M	H	Alice, Bob, Ove
T7.9	Kernel Exploits	An attacker is able to exploit OS kernel bugs to gain access over the host VM or other guest VMs.	H	H	Alice, Bob, Ove
T8.1	Unsecured Auditing	The auditor is not able to infer proof of CIA from the audit trails.	M	M	Alice, Bob, Tom, Ove, Dennis
T8.2	Sensitive Info in Logs	An adversary is able to extract sensitive information such as data subjects personally identifiable information or credentials from the platform.	L	M	Bob, Tom, Ove, Dennis
T8.3	Unusable Log Files	Auditor is not able to extract information from usable audit reports. For example too much information or inconsistent log files due to time synchronization in the distributed systems.	H	M	Bob, Tom, Ove, Dennis

Table 5.3: Risk evaluation matrix for the identified threats. **I** indicates the likelihood of threat and **E** indicates the effect of exploiting the threat on the whole BioBankCloud.

5.7 Threat Mitigation

The cloud participants propose trade-off mitigation solutions as a countermeasure for the evaluated threats from the previous step to meet the architectural goals such as privacy vs. usability. Each proposed countermeasure refers to the corresponding threat in Section 5.5, i.e. C4.1 indicates countermeasure for the threat T4.1.

- C1.1: Establish procedures with legal basis according to PR1, PR2, . . . , PR8 to clarify the legitimate rights and responsibilities for each role.
- C1.2: Apply the possible changes from the new laws or regulations that might affect processing of the genomic data within the DPD framework.
- C1.3: Upon registration, new users provide definitions of the platform ToS and user responsibilities.
- C1.4: Assume uploaded data could have been collected illegally by a Cloud Consumer and for this purpose ethical and consent forms of data subject must be presented and validated by Administrator role.
- C2.1: The BioBankCloud ToS should not contain complicated legal terms or conditions that are difficult for a non-law expert to interpret. It only requires to clearly describe the responsibilities short and precisely.
- C2.2: The consents and ethical information can be updated by the Cloud Consumer whenever there are new forms. The platform stores older versions for auditing purposes.
- C3.1: For every new data set there should be new consents and ethical forms. For example, if there are a running approved project and a set of new data is uploaded, the Administrator should be provided with further evidence of ethical permits over the study.
- C3.2: Restrict users with the aim to run cross-linking analytical jobs over multiple independent data sets that are not consented by the data subjects. Also, data sets or Cloud Consumer information must be kept confidential to the BioBankCloud.
- C4.1: The platform should not be implemented in a way to require excessive information from the Researchers or data subjects.
- C4.2: Users will update the data retention period and when this date expires the Administrator manually delete the data after notifying the user.
- C5.1: Allow all users to be able to update their information on-demand in addition to update or erasure of genomics data.

- C5.2: Administrator should be able to approve or reject projects based on their validity. For example, if an uploaded data is incompatible with the platform or data has been acquired from unknown sources, the Administrator can reject processing that project.
- C6.1: The platform data usage should be transparent for authorized parties such as Administrators and Auditors to see all the activities. Data Provider to see the data access attempts on the uploaded data sets.
- C6.2: Provide functionalities to update the information such as genomics data or ethical consent forms that are required to be modified by the authorized roles. Updated information should be transparent to the user who made the changes.
- C7.1: Implement usable strong authentication mechanisms such as two-factor authentication, biometrics authentication or public key certificates to avoid compromising the security of the platform through stealing user credentials.
- C7.2: Validate the input at least in the LIMS server to restrict any possible SQL injection, XSS or cookie/query string/HTTP manipulation.
- C7.3: The platform should classify the responsibilities and permissions of each role to avoid granting a broad range of responsibilities to each user. For example, restrict Auditor to have Administrator permissions.
- C7.4: For every attempt to access the resources ensure the identity of the user and enforce the predefined policies for that user. For example, in the LIMS platform ensure RBAC while in the HOPS enforce discretionary access control in a fine-grained approach.
- C7.5: Protect the internal services through the internal firewall that restricts the I/O web traffic to the external firewall. Using intrusion detection techniques identify the intruders and block their Internet Protocol (IP) address.
- C7.6: Do not trust the safety of user computers and keep the session state information on the server side and do not send more than an opaque session identifier to the users. Encrypt whatever that is sent to the client with the server keys for making it confidential in the client machine over secure HTTPS channels. Also, for every new session create new session information both on the server and for the client.
- C7.7: Anonymize the microdata that contains an explicit identifier of data subjects. For example, apply k -anonymity or ℓ -diversity to generalize the sensitive attributes.
- C7.8: Avoid using the secret credentials such as connection names, passwords in the source code or storing them in plain text in the platform.

- C7.9: Run the HOPS and other services in virtualized secure environments where the integrity of images are ensured prior to installation. Apply latest patches to ensure the safety of the VMs and ensure all communications between the user and LIMS server are encrypted.
- C8.1: The platform should store log files related to study data, consents management, authentication, authorization, identity and access management for all users actions. Store the log files in the high availability HOPS services. Do not modify or delete the log files.
- C8.2: Prevent the application, server, database components or the logger of the LIMS server to include sensitive credentials and information. For example default setting of the application server can be turned off to avoid printing the contents of issued queries.
- C8.3: Implement custom audit modules for consistent logging of an important feature such as a browser, OS, IP/Mandatory Access Control (MAC) addresses, timestamp, initiator of events, targets and outcome of the actions. This will ensure excessive information from the log files will not be displayed in the audit report. Additionally, the log server should be synchronized with the rest of process in the BioBankCloud to ensure time consistency in the audit reports.

5.8 Summary

Privacy-preservation for sensitive data has become a challenging issue in cloud computing. Threat modeling as a part of requirements engineering in secure software development provides a structured approach for identifying attacks and proposing countermeasures against the exploitation of vulnerabilities in a system.

This chapter described implementation of the CPTM methodology for BioBankCloud case study. It described the modeling methodology that involved applying ME to specify characteristics of a cloud privacy threat modeling methodology, different steps in the proposed methodology and corresponding products. The results demonstrate the usability of the proposed methodology. We believe that the extended methodology facilitates the application of a privacy-preserving cloud software development approach from requirements engineering to design.

The results of this threat model will be used to build a functional prototype of the BioBankCloud in Chapter 6 for compliance with the EU DPD requirements.

Chapter 6

Design and Implementation of the Secure BioBankCloud

This chapter is mainly based on publication VI. It outlines the design, implementation, and validation of a security framework for BioBankCloud, a platform that supports the secure storage and processing of genomic data in cloud computing environments.

6.1 Introduction

In this chapter, we present design, implementation and validation of a privacy-preserving framework according to the CPTM to ensure the legal processing of genomic data in cloud computing according to the DPD requirements such as lawfulness, informed consent, purpose binding, data minimization, transparency, data accuracy, data security and accountability.

The design step includes proposing a security architecture to provide a trade-off solution to meet the BioBankCloud overall goals such as flexibility in terms of the analysis being performed, scalability up to very large data sets and very large cluster setups, ease of use and low maintenance cost, strong support for data security and data privacy, and direct usability for users. Additionally, it includes a detailed description of each component in the security framework that is needed to be constructed.

To develop and build the privacy-preserving BioBankCloud the design recommendations are implemented thoroughly to avoid any possible software flaws in the proposed system. This includes a technical solution that implements the logical components and deploys them in terms of a prototype.

The rest of this paper is organized as follows. Section 6.2 explains the choice of architecture through evaluating existing security solutions and libraries. Section 6.3 sketches the design of each component in the security framework. Section 6.4 discusses the implementation of proposed design. Section 6.5 briefly describes

the validation strategy. Section 6.6 discusses how privacy requirements have been enforced. Finally, Section 6.7 summarizes our findings.

6.2 Security Architecture

6.2.1 Comparison of Existing Solutions

There are several widely used popular authentication, authorization and auditing solutions that can be used for building secure systems that are summarized as followings.

Password: A password is a simple word or a combination of letters and numbers typically between 8-15 characters that are provided by the user upon authentication. Password-based authentication enables a user to claim an identity (usually a user identifier or email address) by presenting the knowledge of a secret known only by the owner and the system. As a good practice, passwords should be as much as possible complex and random and also easy to remember by the owner.

- Pros: Very common and deployed in many systems, convenient for users, easy to be implemented, no special hardware/software requirements, can be changed at the user's request
- Cons: Users often create simple passwords or forget complicated passwords if not used frequently, not scalable in multi-server environments. In case of encrypted data, loss of password results in the loss of data, weak against several attacks such as sniffers, or crackers

Quick Response Code (QRC):¹ A QRC is the trademark for a type of matrix barcode. A barcode is a machine-readable optical label that contains information about the item to which it is attached. A QRC uses four standardized encoding modes: numeric, alphanumeric, binary and kanji to efficiently store data. A QRC consists of square dots in a square grid over a white surface. These dots are readable by a camera or scanner. The Reed–Solomon error correction algorithm is used for processing until the image is interpreted [29]. Reed–Solomon codes are error-correcting codes that were proposed by Irving S. Reed and Gustave Solomon. Reed–Solomon code belongs to the class of non-binary cyclic error-correcting codes.

- Pros: Ease of installation of new accounts, reducing account registration process, easy to implement, easy to maintain
- Cons: Difficulty for novice users to install, requires to implement custom libraries

¹<http://www.qrcode.com/en/>

Authenticator: The Google Authenticator (Authentication)² is an open-source mobile application that implements TOTP security tokens from RFC6238 [67]. It generates a 6-digit code in 30 second periods. The user supplies this code as one factor and a plain password as the second factor of authentication.

- Pros: Easy to install, easy to use by users, open-source software
- Cons: Depends on smartphones, for every login user should open the App

Smart Card: This technology delivers a credit-card sized hardware with an embedded integrated circuit that must be plugged into a reader device [189]. For example, a flash memory card to store the digital information that is protected by a PIN code. A smart card also can be a microprocessor with processing power for encryption capabilities such as public key encryption, signatures, and verification to provide secure identification [190]. Smart cards provide some mechanisms to store a user's certificate and private key.

- Pros: Limits the number of login attempts that locks the smart card, lightweight, portable and easy to use, tamper-proof identity of users, common in the governmental branches with demand on strong authentication
- Cons: The PIN can be forgotten or compromised, requires a reader device, lost smart cards can cause extra costs

USB contains a smart card and a built-in reader chip to provide functionality of both USB token and smart card. USB token delivers two-factor authentication similar to smart cards [191]. Yubikey devices³ can be considered as USB tokens which can be plugged in USB ports to automatically fill in the OTP code for each authentication attempt by pressing a button. Yubikey token doesn't have a notion of time compared to the Authenticator that every 30 seconds generates a new OTP. Instead, it uses a counter value that is embedded in the device and it increases when users press the OTP button. A Yubikey token consists of two slots. One slot can be used to store static passwords and another one can be used to generate OTP using different protocols such as Yubikey One-time Password (YOTP), OATH-HOTP or challenge-response.

- Pros: No reading device is required, can be easily plugged into the user devices, low deployment costs compared to biometric and smartcard
- Cons: Lost tokens can cause extra costs, must be always carried by the user

PKI X.509 Certificates that can be considered as an standard for the PKI to manage digital certificates and public key encryption. Another usage of the X.59 certificates is in the Transport Layer Security (TLS) protocol for secure Web and email communication.

²<https://code.google.com/p/google-authenticator/>

³<https://www.yubico.com/>

- Pros: Simple authentication without administration efforts of user passwords, scalability for the joining organizations and users, higher security than basic password authentication, capability of creating a proxy certificate for delegation purposes, support for TLS/SSL secure communication and Single Sign-on (SSO)
- Cons: Many users do not understand the purpose of certificates, certificates must be carried by users and private keys must be kept secure, certificate distribution to user and installation can be difficult, establishing revocation mechanism in case of compromised private keys

Biometrics: Biometric authentication methods are based on a measurement of the unique physiological characteristics of a user, such as a fingerprint (to replace password), face recognition, iris code and behavioral characteristics such as voice recognition, handwriting, and signature scan [192]. Generally, the identity provider stores the biometric data in the encrypted format in a database. Biometric authentication uses pattern matching of the claimed biometric characteristic against a stored template. An accurate biometric authentication balances the equation between a false negative (rejecting authorized user) and a false positive (accepting unauthorized user) rate [193].

Authentication based on fingerprints can be based on embedded sensors or special devices for fingerprint recognition. Some devices such as mice are equipped with embedded sensors which are built in 3D imaging which eliminates the need to buy standalone devices for user authentication.

Voice authentication is another form of biometric authentication which resembles “what the user is”. This form of biometric is based on voiceprint technology. A microphone captures the user’s voice and matches it with the backend for successful authentication.

- Pros: No key management issues such as secure key preserving, lost and compromised keys, secure against different threats, devices such as webcam and microphone are available in most user devices, easy to use by users
- Cons: Voice recognition may not be secure against background noise, sickness, and abnormal conditions, not perfect solutions to satisfy the false reject and false accept rate, user privacy concerns, involves cost for support and maintenance and deployment (fingerprint and iris-scan devices are costly)

Kerberos v5 [194] is an authentication protocol based on symmetric key encryption to provide authentication in client/server scenarios in open networks without exposing the shared symmetric key between the principals (client or server instances). Both client and servers mutually authenticate each other without transmission of any password in the plain or encrypted form. The Kerberos server sends encrypted tickets with the user shared a symmetric key that only can be decrypted by the user knowing the password. A ticket is a token protected by encryption that

allows a user to be authenticated without requiring her credentials, such as her password. Tickets can also be used to identify a user who has been authenticated to use SSO to avoid re-authentication for frequent access. After the authentication phase, encrypted channels will be established to provide secure communication to ensure confidentiality and integrity of the data.

In the Kerberos protocol, the KDC is a service composed of the authentication server and the Ticket Granting Server (TGS). The authentication server acts as a trusted third-party in charge of user authentication in a realm that users and servers belong. The TGS issues a Ticket Granting Ticket (TGT) during the user first authentication. There are several open source implementations of Kerberos v5 protocol, mainly such as Heimdal version 1.6⁴ and MIT Kerberos⁵ versions. Heimdal supports the Kerberos raw API, in addition to PKI, GSS-API [195]. GGSS-API provides an abstraction layer using a standard API for applications to use for the underlying protocols.

Heimdal also supports simple and protected GSS-API negotiation mechanism [196], to authenticate client application to a remote server, but neither end knows what authentication protocols the other uses. SASL [197] also implemented in Heimdal as a layer to provide pluggable authentication and data security for the existing application protocols such as Lightweight Directory Access Protocol (LDAP) [198]. Heimdal is used on several platforms and it also supports cross-realm authentication where trust between different realms is required to be established.

- Pros: Strong authentication for network-based services, cross-realm authentication using cross-realm tickets, delegation through forwardable tickets
- Cons: KDC can become a single point of failure against DoS attacks or insufficient security, brute-force attacks against weak passwords, enabling applications with Kerberos authentication can be a cumbersome task, complicated for ordinary users to set up and use

LDAP: LDAP is a client-server access control protocol for accessing directory services. LDAP defines how entities in a database can be queried and accessed. LDAP defines a simple protocol for updating and searching hierarchical directories over open networks. LDAP v3.0 supports secure communication through transport layer security (TLS v1) to encrypt the communication channel using client and server X.509 certificates by an regular LDAP connection (`ldap://:389`). Also Netscape SSL v3 can be used to establish secure connection (`ldaps://:636`). LDAP can be used to store information about users, including usernames and passwords to be validated by applications.

The LDAP schema specifies objects and attributes and their relation to be stored in the LDAP server in directories for instance “ou=roles, ou=resource, ou=projects” to provide scalability in distributed environments. Common attributes of an LDAP

⁴<http://www.h51.org/>

⁵<http://web.mit.edu/kerberos/>

entry are: distinguished name (dn) as primary keys, common name (cn), domain component (dc), and organizational unit (ou).

There are different implementations of the LDAP protocol such as OpenLDAP v.24 that is open source and widely popular⁶ or the Microsoft active directory implementation of the LDAP that can be utilized through plugins for integration with UNIX based systems⁷. OpenLDAP supports several databases as backend such as Berkeley database, which is a key/value database, hierarchical Berkeley database as default backends. Other backends such as Network Database Technology (NDB), in-memory databases. OpenLDAP offers several replication strategies such as multi-master (to allow multiple servers to act as masters), mirror mode and synchronization-replication mode.

- Pros: High availability and resilience topologies, easy to implement and backend with various databases, secure communication using TLS/SSL
- Cons: Mostly suitable for hierarchical structures, difficult to implement and deploy with applications, i.e., schemas, replication, interoperability with web development frameworks

OpenID⁸ is an open standard to provide user-centric authentication without the need for users to release their credentials such as a password or other sensitive credentials to a relying party who relies on the OpenID provider to offer services. Users create accounts with an identity provider they prefer and use them for authentication. OpenID authenticates users dynamically based on the information in the identifier to the related identity provider. Identities or subjects identifiers are represented as HTTP or HTTPS Unified Resource Locator (URL) or extensible resource identifier that can be used in different domains.

OpenID delivers federated SSO for web applications in different realms through one identifier provided by the user. Attribute exchange is an approach to fetch or store user attributes such as username, email address to RP during authentication phase based on user consent.

Provider authentication policy extension is another feature of OpenID to enforce strong authentication by relying on the party through multifactor authentication or anti-phishing authentication methods. OpenID provides a mechanism for simple registration to free users from multi-registration in different relying parties [199].

OpenID Connect is the next generation of OpenID which operates as a layer on top of OAuth v2.0 to provide passing the delegation access instead of the authentication access to different sites [200].

- Pros: Can be easily integrated with the web applications, provides user-centric approach, trust all users and no need for preconfigured trust, commercial support from major cloud vendors and social networks

⁶<http://www.openldap.org/doc/admin24>

⁷<http://technet.microsoft.com/library/bb463150.aspx>

⁸<http://openid.net/developers/specs/>

- Cons: Not interoperable with the networked services, no built-in authorization mechanisms, weak against Domain Name System (DNS) cache poisoning

Open Authorization (OAUTH) v 2.0 [201] is an open standard to provide web applications possibility of authenticating and authorizing users based on their credentials instead of a direct password. For instance, it allows users to share their private resources such as photos, calendar and contact lists stored on one site with another site without releasing their credentials such as username/password.

OAuth specification defines resource server as an entity that protects data. Users are defined as resource owner who grants or deny access to their protected data. The term client refers to the applications that operate on the data, according to the resource owner granted permission acquired by an authorization server.

Another feature of OAuth is to deliver delegated authorization to grant another application or person to operate on behalf of the data owner. Delegated authorization is done through bearer tokens which are acquired from an authorization server to be sent to other participants to act on behalf of the owner. This token is opaque for clients and can be decoded for the endpoints [202]. OAUTH provides interoperability with OpenID Connect for federated Web SSO across different realms.

- Pros: Lightweight and compatible with REST web services, strong industry support by vendors such Facebook and Google
- Cons: No support for non-web service authentication, weak against DNS cache poisoning

SAML [203] is an open standard based on Extensible Markup Language (XML) introduced in 2002 by OASIS for exchanging user authentication and authorization information between identity provider and service provider. SAML is based on the concept of assertions that can be handed to the client who created the access request, the authentication service, and the access decision point. Assertions are statements about a user that can be passed around between different partners. SAML provides a standard request/response protocol for exchanging XML messages.

SAML provides “Portable Trust” for a verified user in one domain to invoke services in another domain using SSO capabilities within a federation. Also for web services, SAML provides a means by which security assertions about messages and service requesters can be exchanged. SAML is built on XML Schema, XML signature, XML encryption, HTTP, and Simple Authentication and Security Layer (SOAP). SAML IdP and SP establish communication through exchange of SAML metadata.

The SAML specification defines assertions, protocols, bindings and profiles. Assertion (XML) is a claim, statement, or declaration of fact made by some SAML authority about the subjects. SAML protocols define a mechanism to exchange

request/response messages and data between different parties. SAML bindings provide a mapping of a SAML protocol and communication protocols such as HTTP and SOAP to send the SAML messages with SOAP.

SAML profiles provide message exchange information related to a set of functions to be used by different participants. There are several implementations of the SAML protocol such as OpenSAML, which is a set of open source C++ and Java libraries. OpenSAML 2, the current version, supports SAML 1.0, 1.1, and 2.0 [203].

- Pros: Support for strong authentication and SSO, standard message exchange protocol, extendible to support different use cases, PKI recommended
- Cons: Complexity of SAML 2.0 specification, no dynamic identity provider discovery (usually there is a trust list), major public cloud providers such as AWS and Google App engine don't support SAML

Shibboleth⁹ v2.5 is an open-source project that provides federated SSO capabilities to allow sites to make informed authorization decisions for individual access of protected online resources. The Shibboleth software builds SAML 2.0 to deliver federated SSO and attribute exchange framework through authenticating user to their home institutions.

A federation in Shibboleth is built through an agreement between different organizations where users belong. Users not belonging to a federation are not able to login to the service provider services.

The Shibboleth identity provider delivers authentication to users in addition to providing the service provider with several attribute information. Therefore, users do not have to provide manual data every time user logs in to the service provider. The service provider handles the access to a protected resource or application entry point through issuing a SAML authentication request to an identity provider selected by the user- “where are you from” concept in Shibboleth - to choose the home organization, and processing the authentication response. Shibboleth also supports SAML SSO profile with HTTP with the purpose to facilitating usage of SSO for different environments.

For networked-based services that do not rely on web services, Shibboleth has limited support through SAML enhanced client SASL and GSS-API. But none of these mechanisms are used in any production release yet¹⁰.

- Pros: Authentication and authorization based on user attributes, interoperability with Kerberos, LDAP, Java Database Connectivity (JDBC), proven solution in higher education and governmental organizations, strong authentication through smartcard/PKI certificates
- Cons: User has to choose the identity provider, costly to maintain

⁹<http://shibboleth.net/>

¹⁰<https://wiki.shibboleth.net/confluence/display/SHIB2/ECP>

eXtensible Access Control Markup Language (XACML) [204] is an XML based standard which defines a policy language, request/response protocol, and architecture that implements ABAC for access control. However, OASIS also has defined a XACML profile for RBAC [93].

XACML deals with user attributes such as subjects (users), resources (objects which are required to be accessed), actions (operations on resources such as read, write), and the environment (time, location) to create fine-grained policies.

There are several free implementations of XACML such as EGEE Argus, XACMLight¹¹, XEngine¹². The Argus authorization framework¹³ provides an attribute-based authorization decisions software for distributed systems such as web and network, compute and storage. The services are implemented based on the XACML 2.0, and use authorization policies to determine if a user is allowed or denied to perform a certain action on a particular service.

Argus consists of three main components: Policy Administration Point (PAP) to define the policies and store them, Policy Decision Point (PDP) to validate authorization requests against the XACML policies retrieved from the PAP server and Policy Enforcement Point (PEP) server to ensures the integrity and consistency of the authorization requests received from the PEP client. PEP client is a lightweight C/Java library which can be integrated easily with different applications. Argus provides interoperability with Shibboleth and SAML to exchange the XACML attributes through SAML assertions. It also provides the command line capabilities to create human readable policies, based on the simple policy language. These features make it possible to ban a user globally over all sites.

- Pros: Suitable for creating complex authorization policies
- Cons: No standard interaction between PAP, PDP, PEP and client

Virtual Organization Management Service (VOMS)¹⁴ is a centralized system to store authorization information of VO (a group of collaborating organizations) such as user roles, group hierarchies and capabilities to complement limitations of Access Control List (ACL). VOMS issues attributes based on X.509 certificate information and SAML attribute assertions¹⁵. VOMS administrator server provides interfaces based on web services to manage and keep track of user roles. VOMS can use relational databases such as MySQL and Oracle as backend.

- Pros: Rich authorization decisions based on roles for VOs, interoperable with SAML and PKI
- Cons: Single point of failure, difficult to build and maintain

¹¹<http://sourceforge.net/projects/xacmlight/>

¹²<http://xacmlpdp.sourceforge.net/>

¹³<https://twiki.cern.ch/twiki/bin/view/EGEE/AuthorizationFramework>.

¹⁴http://www.globus.org/grid_software/security/voms.php

¹⁵<https://twiki.cern.ch/twiki/bin/view/EMI/EMIVomsDocumentation>

System for Cross-domain Identity Management (SCIM)¹⁶ is an industry initiative to define a simple JavaScript Object Notation (JSON) schema and a standard based on REST API for automated user provisioning. SCIM also contains bindings to define transportation of SCIM information with other protocols such as SAML. SCIM core defines attributes such as user, group, and resource. SCIM connections can be established through secure channels using TLS/SSL and it provides operation such as GET, PUT, POST, PATCH and DELETE for identity management.

- Pros: Lightweight and easy to integrate with cloud application and services, interoperable with SAML
- Cons: Not interoperable with network-based services

Flume¹⁷ is an open source distributed log management streaming mechanism to transfer all log files from different sources to a centralized log server to be analyzed by Hadoop. Flume provides fault-tolerant, scalable and centralized management of the log files based on the ideas of flows. Flume nodes are controller or agents. Each node runs source that accepts the input message from servers or applications and sinks to redirect the messages to storage.

Channels deliver events of 4 kbyte size from source to the sink. A Flume stream defines a schema to store the output from the source in the storage where all logs are stored. In distributed settings a master node keeps track of the nodes.

- Pros: High throughput log management system, SQL-based language to create complicated queries, interoperable with Avro, log4j, Syslog, HTTP post with JSON body
- Cons: Difficult to deploy and configure

Distributed Audit Service (XDAS) provides accountability and compliance through an audit service. XDAS specification mainly defines a set of generic events across the system, a portable audit record, API for applications to submit their events to XDAS or read records from an audit trail. XDAS satisfies audit requirements of different scenarios through audit event services, audit service management, audit event management, audit log management, and audit event inquiry [205]. XDAS defines a standard record format including event, originator, initiator, target, source, and data.

In a distributed environment with several platforms, an XDAS agent collects the audit events through the API or direct import and can send them to a centralized event management service. XDAS event consumers will be able to use the XDAS service through audit event analysis service.

¹⁶<http://www.simplecloud.info>

¹⁷<http://archive.cloudera.com/cdh/3/flume/UserGuide>

OpenXDAS¹⁸ is a complement open source implementation of the XDAS specification including the client-side instrumentation and filtering. There is also the library XDAS4J which is a Java implementation of XDAS.

- Pros: Consistent auditing
- Cons: Difficult to integrate with the existing applications, might require development of the API for custom applications

CloudAudit/A6¹⁹ (The automated audit, assertion, assessment and assurance) CloudAudit/A6 is a group member of Cloud Security Alliance (CSA) to address audit and compliance for cloud providers through a specification for automated audit, assertion, assessment, and assurance API (A6). CloudAudit goal is to enable users to audit and assess remote infrastructure using platform independent API and namespaces.

According to the IETF draft version, CloudAudit provides a set of conventions, standards, and API to utilize the HTTP protocol to enable cloud users and external third parties to get automated detailed performance and security statistics about the cloud services. The CSA has released a free toolkit²⁰ for governance, risk and compliance consisting of a namespace to clarify how different compliance requirements are defined.

- Pros: A lightweight audit mechanism for cloud-based services
- Cons: No standardization exists yet

6.2.2 Proposed Selection of Components

According to the platform architectural goals and analysis, we identified the main trade-off of the platform as security/privacy vs. usability. For this purpose, a set of components that satisfy these architectural goals efficiently were selected. Figure 6.1 shows the security architecture of the platform including user machine, authentication, authorization, IAM and auditing modules.

The platform provides secure interfaces for several services such as account registration, account recovery, authentication by the Authenticator or Yubikey tokens that are present on the user's machine. After successful authentication, the user submits a task that will be validated by the authorization component. Depending on the task LIMS or Hops authorization system will enforce the predefined permissions. Authentication, authorization, and IAM events will be registered in the log repository and will be available for auditing.

¹⁸<http://xdas4j.codehaus.org>

¹⁹<http://tools.ietf.org/html/draft-hoff-cloudaudit-00>

²⁰<https://cloudsecurityalliance.org/research/cloudaudit/>

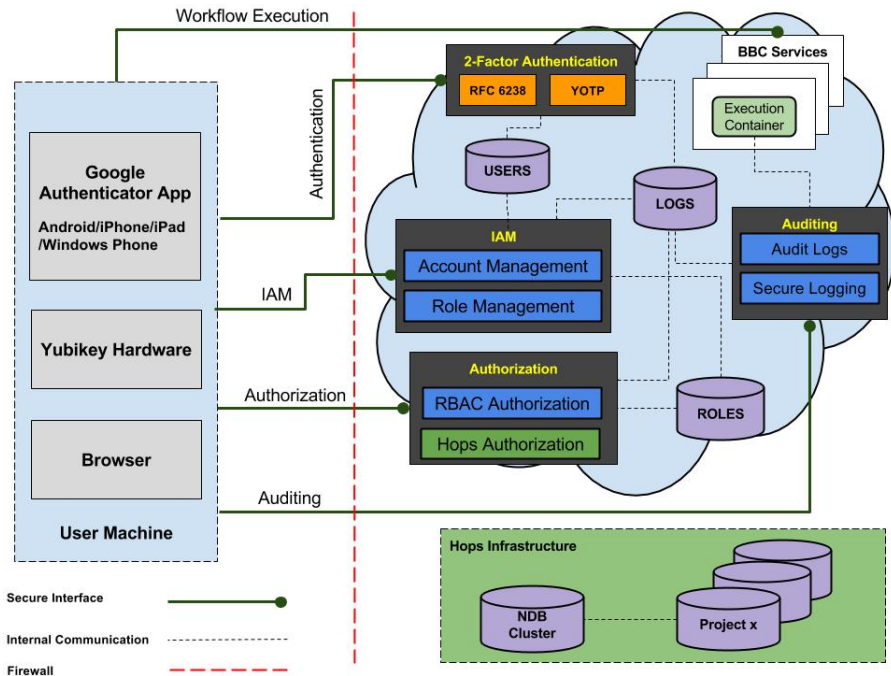


Figure 6.1: Security architecture of the BioBankCloud including various security modules

6.3 Design

6.3.1 Assumptions

In our design, we assume only a limited number of certified personnel have access to the cloud provider’s physical infrastructure. Internal or external auditors ensure that cloud provider implements service organization control (SOC 1/SSAE 16/ISAE 3402) [206] requirements for individual controls to the infrastructure.

6.3.2 Identity and Access Management

The BioBankCloud IAM system plays a critical role for user management to realize the architectural goals of the system in terms of cost-reduction and maintainability. The proposed system contains people, procedures and assets to define the right access at the right time to the right people. This component provides several functionalities for registering new accounts, verifying, activating or terminate them. Figure 6.2 shows the process of user registration and activation in the system.

Each element of this state machine has been described in the following.

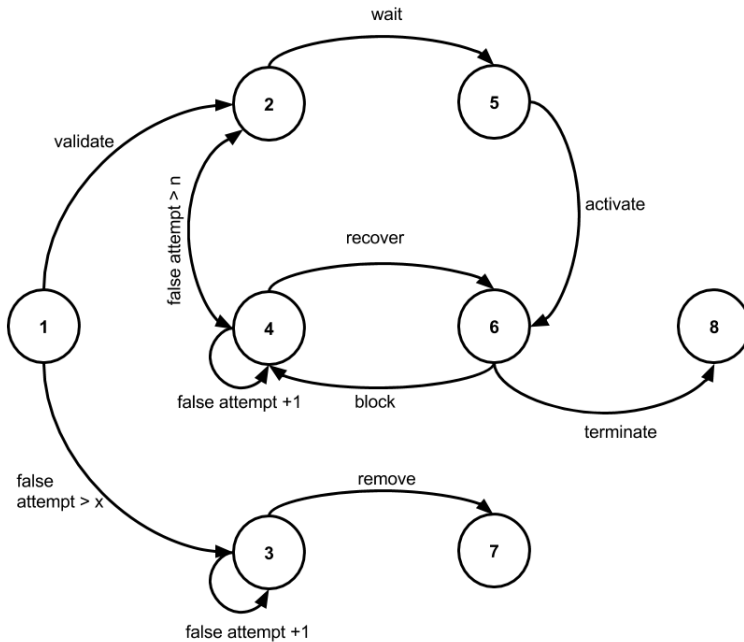


Figure 6.2: Identity lifecycle in the BioBankCloud

- **New (1)**: Set the status of new accounts requests to new. In this step, accounts are considered as inactive.
- **Verified (2)**: New account requests are validated by the actual user who created the request.
- **Spam (3)**: To distinguish between verified accounts and suspicious requests.
- **Blocked (4)**: User with this status can not access the resources. This can be due to the suspicious behavior of users. For example multiple false login attempts.
- **Pending (5)**: Upon successful reset of passwords status of blocked will be changed to pending. The user will be able to reset the temporary password using the status.
- **Active (6)**: When user's request is approved by an administrator to use the platform.
- **Deleted (7)**: Administrator decides to delete spam accounts from the system as such accounts are not associated with an actual user.

- **Closed (8)**: Users that are no longer granted to access the platform. Users with this state, can not log in or change profile information. Either an account holder terminates her account or an administrator closes the account.

When users are created, a system username will be assigned to them for identification. *Identification* can be referred as the first step to establish a relationship with the users and provide services to them. There are several identification methods such as username, account number, email and IP/MAC addresses for computing devices. A good identity scheme usually provides uniqueness (a unique identifier in different domains), non-descriptiveness (not to disclose extra information such as user role or plain names) and secures issuance (logging and documentation of identities) [207].

To meet the demands for HOPS authorization [150] a unique alphanumeric username with length of 8 characters was designed that is compatible with the POSIX semantics. Every time new user registers using the IAM service, a new username will be generated and stored in the user credential store. The username scheme is composed of 3 letters that indicate the prefix of the institution name, and the other 5 digits are the user identifiers. For example, *meb10003* demonstrates a user from the MEB organization with user identifier 10003.

This scheme makes the system flexible to link this username with more convenient user identifiers such as email address. In such scenario, the IAM system will map the POSIX username with the email address of user without posing any usability issue for the user. Besides, when users change their email addresses, still integrity of audit logs are preserved. To hide this complexity, we link the POSIX username with the user's actual email address so that users do not require to remember an additional element for authentication.

6.3.3 Authentication

Authentication is the process of validating a user's physical claimed identity or the digital identity of a process or a computer. User authentication can be categorized into three main categories: authentication by knowledge - i.e., what the user knows such as a password (memo metrics), authentication by possession - i.e., what the user has, such as a smartcard token (cognometrics), and authentication by characteristics - i.e., biometrics such as fingerprints, retinal, iris, voice, face, handwritten (biometrics) [191]. These authentication approaches can be combined or used separately, depending on the demanded level of functionality and security. The BioBankCloud supports strong two-factor authentication using *Mobile* and *Yubikey Tokens*. This provides a trade-off solution between security and usability by allowing users to select a convenient authentication method.

We provide a custom authentication realm that delivers both functionalities of the Authenticator and Yubikey device tokens. Users send authentication requests via a browser to the *authentication* module or to the security policy domain (*custom realm*) [208]: *TOTP* and *YOTP* plugins.

Figure 6.3 demonstrates the custom login context that reads the Java Authentication and Authorization Service (JAAS) configuration to validate the authentication requests using TOTP and YOTP algorithms. The login modules contain interfaces to access the credentials using the JDBC driver. The JAAS configuration represents the configuration of the login modules to be defined by the BioBankCloud platform.

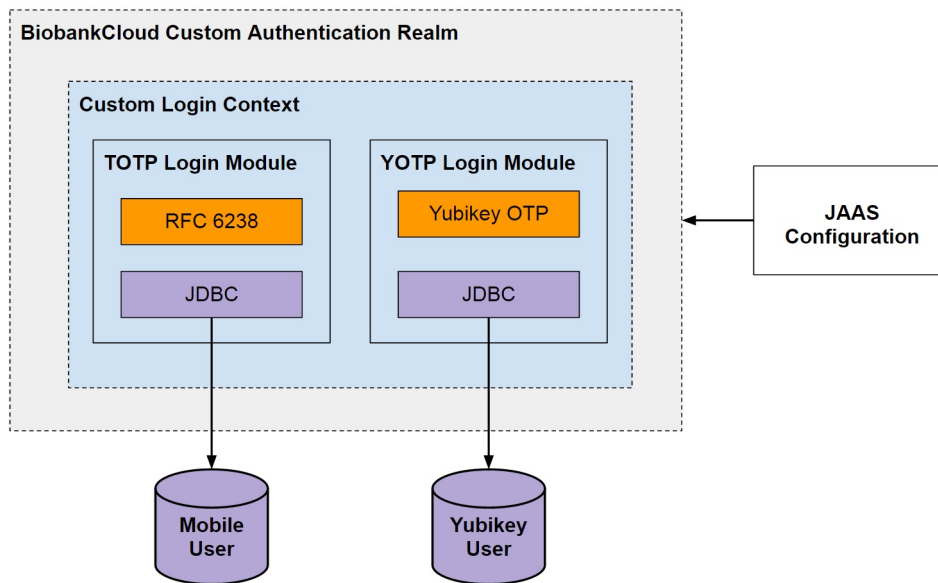


Figure 6.3: Custom authentication realm to support authentication for users with and without mobile devices.

A typical scenario for mobile account registration and authentication is as follows.

1. The user installs the authenticator in a mobile device.
2. The user opens the mobile registration page and creates an account request by entering organizational information, giving consent to the platform ToS and creating a plain password (a minimum 6-character alphanumeric string).
3. The platform creates a guest account and sends a QRC to the user's browser.
4. The user scans the QRC code with the mobile device and the authenticator configures the account (see Figure 6.4).

5. The platform sends a validation email to the user's email address. The user verifies her email address through clicking an URL.
6. An administrator activates the account on the platform and a notification is sent to the user by email, as shown in Figure 6.11.
7. The user opens the mobile login page and enters the username and plain password as one factor of the authentication.
8. The user opens the Authenticator and enters the OTP in the login page as the second factor of the authentication.
9. The user issues an authentication request and the platform verifies the plain password and the OTP.



Figure 6.4: Scanning the QRC using the Autneticator App in smartphones

For users without mobile devices, the platform offers Yubikey authentication through YOTP. The YOTP (44 characters password) consists of two parts: the first 12 characters indicate the public identifier of the Yubikey token [209]. The remaining 32 characters are a unique code for each OTP. A Yubikey token generates the OTPs through a push-button. Generated YOTPs are sent as emulated keystrokes via the keyboard input path, thereby allowing the OTPs to be received by any text input field.

A typical scenario for Yubikey account registration and authentication is as follows.

1. The user opens the Yubikey registration page and creates an account request by entering organizational information, giving consent to the platform ToS and creating a plain password (a minimum 6-character alphanumeric string).
2. The platform creates a guest account to be approved by an administrator.
3. The platform sends a validation email to the user's email address. The user verifies her email address through clicking an URL.
4. The administrator activates the account (see Figure 6.6) and programs a Yubikey token through the personalization Graphical User Interface (GUI) [209]. Afterward, the administrator sends the Yubikey token through a trusted postal system to the end user.

The screenshot shows the HopsWorks registration interface. At the top, the HopsWorks logo is displayed. Below it, the heading 'SIGN UP TO GET ACCESS.' is centered. The form is organized into several sections:

- Personal Information:** Fields for 'First Name' and 'Last Name'.
- Contact Information:** Fields for 'Email address' (with an envelope icon) and 'Phone Number'.
- Security:** Fields for 'Password' and 'Confirm Password', both with lock icons.
- Security Question:** A dropdown menu for 'Security question' and a text input for 'Security answer'.
- Authentication Choice:** Two radio buttons labeled 'Mobile' and 'Yubikey'.
- Terms and Conditions:** A checkbox labeled 'I agree with the terms'.
- Registration Action:** A prominent blue button labeled 'Create account'.
- Existing Users:** A link 'Have an account?' above a 'Sign in' button.

Figure 6.5: Account registration in the AngularJS frontend

5. The user receives the Yubikey and inserts it into the client machine's universal bus port.
6. The user opens the Yubikey login page and enters the username and plain password as one factor of the authentication.
7. The user pushes the Yubikey button and an OTP will be redirected to the OTP input field of the authentication page as the second factor of authentication.
8. The user issues an authentication request and the platform verifies the plain password and the OTP.

6.3.4 Authorization

Authorization is the process of granting or denying access to the platform resources based on the identity of users. An authorization module enforces security policies that are configured for each role in the active security domain where authentication is performed.

Personal Info

Name	Alice Wilson	Member Since	Fri Apr 24 14:28:13 CEST 2015
Email	gholami@pdc.kth.se	Phone	-
Organization	KTH	Orcid ID	-

Contact Info

C/O	-
Street and number	Teknikringen 14, 5tr -
Postal Code	10044 Stockholm
State	-
Country	Sweden

Yubikey PublicId *

Yubikey Secret *

Figure 6.6: Yubikey accounts provisioning

The authorization process checks permission rights when an authenticated user requests access to a service. The BioBankCloud deploys a flexible RBAC model to ensure confidentiality and integrity of data. The RBAC model contains information about the potential roles of individuals within the organization and the associated levels of access to services, as shown in Table 6.1. The platform roles are categorized as administrator, auditor, data provider, guest, and researcher. This role model can be extended for new requirements. The definition of each role is as follows.

- **Administrator** a group of users who acts as the platform manager and Ethics Board.
- **Auditor** a group of users with access to audit trails for auditing.
- **Data Provider** a group of users who create studies, upload data and assign members to studies.
- **Guest** general visitors to the platform who are able to request an account to use the services.

- **Researcher** users of the platform that can join a study to run workflows. Researchers also can become data providers through creating a new study and uploading data to the platform.

The authorization system retrieves the groups' information through the custom authentication realm for users with the valid authenticated sessions. For example, when a user is authenticated, a permission check retrieves all the user's related groups. If the requested action is permitted on a service or a resource, the user will be granted access.

Assume that Alice and Bob are two authenticated users in the system. Alice needs to enable Bob to access her data, as shown in Figure 6.7. For this purpose, the following actions are taken by Alice, Bob, and the platform.

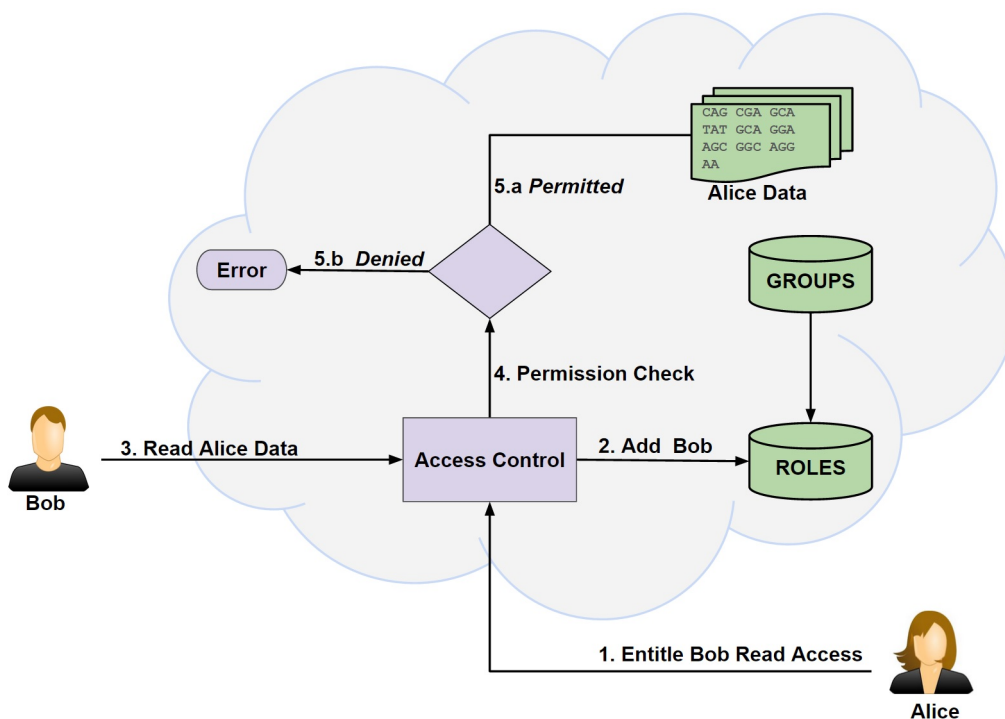


Figure 6.7: BioBankCloud authorization system to enforce permissions to access study data.

1. Alice, as study data owner, gives Bob read access to her study.

Service	Administrator	Auditor	Data Provider	Guest	Researcher
Audit Management	R	R			
Anonymization Service			X		X
Platform Public Pages	C,R,U,D	R	R	R	R
Privacy Management	R,U		C,R,U,D		R
Study Audit Trails	R	R	R		R
Study Browser	R		C,R,U,D		R,U
Study Data	R,U,D	R	R,U,D		C,R
Study Members	R				R
User Administration	R,U,D	R	R		
Workflow Execution			C,R,U,D,X		C,R,U,D,X

Table 6.1: Access control table to define the permissions for each role in the platform in regard to using the BioBankCloud services. For example, a researcher can *create* (C) a new study and will be assigned the data provider role afterwards. Then, as a data provider, the user will be able to *read* (R), *update* (U) and add new members or *delete* (D) or *execute* (X) the study.

2. The access control component updates the **ROLES** database where users are mapped to **GROUPS**. In this example, Alice and Bob are respectively mapped to the Administrator and Researcher groups.
3. Bob initiates a read request to access Alice's study in the cloud.
4. The access control component enforces the existing policies for Bob's request through a permission check (which will have returned either permit or denial).
5. The platform enforces the results of the permission check as permitted or denied:
 - a) If Bob is authorized to access Alice's study, he will be permitted to perform a read operation.
 - b) If Bob's operation is not permitted on Alice's study, he will be denied access to Alice's data.

The proposed RBAC model has provided the capability for the Hops platform to ensure authorized access in multi-tenant Hadoop environment through a DAC model [150].

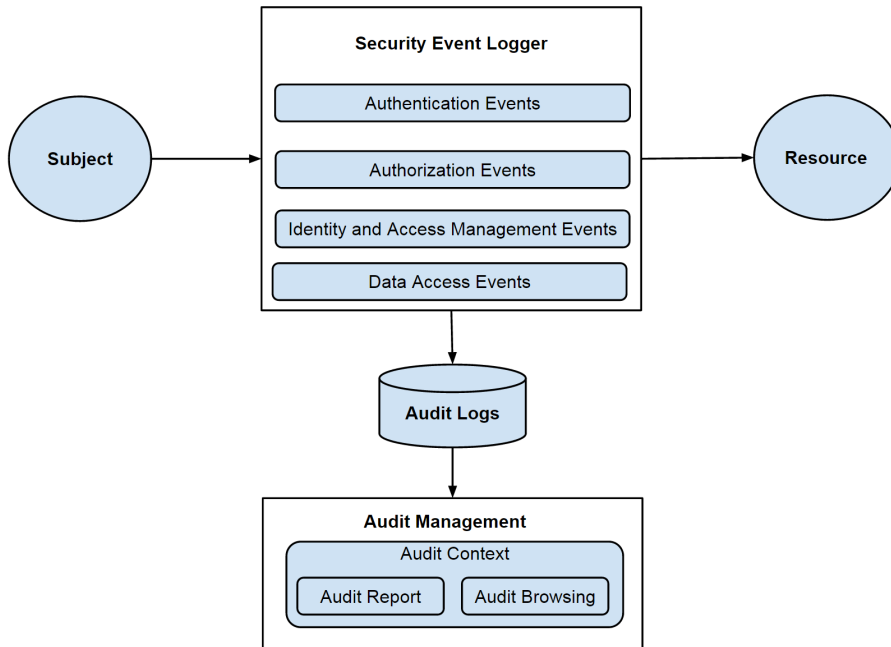


Figure 6.8: Audit system

6.3.5 Auditing

Auditing is the process of providing proof for resource usage in the platform, for example, giving details of who has accessed each resource and what operations are performed during a given period of time. Auditing helps to ensure that users are accountable and also helps to detect unauthorized attempts to access the resources. The auditing component stores the audit trails with timestamps in the audit storage. All users are assigned a unique POSIX compatible user identifier that is an 8-character length alphanumeric username to run the actual workflows in Hadoop environment.

The Audit context component provides secure logging and audit trail browsing for the authorized roles defined in Table 6.1. A user with different roles might have different interests and choose different contexts to get information from the audit trails. For example, a data provider may only want to audit the usage of a particular study that belongs to him/her, while an auditor may need to audit all the user administration and data access events with different audit options.

Security event logger provides the following security events and pushes the messages to the audit logs data store:

- **Authentication Events:** Authentication requests that a user initiates to

access to the platform are recorded by the security event logger. The log events include information such as user identifier, IP/MAC address, epoch, browser, OS, and the result of the authentication action.

- **Authorization Events:** Authorization requests that a user initiates to access to a resource in the platform are recorded by the security event logger. The log events include information such as user identifier, IP/MAC address, epoch, role, browser type, OS version, and the result of the authorization action.
- **Identity and Access Management Events:** Events related to identity and access management as following along with IP/MAC address, epoch, browser type, OS version, are stored by the security event logger.
 - Profile update: When a user updates her information in the platform
 - Password Change: Information regarding the password change by a user
 - User Management: Changing the user’s account’s status by the administrator role
 - Security Questions change: When a user changes the current security question for account recovery purposes
 - Mobile Requests: When a new user makes a mobile account request
 - Yubikey Request: When a new user makes a Yubikey account request
 - Role Assignments: Entitlement or removal of roles/permissions to users
 - Lost Devices: Yubikey or compromised/lost mobile accounts

Privacy and Data Access Events:

- Retention Period: Changing the study retention period by the data provider
- Consent Upload: When data provider uploads new consent or ethical approval forms
- Study Status: When administrator role updates or changes status of a study
- Consent Status: When administrator role approves or rejects uploaded consents by the data provider

The audit management component provides interfaces for the administrator, auditor, and data provider to generate audit reports or browse the audit trails using different contexts.

A study consent is approved by an administrator who acts as the platform manager and the Ethics Board to ensure legal processing of genomic data. The status of a consent can be *approved*, *rejected* or *not specified*, for example, someone with an administrator role might reject consents without legal basis or informed

consent in order to stop users with data provider and researcher roles from running experiments. The “not specified” status is used to indicate that a consent form is waiting to be approved by someone with an administrator role.

Privacy management enforces the requirement to provide evidence of consent to allow a data provider to share data or run experiments. The privacy management component controls the granular consent of uploaded genomic data since each study has its own privacy settings. The data provider uploads the consent form after uploading the study data.

6.4 Implementation

This section describes the implementation of the BioBankCloud security framework including extended libraries, IAM, QRC, custom two-factor authentication, authorization, auditing and privacy control components. We published the implementation as open-source in this repository [210], however we are currently improving the security feature continuously under this repository [211].

6.4.1 The Middleware and Libraries

To meet the privacy requirements of the BioBankCloud, we implemented a platform-independent security framework using Java Enterprise Edition (enterprise edition)²¹. The proposed solution includes several plugins and modules embedded in the security framework, as shown in Figure 6.1. The logic is implemented using the Java Server Faces (JSF) version 2 model-view-controller pattern. We also use Enterprise JavaBeans (EJB)²² version 3 technology that is the server-side component architecture of Java Enterprise Edition. EJB facilitates the development of distributed, transactional, secure and portable applications based on Java technology. The GUI for various functionality is implemented using Primefaces 5.1²³ however we re-implemented user registration, privacy settings and login GUI using AngularJS version 1.3.x²⁴ due to the platform constraints.

6.4.2 Identity and Access Management

The IAM implementation supports two groups of users: users with smartphones and Yubikey users who don't use smartphones (see Figure 6.5). There is another alternative for organizations that do not need strong security to allow regular authentication based on username/password that can be switched on on-demand.

Users are registered after an agreement to the BioBankCloud simplified ToS that aims to articulate different rights and responsibilities between the users and platform. After registering of accounts users are identified through unique email

²¹<https://docs.oracle.com/javaee/7/tutorial/>

²²<http://www.oracle.com/tech/network/java/javaee/ejb/index.html>

²³<http://www.primefaces.org/>

²⁴<https://angularjs.org/>

addresses for better usability. However, we keep the 8-character username for the internal job execution and consistency of event logs for auditing/accounting purposes.

- **Mobile Accounts:** When a user with smartphone registers an account, a random symmetric secret key is generated (see Listing 6.1). This code either can be sent in encrypted human readable or a machine readable format. We sent this code a QRC over HTTPS so user scans it easily through the Authenticator.

Listing 6.1: Random secret key algorithm to build the QRC

```
public static String calculateSecretKey() throws
    NoSuchAlgorithmException {
    byte[] secretKey = new byte[10];
    // use SHA1PRNG to setup a random seed
    SecureRandom sha1Prng = SecureRandom.getInstance("SHA1PRNG");
    sha1Prng.nextBytes(secretKey);
    Base32 codec = new Base32();
    byte[] encodedKey = codec.encode(secretKey);
    return new String(encodedKey);
}
```

We implemented a customized QRC library that contains usernames and their association information. Users only require facing their mobile devices with the Authenticator towards the generated QRC. This will load the account information automatically to the scanning device. The QRCs are presented in portable network graphics with the size of 200 * 200 pixels (see Figure 6.4).

Listing 6.2: QRC algorithm

```
public static byte[] getQRCodeBytes(String user, String host,
    String secret) throws UnsupportedEncodingException,
    IOException,
    WriterException {

    // Format of the QR code
    String chl = "otpauth://totp/" + user + "?secret=" + secret +
        "&issuer="
        + host;

    // Build a stream content to be loaded by user mobile
    ByteArrayOutputStream stream =
        QRCodeGenerator.qrCodeURLFormat(chl).
        qrCodeStream();
}
```

```

    BufferedImage bufferedImg = new BufferedImage(100, 25,
        BufferedImage.TYPE_INT_RGB);
    // Build portable network graphics to be sent to user
    Graphics2D g2 = bufferedImg.createGraphics();
    ImageIO.write(bufferedImg, "png", stream);

    return stream.toByteArray();
}

```

- **Yubikey Accounts:** To register a Yubikey account, the user enters the personal security credentials and organization information. Platform register such data in *users*, *address*, *organization* and *yubikey* tables in the credential repository. The platform administrator will be able to see the new request and issue an account according to the YOTP - a Yubikey implementation of HOTP.
- **Validation Request:** After submitting a new account request (for both mobile and Yubikey), a validation request is generated and sent to the email that was provided during the registration phase. The validation email contains a URL with a 64-characters random string. This URL contains the context, a key/value pair indicating username and associated activation key. The username in this URL is used for user lookup in the credential repository to validate the code. For example, a validation email that contains a URL with a 64-characters random string *40f65087-3712-442f-b859-9100c88be7d13b731b51-9400-4ff2-938d-efbf* that is appended to a POSIX compliant username (*meb10002*). This 64-characters random string makes it nearly impossible for a spammer or an attacker to register an account with someone else's email address. We set up a threshold so that if there are more than a certain number of false attempts to validate an email address, the account request will be marked as spam. When the user clicks on the validation link, the platform considers the registered account as a valid request. This step makes the request visible for an administrator to approve the new account request as a final decision.
- **Account Recovery:** Account recovery functionalities such as resetting forgotten passwords, compromised mobile accounts or lost Yubikey devices were also implemented in the IAM system, as shown in Figure 6.9. Depending on the issue for login to the platform, a user may select an option in such cases when their normal password is forgotten or their mobile or Yubikey device is compromised somehow. For the forgotten passwords, the user may reset the normal password by supplying the answer to the security question that was provided during the registration. After successful authentication, a temporary code is sent to the user's email address and after the first login to the platform, the user is required to change the password for security reasons. If user's mobile device is compromised, after authenticating the user with her

normal password, a temporary random code is generated, as shown in Listing 6.3. This random password is sent to the user's email. This code is required to be entered in the account recovery window by the user.

Listing 6.3: Random Password Algorithm

```
public static String getRandomPassword(int length) {

    String randomStr = UUID.randomUUID().toString();
    while (randomStr.length() < length) {
        randomStr += UUID.randomUUID().toString();
    }
    return randomStr.substring(0, length);
}
```

The platform invalidates the previous account information. In this way, an adversary will not be able to use the compromised credentials. After validation of the temporary code, the platform issues a new QRC to be scanned by user's mobile Authenticator and set up a new account. Figure 6.9 presents the account recovery functionalities.

Figure 6.9: Account recovery options to be selected for resetting the users accounts

- **User Provisioning:** Upon login to the platform, administrator roles will be notified of the new requests through an alarm window. The alarm window redirects the administrator to an associated functionality (see a list of options in Figure 6.10).

The administrator may decide to activate or delete new requests in case of being spam accounts. The administrator can verify the identity of requests

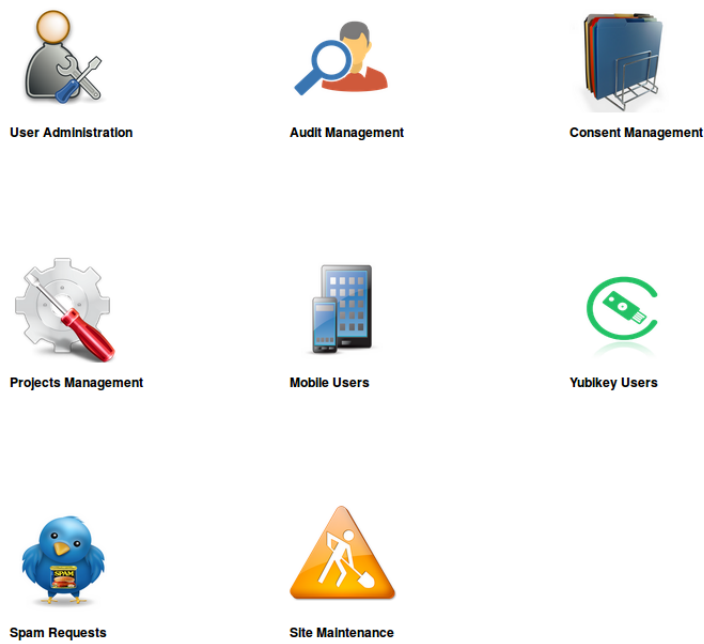


Figure 6.10: Accounts management functionalities to add/remove roles or changing the users status

in several ways based on the supplied information by the user. Figure 6.11 shows how an administrator enables a new mobile account.

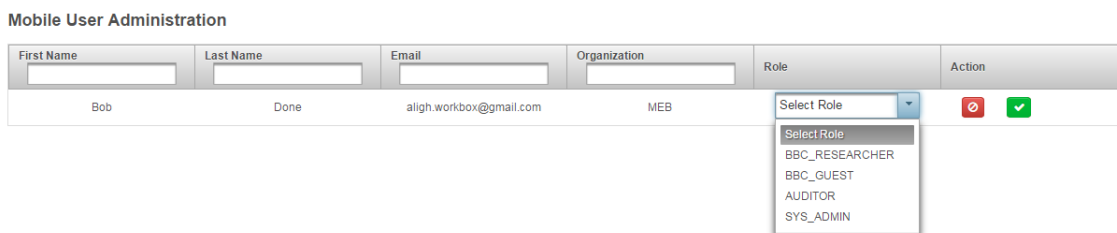


Figure 6.11: Activation of new incoming user accounts requests

- **Profile Information:** As shown in Figure 6.12, the user may change the personal information, address, organization, password, security question or terminate the account. The personal, address and organization information can be changed in place using the Primefaces methods.

Personal Info

Name	Mr Admin Admin	Member Since	Mon Apr 13 14:34:46 CEST 2015
Email	admin@ki.se	Phone	04672223200

Account Info

Current BiobankCloud Roles	BBC_GUEST	SYS_ADMIN
Last Login	Tue Apr 14 13:47:07 CEST 2015 from 127.0.0.1 with Firefox	
Last Password Changed	Mon Apr 13 13:21:00 CEST 2015	

Edit Personal Info			
First Name	Admin	Last Name	Admin
Title	Mr	Phone	04672223200
Edit Organizational Info			
Organization	KI	Department	MEB
Website	www.ki.se	Organization Phone	+4670080010
Contact Person	Admin Admin	Fax	+4670080015
Contact Person Email	admin@ki.se		
Edit Address			
C/O	Admin Office	Street and number	Teknikringen 14
Postal Code	10044	City	Stockholm län
State	Stockholm	Country	Sverige
Security Settings			
Change Security Question	Change		
Change Password	Change		
Close My Account	Deactivate My Account		

[Back to main page](#)

Figure 6.12: User’s profile with the functionalities to change information, security credentials or terminate the account

Other functionalities such as changing security credentials or account termination require multiple steps that we reduced as much as possible for usability reasons. Because users may get bored by interacting frequently through clicking and browsing different pages. Even we further explored the possibility to replace the whole profile interactions to be implemented in AngularJS. Figure 6.13 shows different functionalities that are available to users for changing their credentials using the preliminary JSF implementation.

- **Role and Access Entitlement:** When users are activated in the platform there may be the demand to change the roles within the organization or block or deactivate accounts for any reason. These functionalities have been imple-

Changing security question
Changing password
Closing account by user

Figure 6.13: Steps to change password, security question or terminate the account in the user' profile

mented, as shown in Figure 6.14. The administrator can select a user and then a list of user's existing roles, status, last login to the platform (including IP addresses/ browser, and OS type) will be displayed. The administrator may decide to add or remove more roles or change the status accordingly. This information will be displayed for each view using JSF ViewScoped that is valid for each page navigation.

6.4.3 Custom Authentication Realm

The implementation of the authentication systems contains a custom realm that extends the existing Glassfish JDBC realm. For this purpose, we implemented a plugin module called *CauthRealm* that contains two main components for authenticating mobile and Yubikey users. This module supports TOTP and YOTP methods according to the BioBankCloud authentication demands. Algorithm in 6.4 is the implementation of authentication functionality that intercepts the incoming events from the REST login calls from the LIMS GUI.

Listing 6.4: Authentication of users in the Customized Realm

```
public String[] authenticate(String username, String password) {
```

Personal Info

Name	Lora Dimitrova	Member Since	Thu Mar 19 19:56:57 CET 2015
Email	lora.dimitrova@charite.de	Phone	04672223200
Organization		Orcid ID	

Contact Info

C/O

Street and number

Postal Code/City

State

Organization Phone

Fax

Contact Person

Department

Website

Account Info

Current User Status	ACCOUNT_ACTIVE
Account Type	Mobile Account
Current BiobankCloud Roles	BBC_GUEST SYS_ADMIN
Last Login	from with
Last PasswordChange	Wed Mar 18 18:21:33 CET 2015

Change Settings

[Back to User Management Page](#)

Figure 6.14: Adding/removing roles or blocking/activating/deactivating user accounts

```
String[] groups = null;
// check if Yubikey login request
if (password.endsWith(AuthenticationConstants.YUBIKEY_USER_MARKER)) {

// extract the actual YOTP
String hpwd = password.substring(0, password.length()
    - AuthenticationConstants.YUBIKEY_USER_MARKER.length());
// authenticate the request
if (isValidMobileUser(username, password) ||
    isValidYubikeyUser(username, hpwd)
) {
    groups = findGroups(username);
    groups = addAssignGroups(groups);
    setGroupNames(username, groups);
}
```



```

    }
}
return groups;
}

```

Figure 6.15 shows the login page for user authentication.

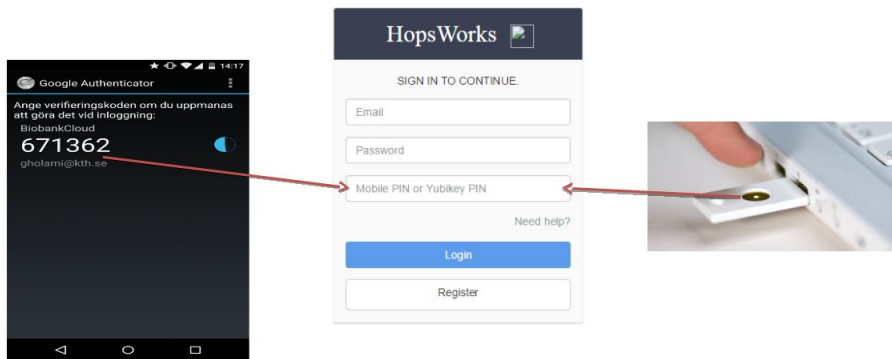


Figure 6.15: User authentication login page

Mobile Authentication. A mobile user needs to install the Authenticator App. The supplied credentials will be validated in the backend using the Mobile OTP validator. For this purpose, the received OTP will be decoded using the Base32 encoding of the symmetric secret key in the users table. We apply a variance, i.e. 5 seconds for the code verification. The verification method loads the HMACSHA1 secret and decrypts the OTP for verification.

Yubikey Authentication. A Yubikey user inserts the token into a USB and then pushes the button to generate an OTP. For every attempt, a new code is generated as below, where the first 12 characters **fifjgjkhchb** resemble the public identity of the Yubikey token. The OTP part comprises 128-bit AES encrypted information of 32 characters with Modhex encoding. This encryption is done through the embedded symmetric AES key of the Yubikey device.

- **fifjgjkhchb***irdrfdnlngghfgrtnnlgedjlftrbdeut*
- **fifjgjkhchb***gefdkbbditfjrlniggevfhenublfnrev*
- **fifjgjkhchb***lechfkfhiiuunbtvngihdfiktncvlhck*

Our validation module converts the received string to a byte string to be decrypted using the symmetric 128-bit AES key. This will be fetched from the credential store (*yubikey* relation in Listing A.1). Then the string's checksum will be

checked and if not valid, the OTP will be rejected. As next step, the non-volatile counter will be compared with the existing value in the *yubikey* table. If it is lower than or equal to the stored value, the received OTP will be rejected. If greater than the stored value, the received value is stored and the OTP will be validated.

6.4.4 Authorization

We implement the authorization system using the declarative security in the business method of an enterprise bean class that specifies method permissions. To enforce these rules, after successful authentication, the role information of the target user will be available for authorization of users. All users in the platform may be assigned single or multiple roles as described in Table 6.2. Such information is stored in the *people_group* relation, as illustrated in Appendix A.1.

Table 6.2: Implementation of the BioBankCloud roles

Group Name	Group Description	Identifier
BBC ADMIN	Data Owner	1001
BBC RESEARCHER	Users to run experiment	1002
BBC GUEST	New users	1003
AUDITOR	To audit the platform	1004
SYS ADMIN	Platform administrator	1005
BBC USER	Registered users in the system	1006

We specified the resources to protect, such as a LIMS URL or an EJB method, and a logical role that has access to the resource in the standard deployment descriptors (i.e., *web.xml* for the JSF modules or annotation for AnuglarJS REST calls). Then we mapped the logical roles to enforce the permissions that were defined in Table 6.1. For example, Listing 6.5 shows that only an administrator is allowed to access the user management functionalities.

Listing 6.5: Logical role mapping in the deployment descriptor to enforce LIMS permissions

```
<security-constraint>
  <web-resource-collection>
    <web-resource-name>User Management</web-resource-name>
    <url-pattern>/security/protected/admin/*</url-pattern>
  </web-resource-collection>
  <auth-constraint>
    <role-name>SYS_ADMIN</role-name>
  </auth-constraint>
</security-constraint>
```

Furthermore, this authorization model can be extended to MAC or DAC to support Hadoop job submission scenarios in multi-tenant environments, where a DAC model is developed to assign members to a specific data set while protecting RBAC model permissions [150].

Upon successful login to the platform, there is a policy decision point that enforces the BioBankCloud roles. For example, if an auditor is authenticated, she will be redirected to the auditing related functionalities. This is done in parallel with a filter that we implemented to periodically checks the user roles and redirect the HTTP requests to authorized resources accordingly.

6.4.5 Privacy and Ethical Settings

Privacy management includes functionalities to upload different consent forms and ethical approvals.

Consent Forms: The platform accepts the following consent and ethical forms that are uploaded by the data provider: *Standard Information About Consent (SIAC)*, *Standard Research Project Approval (SRPA)*, and *Standard Information on Non-Consented (SINC)* data. When this information is uploaded, they will be visible to the administrator for final approval. We initially implemented the front-end in Primefaces and JSF, however, we changed the GUI to AngularJS to meet the platform requirements.

Figure 6.16 demonstrates the initial GUI of the privacy settings for a study called BRCA. BRCA includes two BRCA1/BRCA2 genes that are associated with breast and ovarian cancers. A data provider modifies the privacy settings of the BRCA study that aims to sequence BRCA1/BRCA2 genes in a large cohort of women to identify new pathogenic mutations. In this example, the data provider can change the privacy settings through two panels: consent and audit. The consents (top panel) contains interfaces to upload new consent forms or renew an existing consent. On the right-hand side of this panel, the data provider updates the retention period. The study owner who acts as a data provider can search audit trails in the audit panel. There are filters in the audit panel to make it possible to search in audit trails based on username, role, epoch, and action.

Retention Period: As shown in Figure 6.16, a data provider may update the data retention period accordingly. We set the initial retention period to a reasonable long period when data are uploaded by the user. However, this deadline is demanded to be set by the data provider later. When the retention date is passed, the administrator will be notified through the panel in Figure 6.19 to delete the datasets by pressing the delete button. This event triggers a method in the platform that removes the information from the HOPS data storage.

Ethical Approval: Each project has an ethical status that is approved by the administrator when consent forms have been reviewed. To achieve this functionality, we implemented the consent management panels as illustrated in Figure 6.17 to allow fine-grained management of consents for each project. This implementation

BRCA Study...

Show Team Browser Cuneiform Study info Samples ADAM Privacy

Consent and Ethical Approval

Consent Status: PENDING Retention Period: Thu Nov 30 00:00:00 CET 2017 Update Period

brca_consent.pdf

Mon Apr 13 00:00:00 CEST 2015

PENDING

new_brca_consent.pdf

Fri Apr 24 00:00:00 CEST 2015

PENDING

ethical_approval.pdf

Fri Apr 24 00:00:00 CEST 2015

PENDING

brca_nonconsent.pdf

Fri Apr 24 00:00:00 CEST 2015

PENDING

Audit			
Name	Email	Action Date	Action
Ali Gholami	gholami@kth.se	Fri Apr 24 12:11:10 CEST 2015	changed role of lora.dimitrova@charite.de to MASTER
Ali Gholami	gholami@kth.se	Fri Apr 24 12:11:05 CEST 2015	changed role of lora.dimitrova@charite.de to AUDITOR
Ali Gholami	gholami@kth.se	Fri Apr 24 12:11:03 CEST 2015	changed role of lora.dimitrova@charite.de to MASTER
Ali Gholami	gholami@kth.se	Fri Apr 24 12:11:00 CEST 2015	changed role of lora.dimitrova@charite.de to RESEARCHER
Ali Gholami	gholami@kth.se	Mon Apr 13 13:30:09 CEST 2015	removed study
Ali Gholami	gholami@kth.se	Mon Apr 13 13:29:32 CEST 2015	added new member lora.dimitrova@charite.de
Ali Gholami	gholami@kth.se	Mon Apr 13 13:29:20 CEST 2015	created new study

Figure 6.16: Controlling privacy settings including uploading consent forms or updating the retention period of data by the data owner

also includes filtering of the information according to a specific project or consent information.

Study and Consent Management

Study Management Consent Info New Consents Expired Studies

Study Owner	StudyName	Study Status	Retention Period	Action
gholami@kth.se	BRCA	PENDING	Thu Nov 30 00:00:00 CET 2017	<input type="button" value="⊘"/> <input type="button" value="✓"/>

Figure 6.17: Reviewing overall project status

Project or study consents are approved by an administrator who acts as the platform manager and Ethics Board to ensure legal processing of genomic data. Status of consent can be *Approved*, *Rejected* or *Pending*. For example, administrator role rejects consents without legal basis or informed consent to restrict data provider and researcher roles for running experiments. Pending is another status that indicates consent form is waiting to get approved by administrator role. Fig-

Figure 6.18 shows the consent management GUI for new consents that are in pending state. Administrator is able to approve or reject them through the action buttons column

Study and Consent Management

Study Management Consent Info **New Consents** Expired Studies









Study Owner	StudyName	Consent Type	Date	Document	Action
gholami@kth.se	BRCA	Consent Info	Mon Apr 13 00:00:00 CEST 2015	brca_consent.pdf	 
gholami@kth.se	BRCA	Consent Info	Fri Apr 24 00:00:00 CEST 2015	new_brca_consent.pdf	 
gholami@kth.se	BRCA	Ethical Approval	Fri Apr 24 00:00:00 CEST 2015	ethical_approval.pdf	 
gholami@kth.se	BRCA	NonConsent Info	Fri Apr 24 00:00:00 CEST 2015	brca_nonconsent.pdf	 

Figure 6.18: Reviewing the new consents to be approved or rejected

When retention period of a study is expired, the administrator is notified through the “Expired Studies” tab as shown in Figure 6.19. The administrator selects the expired study and presses the delete button (the red button in action section). This will remove all the data related to that specific study. However, for auditing purposes, the platform keeps track of the study meta-data which cannot be used for re-identifying of individuals.

Study and Consent Management

Study Management Consent Info New Consents **Expired Studies**


Study Owner	StudyName	Study Status	Retention Period	Action
gholami@kth.se	BRCA1	APPROVED	Wed Apr 24 00:00:00 CEST 2013	

Figure 6.19: Expired data sets to be removed by the administrator

6.4.6 Auditing

The auditing system is implemented using the JSF and EJB technologies of the Java Enterprise Edition. We mapped the auditing related relations such as *account_audit*, *user_logins*, *activity*, *projects*, *consents_audit* and *roles_audit*. For example, in Listing A.2, we show how *account_audit* relation has been mapped to the roles related entity in the model layer. In addition, we developed several

managed beans view scoped and stateless EJB to store or retrieve the security related events. Listing 6.6 shows the implantation of creating audit events for role entitlement by the administrator.

Listing 6.6: Example of creating role entitlement events in the database

```
public boolean registerRoleChange(Users user, String action, String
    outcome, String message, Users target, HttpServletRequest request) {

    // create an role audit event
    RolesAudit rolesAudit = new RolesAudit();

    // initialize the object with the event info
    rolesAudit.setInitiator(user);
    rolesAudit.setBrowser(AuditUtil.getBrowserInfo(request));
    rolesAudit.setIp(AuditUtil.getIPAddress(request));
    rolesAudit.setOs(AuditUtil.getOSInfo(request));
    rolesAudit.setEmail(u.getEmail());
    rolesAudit.setAction(action);
    rolesAudit.setOutcome(outcome);
    rolesAudit.setTime(new Timestamp(new Date().getTime()));
    rolesAudit.setMac(AuditUtil.getMacAddress(AuditUtil.getIPAddress(request)));
    rolesAudit.setMessage(message);
    rolesAudit.setTarget(target);

    // persist the info in the data layer
    entityManager.persist(rolesAudit);

    return true;
}
```

The audit panel is accessible for both administrator and auditor roles, as shown in Figure 6.20. Through this panel, different auditing contexts can be selected to generate custom auditing reports. The audit contexts consist of *Roles Audit*, *Login Audit*, *Account Audit* and *Project Audit*.

Role Audit Context provides audit trails related to changing the roles of users by administrator role (see Figure 6.21). For example, when a new account is approved and is assigned a role by the administrator. Through this GUI, an administrator or an auditor is able to generate audit reports for a specific username in a period of time for different options such as login, logout, and registration.

Login Audit Context provides information related to user login, log out attempts or registration. Figure 6.22 shows an audit report for all users' logins events during a specific period of time. The report can be filtered using different parameters such as username, browser, OS, IP/MAC address, or outcome of the event.

Account Audit Context helps top generate audit trails for user accounts modifications, such as password change, status change, and security question change

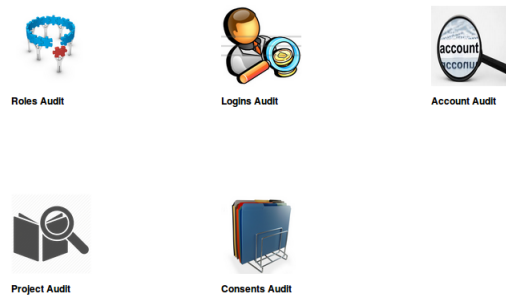


Figure 6.20: Audit panel accessible for administrator and auditor roles

Roles Audit

username 5/1/15 5/31/15

Initiator	Target	IP Address	Action	Action Detail	Date	Outcome
admin	admin	127.0.0.1	ADDED ROLE	BBC_RESEARCHER	Thu May 28 2015	SUCCESS
admin	admin	127.0.0.1	ADDED ROLE	AUDITOR	Thu May 28 00:00:00 CEST 2015	SUCCESS

Figure 6.21: Role access and entitlement events audit panel

(see Figure 6.23).

Project Audit Context provides fine-grained audit reports for studies belonging to different projects in the platform. Figure 6.24 shows the audit report for the registered studies. The Auditor specifies the study name which is an optional parameter, in addition to start and end dates of an action.

6.5 Verification and Validation

The verification and validation of our proposed solution included extensive testing, dynamic analysis on the executing codes and fuzzing to highlight coding errors and the potential security bugs. Additionally, we involved the end-users in assessing the actual platform and expressing their expectations. In summary, following steps were carried out.

- **Fuzz Testing:** We examined different scenarios where unexpected data were given using WebScarab [212] which is a popular open source web security

Logins Audit

username 5/1/15 5/31/15

Username	Browser	OS	Date	IP Address	Mac Addr	Action	Result
admin	Firefox	Unix	Fri May 29 00:59:01 CEST 2015	127.0.0.1	00-50-B6-58-BA-15	LOGIN	SUCCESS
admin	Firefox	Unix	Fri May 29 00:59:25 CEST 2015	127.0.0.1	00-50-B6-58-BA-15	LOGOUT	SUCCESS
admin	Firefox	Unix	Fri May 29 01:03:16 CEST 2015	127.0.0.1	00-50-B6-58-BA-15	REGISTRATION	SUCCESS
admin	Firefox	Unix	Fri May 29 01:03:27 CEST 2015	127.0.0.1	00-50-B6-58-BA-15		SUCCESS
admin	Firefox	Unix	Fri May 29 01:03:36 CEST 2015	127.0.0.1	00-50-B6-58-BA-15		SUCCESS
admin	Firefox	Unix	Fri May 29 01:04:50 CEST 2015	127.0.0.1	00-50-B6-58-BA-15		SUCCESS
admin	Firefox	Unix	Fri May 29 01:07:26 CEST 2015	127.0.0.1	00-50-B6-58-BA-15		SUCCESS
admin	Firefox	Unix	Fri May 29 01:07:34 CEST 2015	127.0.0.1	00-50-B6-58-BA-15		SUCCESS
admin	Firefox	Unix	Fri May 29 01:11:09 CEST 2015	127.0.0.1	00-50-B6-58-BA-15		SUCCESS
admin	Firefox	Unix	Fri May 29 01:11:30 CEST 2015	127.0.0.1	00-50-B6-58-BA-15		SUCCESS
admin	Firefox	Unix	Fri May 29 01:12:04 CEST 2015	127.0.0.1	00-50-B6-58-BA-15		SUCCESS
admin	Firefox	Unix	Fri May 29 01:13:51 CEST 2015	127.0.0.1	00-50-B6-58-BA-15		SUCCESS
admin	Firefox	Unix	Fri May 29 01:26:42 CEST 2015	127.0.0.1	00-50-B6-58-BA-15		SUCCESS
admin	Firefox	Unix	Fri May 29 08:35:15 CEST 2015	127.0.0.1	8C-70-5A-83-E0-2C		SUCCESS

Figure 6.22: Auditing the login events of users

Account Audit

username 5/1/15 5/31/15

Username	Date	Action	Action Detail	IP Address	Mac Addr	Result
admin	Fri May 29 00:00:00 CEST 2015	PROFILE UPDATE	UPDATE INFO	127.0.0.1		SUCCESS
admin	Fri May 29 00:00:00 CEST 2015	PROFILE UPDATE	UPDATE ORGANIZATION	127.0.0.1		SUCCESS
admin	Fri May 29 00:00:00 CEST 2015	PROFILE UPDATE	UPDATE ADDRESS	127.0.0.1	00-50-B6-58-BA-15	SUCCESS
admin	Fri May 29 00:00:00 CEST 2015	PROFILE UPDATE	UPDATE INFO	127.0.0.1	00-50-B6-58-BA-15	SUCCESS

Figure 6.23: Auditing of account management activities

application testing tool. Using WebScarab we managed to intercept and alter both HTTP and HTTPS requests header fields and web server responses for different functionalities in the IAM, authentication, auditing and privacy control. For example, testing the session identifier strength (cookie values over time), altering the encodings such as Base32, initiating SQL injection or XSS attacks that are included in the full-featured interface of WebScarab.

Adversarial attacks such as cookie-theft, session hijacking or SQL injection or

Study Audit

Study Name: 5/1/15 5/31/15 AUDIT TRAILS

Name	Study Name	Email	Action Date	Action
All Gholami	IranianDiabetesStudy	gholami@pdc.kth.se	Mon May 04 13:38:17 CEST 2015	created new study
All Gholami	IranianDiabetesStudy	gholami@pdc.kth.se	Mon May 04 13:38:40 CEST 2015	added new member lora.dimitrova@charite.de
Karin Zimmermann	Teststudy	zimmer@informatik.hu-berlin.de	Mon May 04 16:31:55 CEST 2015	created new study
All Gholami	BRCA1	gholami@kth.se	Thu May 07 11:22:42 CEST 2015	created new study
All Gholami	BRCA2	gholami@kth.se	Thu May 07 12:34:39 CEST 2015	created new study
All Gholami	BRCA2	gholami@kth.se	Mon May 25 15:00:00 CEST 2015	updated retention period to: Tue Jun 30 00:00:00 CEST 2016
All Gholami	BRCA2	gholami@kth.se	Mon May 25 15:00:06 CEST 2015	uploaded consent info: brca-consent.pdf
All Gholami	BRCA2	gholami@kth.se	Mon May 25 15:04:23 CEST 2015	uploaded ethical approval info: brca-approval.pdf
All Gholami	BRCA2	gholami@kth.se	Mon May 25 15:11:33 CEST 2015	updated retention period to: Tue Jun 30 00:00:00 CEST 2015
All Gholami	BRCA2	gholami@kth.se	Mon May 25 15:11:55 CEST 2015	uploaded non-consent info: brca-nonconsent.pdf

Figure 6.24: Auditing project information including based on several parameters such as the study name, date of access and username

XSS were denied due to the strong support from Java Enterprise Edition that enabled us to encrypt the sessions and cookies or enforcing security through the EJB API that makes SQL injection almost impossible. However, we didn't perform any DNS cache poisoning or network-level DoS attacks because our focus is to ensure security on layers above infrastructure (PaaS and SaaS).

- **Authentication:** To ensure that authentication requirements are always enforced we performed several test scenarios to bypass the two-factor authentication system through direct page requests, reusing expired sessions after 30 minutes or brute-force attacks which were not succeeded. Also, we tested the “remember me passwords” for being stored in a cookie. We ensured that such credentials or QRCs are not stored in the browser's cache.

We also examined the strengths of our pre-generated security questions and elaborated not to choose fairly simplistic questions that can lead to insecure answers. For example “What is your favorite singer?” or “What is your favorite car?”. For this purpose, we started guessing answers to such questions through brute-forcible answers. However, the security framework is able to block the accounts after a fair amount of wrong answers (in our case we decided 5 incorrect answers).

Resetting passwords or account recovery options were also tested at this stage. The security framework sends a verification code to the user's email to be entered for account recovery. The framework allows only a specific number of attempts and after it blocks the account. To avoid DoS attacks we allow users with valid credentials to activate their blocked accounts due to false

attempts.

- **Authorization:** Verification of the authorization system included defining single-role and multi-role test cases, where users with lower predigest access attempted to perform tasks with higher role permissions. For example, a simple researcher to perform administrative tasks or access functionalities of a data provider that were prohibited by our policy. Our authorization system in the server-side was able to deny such attempts and enforce the required permissions.
- **Information Disclosure:** We published our source code under the Apache version 2 License in Github and consider several security issues when deploying the framework in real infrastructures. We did an analysis to ensure the published source code, log files or documentation will not pose any risk on the security credentials or sensitive information. All communications in the public networks are encrypted over TLS/SSL channels. Internals of the BioBankCloud are protected inside an isolated network behind the firewall to minimize the eavesdropping through the man-in-the-middle attacks.

To validate our implementation we deployed the security framework in a private cloud that was provisioned using Chef/Karamel. We then tested all the functionalities we discussed in this chapter and could improve any possible defects.

The security framework was presented in several workshops and training events for bioinformaticians, biobankers, data scientists and administrator. All users were able to successfully create accounts and run their experiments on the platform. Despite the rigorous development of smartphones in our daily lives surprisingly there were approximately 40% of workshop participants preferred to use Yubikey for authentication due to more simplicity.

Feedbacks from workshop participants and also other users motivated us to improve the GUI for a more efficient and trendy framework using AngularJS. As we discussed throughout this chapter we have started re-implementing several functionalities in AngularJS that have direct interaction with users, which is an ongoing research²⁵.

6.6 Discussion

This section describes how our implementation of the key privacy requirements of the BioBankCloud from the previous chapter complies with the DPD. We provided technical and non-technical measures, in addition to organizational safeguard for a privacy-preserving cloud platform that handles genomic data within the EU's jurisdictions. For each category of the privacy requirements, we provided the following solutions.

²⁵<https://github.com/hopshadoop/hopsworks>

- **Lawfulness:** To ensure lawfully and legal processing of genomic data in the BioBankCloud, we require evidence including informed consent, ethical approval and other evidence that provide ground for the lawfulness of activities. Such proofs and documents can be uploaded in the privacy settings menu per-study or project as described in Section 6.4.5.
- **Informed Consent:** During the registration process, users are required to read and accept the ToS of the BioBankCloud that clearly states the platform responsibilities and user rights individually. This ToS declares what happens to genomic data and what responsibilities are on each party. This feature is implemented on the account registration page as discussed in Sections 6.4.2. Further, our solution includes renewing different categories of consents that are defined in Section 6.4.5, granularly for each study or project.
- **Purpose Binding:** The BioBankCloud administrator monitors the activities of the researchers and will ensure that experiments are according to the consent and ethical approval information. In the BioBankCloud, administrator or data provider are able to monitor the researcher activities through the project audit context (Section 6.4.6) to ensure data is only used according to the original purposes defined in the provided contracts.
- **Data Minimization:** The BioBankCloud platform stores genomic data for the defined period of time demanded by the data provider. The platform will erase all expired data after pre-defined retention periods, that are set by data providers (see Figure 6.19). This feature also ensures the principle of being forgotten in the event of a removal request by the data provider.
- **Data Accuracy:** Users and data providers are able to modify and update their data and erase what is believed to be inaccurate information. For example, through removing a study, uploading new consents, or modifying personal information (Section 6.4.5).
- **Transparency:** The BioBankCloud platform maintains a transparent processing of the genomic data, where users are clarified about the physical handling of the data till fine-grained access by other authorized researchers. Users are able to get evidence on their data usage as discussed in Section 6.4.6. Any changes in the data processing procedures or platform and regulations will be immediately propagated to the users. In addition, we will notify the users upon any privacy breach or unauthorized access to their data.
- **Data Security:** We implemented several strong security measures such as two-factor authentication and using encrypted HTTPS channels to ensure confidentiality. We used encrypted sessions in JSF, secure cookies in the AngularJS frontend, input validation to prevent SQL injection attacks prior to forwarding the user requests to the backend. In addition, we deployed an RBAC control so that different roles with different permissions are able to

perform their tasks without any conflict to ensure the integrity of data. The security implementation details are described in Section 6.4.

- **Accountability:** We proposed organizational safeguards through Service Organization Controls (SOC1) reports [206] in addition to providing internal auditing for administrator and auditor roles. Administrator and auditor roles are able to generate audit reports for different contexts for data access and usage of data and access management (Section 6.4.6).

6.7 Summary

This chapter discussed the implementation of the BioBankCloud as a platform that supports scalable processing of genomics analysis. The CPTM is applied to identify the privacy requirements of the DPD to be deployed in the BioBankCloud security framework.

The security framework includes a role model for sharing genomics data among different participants. The proposed framework lays out the privacy requirements of the DPD through implementing various technical countermeasures and organizational safeguards to prevent or mitigate effects of the identified threats.

We believe this work empowers the BioBankCloud to be run by a trusted cloud provider within the EU's jurisdiction for processing and storing sensitive genomics data.

Part III

Trustworthy Privacy-Preserving Cloud Models

Chapter 7

Privacy-Preserving Data Publishing for Sample Availability Data

This chapter is mainly based on publication VIII and it presents an architecture that provides privacy-preservation for sample availability data across multiple data providers.

7.1 Introduction

Over the last number of years, valuable data has been accumulated in many healthcare-related databases throughout the developed world. By definition, these repositories contain sensitive medical data, which are fragmented depending on the type of clinical and research activities. Although it would be technically possible to merge the different data silos into a central database that could be queried by medical experts and researchers alike to allow for new insights previously thought impossible, the security risks of doing so are unacceptably high. Therefore, there is need to somehow combine the data from different databases, i.e., Bob's study DB and Alice's study DB in a way that minimizes the risk of exposing sensitive personal data.

In its simplest form, a use-case for the combined databases would be the following. Two databases are given: Bob's DB and Alice's DB. The Bob's DB contains entries of people who have some samples deposited related to their illness and are potentially eligible for research purposes. Alice DB is also a list of persons, along with their record of hospitalization and treatments they were subjected to. Both databases contain personal information, such as name, birth date, and the Personal Identifier, which uniquely identifies them in the database. The proposed identifier can be used to link the different records in the different databases, but there is the obvious need to provide anonymity for the patients. While the combination of the information contained in different databases is extremely useful for research

purposes, the actual information used to identify each individual is not essential for the studies to be performed by the investigators themselves.

Existing sample availability systems, such as [165] provide individual level information on the availability of specific data types within a collection, not across foreign collections. That is, researchers are not able to cross-link (similar to an equality join in SQL) data from different outside studies, as the identity of the samples is completely anonymized. However, researchers would like to discover correlations between individuals in different studies, and this is not possible in the existing systems.

In this chapter, we present a privacy-preserving system for publishing availability data about samples from patients to address the limitations of existing solutions, which allows researchers to cross-link sample availability data from different medical study databases while preserving the privacy of the patients. To this end, we build an anonymization toolkit to anonymize and measure the re-identification risk of the sensitive data to be published, while cross-linking queries can be executed by the researcher.

The structure of this chapter is as follows. Section 7.2 describes the privacy-preserving data publishing approaches. In section 7.3, we introduce the pseudonymization data model for personal identifiers. In section 7.4, we define the threat model including main threats to the privacy of the sensitive data. Section 7.5 presents an architectural overview of our solution and implementation details. Finally, section 7.6 summarizes our findings.

7.2 Privacy-Preservation Mechanisms

Dalenius [213] defines privacy-preservation as:

“access to the published data should not enable the adversary to learn anything extra about any targeted victim compared to no access to the database, even with the presence of any adversary’s background knowledge obtained from other sources.”

To anonymize the data, several statistical disclosure control techniques can be applied to achieve privacy-preservation through anonymity, such as k -anonymity and ℓ -diversity. This anonymity model ensures privacy-preservation in record and attribute linkage.

- **Record Linkage:** Sweeney [68] proposed k -anonymity as a model for privacy preserving of QIDs. The k value is the minimum number of the records in a table that have similar QIDs. This notion of k records in a group reduces the risk of re-identification of a participant to the probability of $1/k$. However, k -anonymity is weak regarding the background knowledge of the adversary about a victim [72].
- **Attribute Linkage:** To overcome the limitations of k -anonymity, we use ℓ -diversity [72] as an extra privacy measure to protect the anonymity of the

individuals from re-identification through the adversary’s background knowledge. The value of ℓ defines, at least, ℓ “well-represented” sensitive values in the table to reduce the confidence of inferring a sensitive attribute within a group. Entropy (E) of the entire table must hold $E \geq \log(\ell)$ to ensure every distinct QID block, at least has ℓ distinct values for the sensitive attribute.

7.3 Obscuring the Key Attributes

Data anonymization methods remove PII of patients, helping to reduce the risk of patient re-identification when patient data is used for research purposes. Researchers often want to cross-link different sample databases, for example, to discover correlations between different studies. For cross-linking, a PII could be used to link the individuals to their original records, however, using the original PII allows for patient re-identification. Pseudonymization is an alternative approach, where the PII are not stored in their original format, but cross-linking sample databases are still possible. There are two well-known pseudonymization schemes: to build a database to store mappings of the PII-to-pseudonyms or using cryptographic mechanisms applied to the pseudonyms [214].

Our proposed pseudonymity model to extract and obscure the key attributes using a two-level mechanism that maintains the possibility of joint queries over anonymous records in different collections.

We assume that our data model as a sample table (T) of a population, fragmented as PIDs, QIDs, sensitive and non-sensitive attributes: $T(\text{PID}, \text{QID}, \text{Sensitive}, \text{Non-Sensitive})$. A QID can be considered as a combination of attributes that can be linked with external information to re-identify an individual i.e., ZIP code, date of birth and gender [4].

7.3.1 Hashing and Encryption

To de-identify the records, we use the Secure Hash Algorithm (SHA-3) [215] to convert the PIDs to irreversible pseudonyms. To add another level of security, the de-identified records will be encrypted using the AES [216] with an embedded key in a slot of a Yubikey device that is distributed off-line to the data providers.

As an example, assume in Sweden, a personal number (Swedish civic registration number) is a 10-digit PIN issued by the National Tax Board for all residents in the country. The personal number or PID is structured in three parts: date of birth, a three-digit birth number, and a check digit. The date of birth construction contains two digits each for the year, month, and date of birth, i.e., 610514. This is followed by a three-digit birth number (i.e., 323) and a check digit (i.e., 4), as shown in Figure 7.1. The birth number value will be a number between 001 and 999, where the last digit is also used to indicate the gender, with men were given an odd number and women an even number.

Figure 7.1 illustrates the two-level mechanism, where SHA-3 de-identifies a Swedish PID and AES encrypts the pseudonymized (PPID).

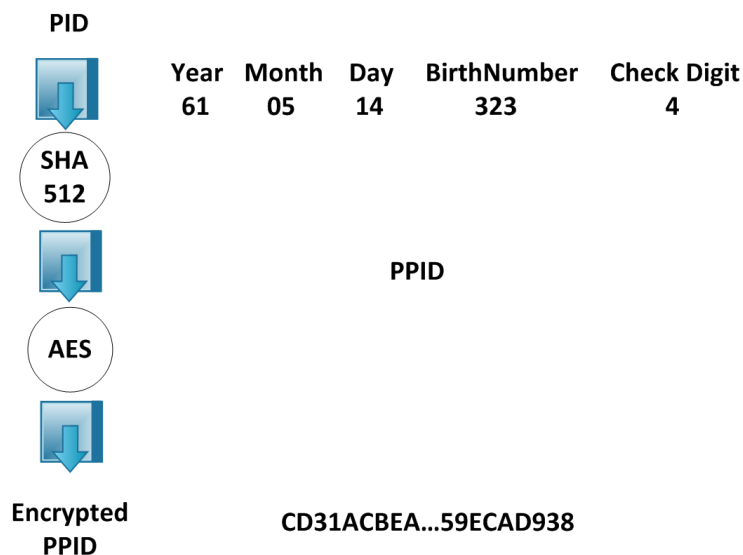


Figure 7.1: PID pseudonymization through a two-level hashing mechanism to provide the functionality for joint queries over different data sources.

7.4 Threat Assumptions

To ensure privacy protection through the proposed data model discussed in section 7.3, we define a threat model to declare the possible attacks and security breaches that cause loss of confidentiality and integrity of the data. The main threats to our data model, are mainly due to cross-linking of the data sets with the de-identified PIDs that are not fully anonymized. Hence, we use an integration server, that can be considered as a safe third-party server behind a firewall that will be updated regularly with the patches and libraries, with restricted access to only administrative staff to ensure sufficient security and reliability.

7.4.1 Inference Attacks

If a malicious adversary has access to both unencrypted anonymized data stored by the TTP and the key to pseudonymize the PIDs, then the adversary will be able to make inference attacks through linking the victim’s generated hashed personal identifier to the data acquired from the integration server i.e., through dictionary attacks. To mitigate the likelihood of such threats, all published data will be stored encrypted with the TTP’s private key.

When a researcher issues join queries over different databases, inference attacks become possible. For instance, issuing a query containing a small number of participants to find out, whether a specific person is available in any of the samples

that are published on the integration server. As a countermeasure, the integration server will not accept queries less than N number of participants.

7.4.2 Malicious Sample Publication

We assume that sample data providers are trusted bodies and that biobanks or other parties publish correct data sets to the integration server. But, a malicious or incorrect data provider could publish incorrect data sets either intentionally or by mistake. To reduce the effects inaccurate disclosed data, our system will keep track of the registered data and provide a flexible approach to remove the stored data.

7.4.3 Audit and Control

The TTP stores the audit trails and log files securely, to enable data providers to audit access to their data by the users. To ensure the integrity of the audit trails, our system encrypts the audit logs and information using AES symmetric keys stored in the integration server key store.

7.4.4 Server Private Key Compromised

If server's private key will be compromised or stolen by an adversary, then the server's public key and associated private key should be revoked. In such a case, data providers should be notified about the incident and the anonymized datasets in the server's database should be re-encrypted with the server's new private key.

7.4.5 Ethical Constraints

The usage of the system to issue and get results of the joint queries should be considered as a potential threat to the purpose of the collected data samples in the integration server. For instance, if a researcher uses the information for other studies that are out of the scope of the agreed framework.

7.4.6 Static Passwords

The Yubikey static passwords are vulnerable against a keylogger that records keystrokes by a user. The information collected by a key logger usually saved as a file or sent directly to third parties. Because of the Yubikey function, as a USB keyboard, it will be possible for a keylogger to intercept the text stream when in static mode. Therefore, users should be aware of underlying platforms and as a good recommendation use their local PCs to reduce the risk of password thefts.

7.4.7 Query Reply Limitation

The final results of a query will be a set of records, combined within a table that is generated by the TTP. Though the combined table contains only the anonymized data but uniqueness of the entries in the intersection of different tables might raise the re-identification risk of the records through inference attacks. As a counter-measure, the TTP will check the anonymity of the combined table and will apply another level of minimization to ensure at least k -anonymity with the value of $k = 3$.

7.5 Design and Implementation

7.5.1 Scenario

Assume eCPC is a medical research center with the aim to develop a modular system for prediction of cancer initiation and progression using modeling and simulation ¹. An important part of this ambition is to integrate data from different sources, such as biobanks containing data about samples, and clinical health registries (quality registries) containing information about patients and their diseases, treatments, and outcomes. This integrated data can then be used in subsequent modeling and simulation efforts. A key concern in medical data integration is the acceptance and participation of data providers.

We present a first step in the data integration on sample availability data, which lowers the barriers for data providers to participate. To this end, we developed a toolkit, illustrated in 7.2, that pseudonymizes sample availability data and then securely publishes the pseudonymized data to an integration server that can be queried by researchers (including support for cross-linking queries).

In order to ensure the privacy of sensitive patient data, the eCPC toolkit applies the guidelines for safe microdata, outlined as follows.

- The eCPC toolkit removes all explicit identifiers, and it will extract and de-identify all the PIDs, as described in Section 7.3.
- Then it categorizes the remaining attributes to determine the key variables according to both legal requirements and domain specific judgments, which may be subjective. Key variables might also be split into further categories according to the level that they are identifying.

This distinction is useful when prioritizing which variables need to be modified to enhance safety: more identifying keys are modified first for observations that have a considerable high risk. This graded approach allows for better data quality preservation and, therefore, higher data utility.

¹<http://www.e-science.se/community/eCPC>

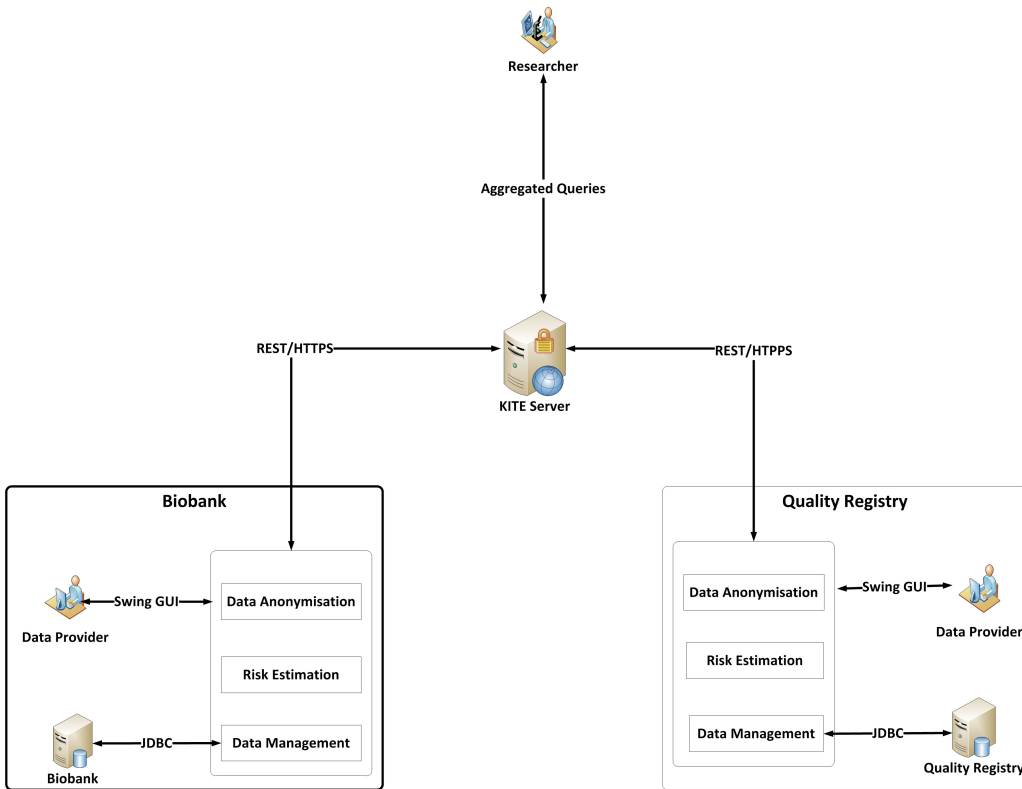


Figure 7.2: The eCPC toolkit design based on the privacy-preserving data publishing methods to upload the pseudonymized data to an external trusted third-party service.

- When key variables have been identified, the re-identification risk needs to be assessed. This is done by looking at the uniqueness of the observed entries through frequency counting and calculating probability estimates based on extrapolating models taking population frequencies into account.
- Entries that stand out from the rest and, therefore, have a considerable risk to be subject to re-identification are then modified. Numerous modification algorithms exist, namely generalization or global recoding and local suppression of outstanding values, swapping, rank swapping or perturbing with post randomization – not to be confused with randomized questionnaires when collecting data, hence the name post randomization. The methods to be applied depend on the nature of the variables, whether they are categorical or continuous, their structure such as significance order, hierarchy, geography, semantics and the size of the dataset in question. Each algorithm applied is

recorded in a logbook for the analyst’s documentation, i.e., the nature of the added noise, if any.

- The re-identification risk has to be measured again and the information loss has to be evaluated. If the risk is deemed acceptable and the quality of the data remains adequate, then the resulting microdata can be considered as safe. If not, the previous step needs to be repeated.
- Finally, the data provider i.e., biobank or quality registry, publishes the pseudonymized data sets to the integration server.

7.5.2 Integration Service

The eCPC integration service provides a SaaS model in a private trusted cloud infrastructure using the Java Enterprise Edition, as shown in Figure 7.3. Researchers can visit the main page of the eCPC service and then they will be asked to authenticate to the system through TLS/SSL encrypted channels to protect their credentials from eavesdropping attacks. Users are authenticated using container-supported application-level authentication provided by the web application server.

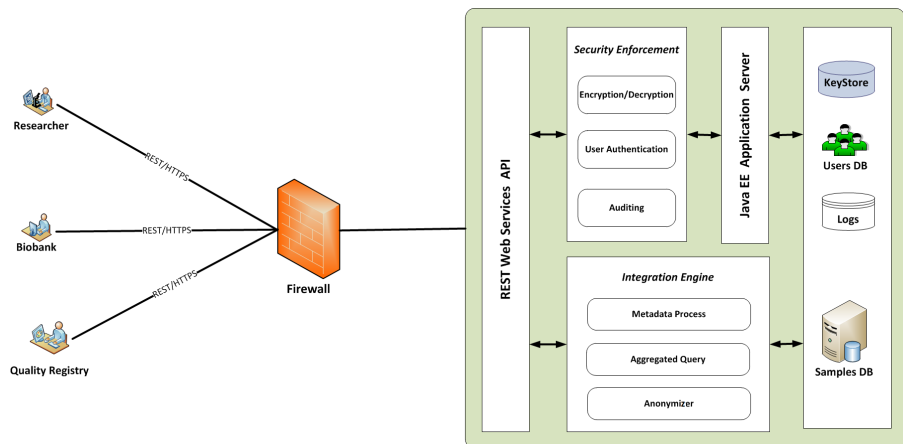


Figure 7.3: Overview of the eCPC integration server that is protected with firewall to filter the ingoing/outgoing traffic.

As Figure 7.3 demonstrates, the integration server consists of three main components: security enforcement, integration engine, and REST Web services API. The security enforcement component handles the security related tasks using encryption/decryption of the data sets, log files, user’s authentication and auditing processes through the Java Enterprise Edition application server. The integration engine deals with query processing and joins over different data sets. The metadata service, as a part of the integration engine, provides all available attributes

from each data source that can be selected by the researcher when issuing a query. The REST Web services API receives incoming requests from the eCPC clients behind the firewall. We set up a firewall² in a Linux server VM to filter the traffic between the internal and external zones through the HTTPS connections in the eCPC integration server.

In a typical usage scenario, a researcher authenticates via a web page with the web application server and after successful authentication, she will be redirected to a web page where she can issue queries. This is enforced through a RBAC model with the following roles.

- **Administrator:** Provisions or provisions users in the system and manages access to multiple datasets
- **Data Provider:** Uploads or deletes datasets to the integration server through secure channels
- **Researcher:** Issues cross-linking queries to the authorized data sources

We preliminary designed a solution to store the data sets in a MongoDB but for re-usability and interoperability purposes with the Java Enterprise Edition middleware, we decided to use the open-source MySQL database as backend. This design choice enabled us to re-use as much as possible from the BioBankCloud project for authentication, authorization or auditing in the service. This way, researchers can browse available data sets and select attributes of datasets for cross-linked queries (see Figure 7.3). For example, a researcher might search for cross-linked samples in the prostate and diabetes quality registries by issuing a query like: “how many samples are available for patients who have had both prostate cancer and diabetes and have a BMI greater than 30”. The join will run over different data sets. As our queries are executed at application-level, we can still join across encrypted data sets by decrypting the contents of each data collection on the fly using the server’s private key, stored in a secure key store.

The integration server also stores the logging events in a separate database by encrypting them using AES 256-bit symmetric keys, where keys are stored securely in the key store. When data providers wish to know about the access to their data in a specific period of time, the auditing component retrieves the logs associated with the data published by the owner and sends back the results to the data provider through a REST API component.

7.5.3 Secure Data Management

In order to deploy a secure solution to store the pseudonymized published information on the integration server, we used public key certificates for encryption/decryption of the data sets. Although our data providers typically store their data

²<https://help.ubuntu.com/lts/server/guide/firewall.html>

sets in relational databases, the analysis of that data is typically carried out on Comma Separated Values (CSV) files. As such, the main data format used for publishing data is a CSV file. The integration server provides an X.509 public key certificate to the client for data encryption, prior to data publishing. Figure 7.4 illustrates the data encryption/decryption process, where a data provider encrypts its CSV files with the server’s X.509 public key, and the integration server decrypts the uploaded data on-demand using its private key.

A limitation imposed on us by public key certificates is that the length of individual fields cannot exceed a size determined by the server’s public key size, i.e., 512 bytes for a 4098-bit public key. However, this is not a problem for our toolkits as all fields are significantly smaller than 512 bytes.

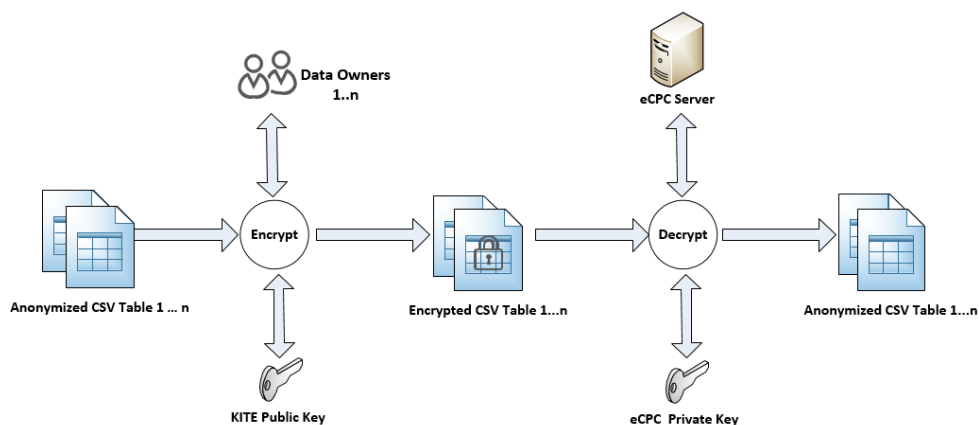


Figure 7.4: Public key encryption of the large sensitive data sets using the TTP’s private key.

The data management component (Figure 7.2) uses JDBC to export sensitive data sets from a relational data source to CSV files for pseudonymization. We implemented JDBC because the majority of data providers use clinical data in structured formats. However, the toolkit allows the data provider to upload CSV files if not provided by a relation storage. The data sets are then securely published to the integration server via a REST API running over HTTP on top of the TLS/SSL protocol, preventing eavesdropping attacks.

7.5.4 Data Pseudonymization and Anonymization

The eCPC toolkit leverages microdata protection through anonymization algorithms implemented by k -anonymity and ℓ -diversity. Although the microdata as

a whole is pseudonymized, we leverage existing data anonymization algorithms to anonymize sensitive attributes in the microdata. We now describe the anonymization phase in our pseudonymization process. A data provider selects a CSV data set that is generated from a relational database and defines the key and sensitive attributes to be anonymized. For this purpose, the CSV toolkit visualizes the metadata of a specific CSV file, as shown in Figure 7.5.4, to enable a data provider to tag attributes as sensitive, key or non-sensitive. We implemented our solution based on `sdcMicro` [171] that provides R-based API for both data anonymization and risk estimation. As our toolkit is implemented in Java, we ran `sdcMicro` in batch-mode. We did not find any existing anonymization toolkits, i.e., `μ-Argus` [170] or UTD Anonymization Toolkit [172] that supports both the calculation of re-identification risk and our anonymization algorithms, k -anonymity and ℓ -diversity, in a platform-independent approach. Moreover, the R Java environments such as `Rcaller`³ and `Renjin`⁴ were not stable enough to run our `sdcMicro` tasks.



Figure 7.5: Anonymization of the sensitive data using `sdcMicro` library.

When a user presses the “Anonymize” button, two things happen: the PID is pseudonymized and the key attributes are anonymized. Afterward, the user ensures that ℓ -diversity is satisfied by setting the ℓ -diversity value and pressing the relevant button.

The pseudonymization process converts PIDs using converted by the SHA-3 function, as described in Section 7.3.1.

³RCaller, A library for calling R from Java, <http://code.google.com/p/rcaller/>

⁴Renjin, a JVM-based Interpreter for the R Language for Statistical Computing, <http://code.google.com/p/renjin/>

The pseudonymization process reads the embedded key in the Yubikey device to be used by the AES encryption function prior to data publishing.

The anonymization phase provides the user with visual feedback in the form of a chart containing the number of suppressions for each sensitive attribute. An example of such a chart given in Figure 7.5.

Anonymization will result in attribute values being suppressed when either k -anonymity or ℓ -diversity constraints are not met. Our toolkit enables data providers to iteratively change the values of k and ℓ to minimize the number of suppressions for a desired re-identification risk level. The actual implementation of k -anonymity and ℓ -diversity are presented in Appendix B: Listing B.1 and Listing B.2.

7.5.5 Re-identification Risk

The purpose of the re-identification risk estimation process is to enable the data provider to measure the re-identification risk of anonymized sensitive attributes as a result of the data anonymization process (see Section 7.5.4). The eCPC toolkit allows the data provider to select the sensitive attributes for risk calculation (see Listing B.3 for the actual implementation). The risk measurement diagram of Figure 7.6 demonstrates different levels of risk for different individual records. In this example, 43 out of 100 records will be re-identified with the risk of $r \leq 0.1$.

If the risk levels are deemed to be unsafe, the data provider will repeat the anonymization process with different values of k and ℓ until the risk is considered to be acceptable.

7.5.6 Auditing Process

Our integration server supports researchers issuing queries for data availability from different data sources. In order to reduce the threat of malicious queries, we securely audit queries, identifying the queries that have been issued, by whom, and when. The eCPC toolkit client (a Java GUI) uses a REST API to allow data providers to download an audit trail for the queries that accessed their data source. For each query, the audit trail includes the name and institution of the requester, date of access, the purpose of the study, the IP address of the host that issued the query, and the actual query. Data providers can use this information to infer whether there has been a breach of privacy, and who was responsible for that breach. The audit trails can also be used to determine when a researcher is executing a brute force attack on the data sets.

7.6 Summary

This chapter introduced a privacy-preservation publishing toolkit that supports the secure publishing of pseudonymized data sets to an integration server by data providers and audited querying of the data sources by researchers. Our toolkit includes a secure de-identification mechanism for publishing pseudonymised patient

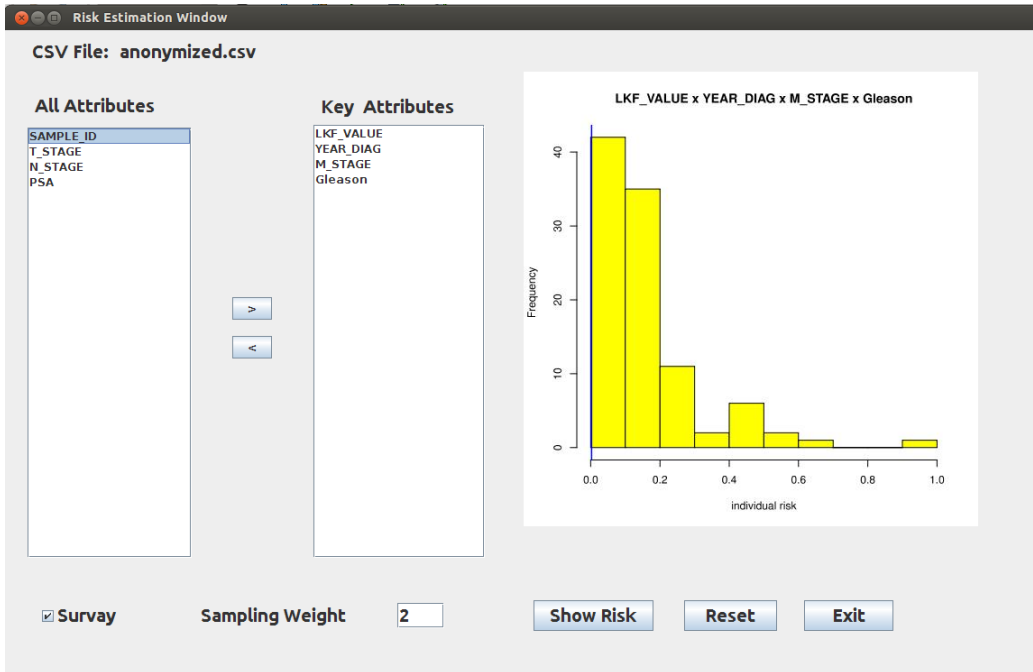


Figure 7.6: Individual risk estimation of the pseudonymized data using `sdcMicro` library.

identifiers through a two-level hashing mechanism, as well as tools to anonymize sensitive clinical data using k -anonymity and l -diversity algorithms. Our toolkit also estimates the re-identification risk for individual records, providing data providers with feedback for configuring the k -anonymity and l -diversity parameters. Data providers can encrypt their large data sets using the public key certificate of the integration server for additional security. We also securely audit queries, helping to reduce the risk of misbehavior by researchers, and enabling subsequent identification of rogue users.

Furthermore, we proposed a solution that allows researchers to access multiple data sources to run cross-linking queries to get insight into diseases. The proposed architecture contains several modules for secure access to storage of encrypted datasets through RBAC. There is also an auditing module designed to provide evidence of data access and platform usage to the interested participants such as data providers or other external auditing entities.

The eCPC toolkit is designed to support anonymization of clinical structured datasets. There are scenarios that involve usage of sensitive medical data with other data types such as brain imaging. Next chapter outlines a privacy-preserving for brain imaging data.

Chapter 8

Privacy-Preserving Brain Image Analysis in the Cloud

This chapter is mainly based on publication IX and it presents a privacy-preserving SaaS cloud that provides scalable analysis of brain images datasets.

8.1 Introduction

Functional neuroimaging has been used to study a wide array of psychological traits, including aspects of personality and intelligence. Cloud computing as an enabling technology helps to eliminate the barriers such as restricted resource scalability imposed on researchers by the existing systems in the brain imaging area. Among the most prominent of these concerns is the potential use of functional neuroimaging to obtain personal information about individuals, i.e., correlation of personality, attitudes, and intelligence.

Similar to the genetics or clinical data, brain images contain sensitive information about the individuals that is demanded by privacy laws to be protected. Different from such data type, brain images require different tools due to different semantics and data formats. In this chapter, we apply anonymization and other security measures such as authentication and authorization on the functional analysis of 3D brain imaging data acquired mainly from magnetic resonance scanners so-called Functional Magnetic Resonance Imaging (fMRI).

We implement our prototype using GW on top of Microsoft Azure to bridge the gap between PaaS and SaaS layers [32], since Microsoft Windows is used as a de facto platform by many brain imaging research groups and individuals. The prototype has been developed based on an open source toolkit called SPM¹ which utilizes MATLAB².

¹<http://www.fil.ion.ucl.ac.uk/spm/>

²<http://mathworks.com/products/matlab/>

The rest of this chapter is structured as follows: Section 8.2 describes how neuroscientists use SPM. Section 8.3 outlines the architecture of our prototype and the functions of the main components, such as the application manager, job manager, and data manager. Section 8.4 describe the security and privacy features of the application. In Section 8.5, we describe the deployment, integration and implementation aspects of SPM on Microsoft Azure using MATLAB. Finally, Section 8.6 summarizes the findings from the development of the application.

8.2 Statistical Parametric Mapping (SPM)

SPM is one of the most popular toolkits in neuroscience for running compute-intensive brain image analysis tasks. However, issues such as sharing raw data and results, as well as scalability and performance are major bottlenecks in the “single PC”-execution model.

The goal of the functional analysis of brain images is to find the parts of the brain that are activated when people (subjects) perform certain tasks. Since the signals that can be measured from the brain are noisy and there is considerable variation between individuals, many subjects, and many images of each subject’s brain, are required to get any statistically significant results from the analysis. There are also several parameters that need to be varied during the analysis. This means that the analysis normally has to be executed many times with different hypotheses and parameters.

A typical study consists of hundreds to thousands of 3D images of each subject; the resolution of a single image is normally in the order of a few mm which gives an image size of several Mbytes. There are normally 10 to 100 subjects in each study, and it usually takes over a day for a single analysis on a normal PC. Hence, there is a need for scalable compute and storage resources.

A typical analysis of brain images generally consists of several steps as shown in Figure 8.1:

- Re-align (compensate for subject head movement),
- Co-register (align structural and functional images),
- Normalize (transform to standard brain space),
- Segment (remove the skull bone etc. and leave only the brain),
- Filter (remove noise by low-pass filter), and
- Apply statistic methods, which normally use the General Linear Model (GLM).

After running all stages required for an analysis described in Figure 8.1, users may try to make inference using different parameters in their model or do Bayesian analysis or several other methods on the results. As an example, Figure 8.2 illustrates results of estimation stage - the last process in Figure 8.1- to make an inference by the user in a parametric approach.

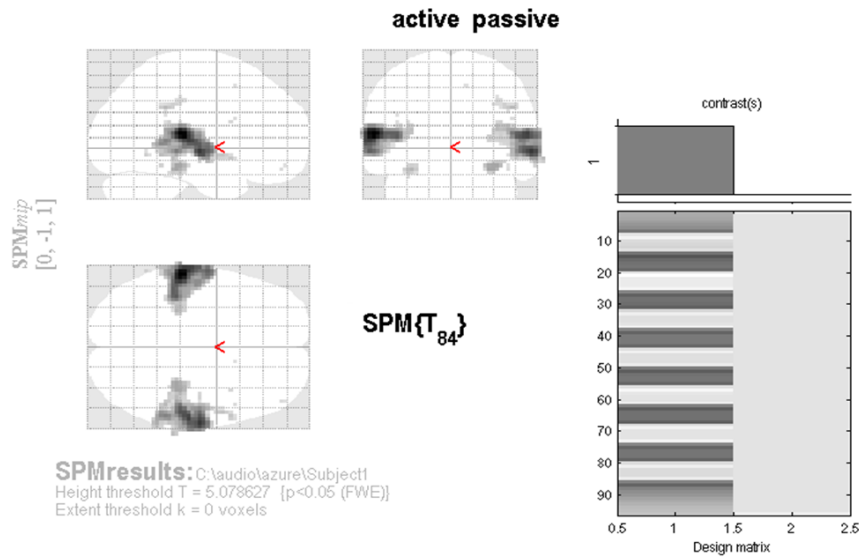


Figure 8.1: Resulting activation map of an experiment

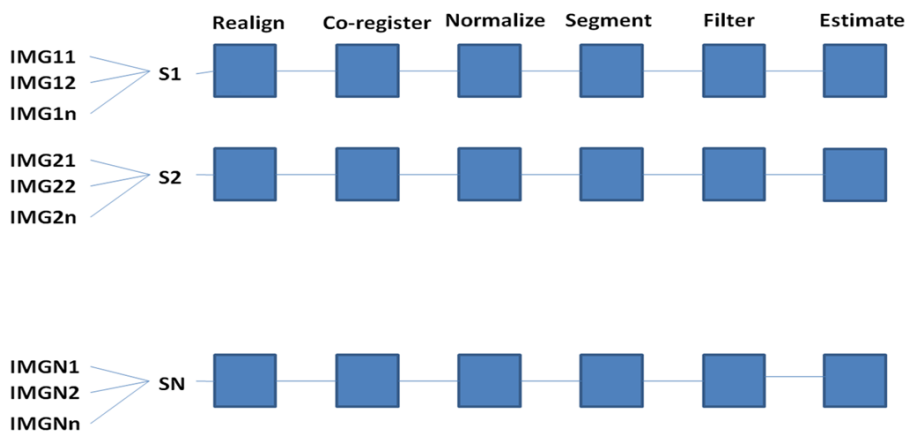


Figure 8.2: A series of stages to do an fMRI data analysis over N subjects (S_1, S_2, \dots, S_N) each subject i containing n images ($IMG_{i,1}, IMG_{i,2}, \dots, IMG_{i,n}$)

8.3 Design

In order to make it easy for researchers to use the cloud, the proposed design focuses on a system that would not require the end user to be aware of the complexity of cloud computing or of any dependencies on GW libraries. It was important to preserve the job execution style of SPM, so that users could run their brain imaging jobs using the MATLAB command line. Thus, in our architecture, we opted for a user-friendly interface, with minimal necessary dependencies on third party libraries (to provide a secure communication channel between the GW endpoint and the clients).

To enable users to communicate with and submit brain imaging tasks to the GW, four main components are included in this prototype: a security management, an application management, a job management and a data management, as illustrated in Figure 8.3.

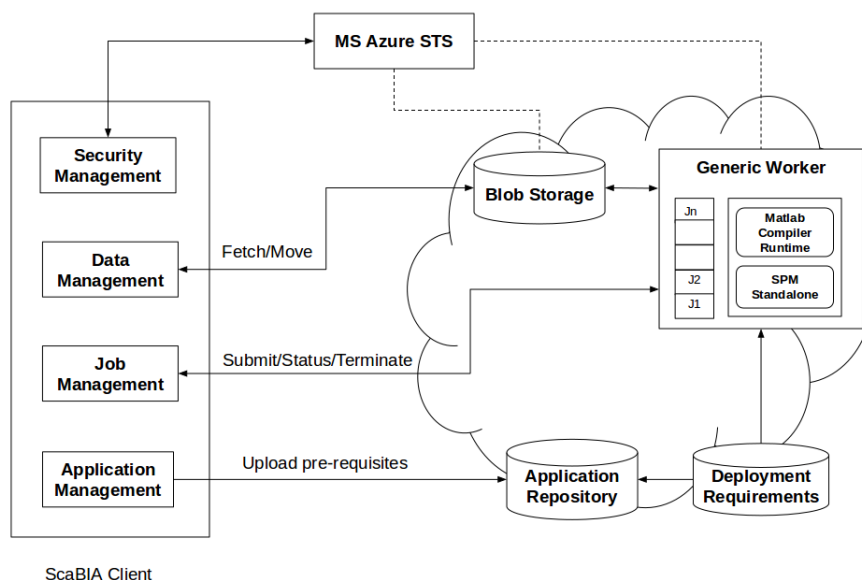


Figure 8.3: Architectural view of the ScaBIA in the Cloud

User requirements for execution of cloud-based SPM can be divided into three major areas. Users need to be able to prepare the execution infrastructure without experiencing difficulties caused by the cloud middleware. Secondly, the users need to be able to execute their jobs and keep track of all submissions. Finally, users need to be able to transfer and share data securely between the cloud storage and their local hard disk over the Internet without encountering cross-organizational restrictions introduced by the network services.

8.3.1 Security Management (SM)

The Security Management (SM) component ensures secure interaction of the user and clients to the SPM instances in the server. To achieve this, the SM authenticates the user with the STS and after successful authentication, the user presents the issued security tokens along with other credentials to access the services such as Data Management (DM), Job Management (JM), and Application Management (AM).

The user and application credentials are defined by the data owner in the Microsoft Azure portal. For example, the data owner can act as the administrator to assign permissions to users from a specific host which is identified through an X.509 certificate to access and perform experiments on the brain images in the cloud storage.

8.3.2 Data Management (DM)

Each SPM job requires hundreds of granular images that must be present for the job execution. Therefore, we required an effective data transfer solution and also had to be able to download results with names that humans could read. The DM enables users to upload or download data results from/to their local hard disk, and lets them rename data in the data storage. An excellent example can be a scenario where the user completes the analysis stages (Figure 8.1) and wishes to download the results locally to visualize the results within the SPM toolkit.

Furthermore, in a distributed environment which is typical for the research world, since users are scattered among different organizations, the DM provides a useful facility for sharing brain images between users and cloud applications. The DM acts as a cross-boundary client enabling users to perform create, read, update and delete (CRUD) operations. External users who wish to access the brain imaging data can easily invoke the client independent of their geographical locations.

8.3.3 Job Management (JM)

To submit SPM jobs, the user provides a job description defining the arguments for the job and their values, in addition to the SPM script created by the user through the SPM GUI (which is the real job that will be executed). The JM compresses all the brain images, together with the SPM scripts and job description, to submit to the GW endpoint. GW job description API implements Job Submission Description Language (JSDL) that is a specification to define submission aspects of jobs such as job name, resource requirements and so on [217]. There are some scenarios where data, such as brain images, are already stored in the data storage in the cloud and there is no need to re-transfer the input data. For instance, Figure 8.5 shows a situation where, in a chain of SPM tasks, the results from task $N - 1$ will be used as input for a new task N . In such scenarios, the user only needs to use the JM as a job submitter and provide the location of the input files within the data storage.

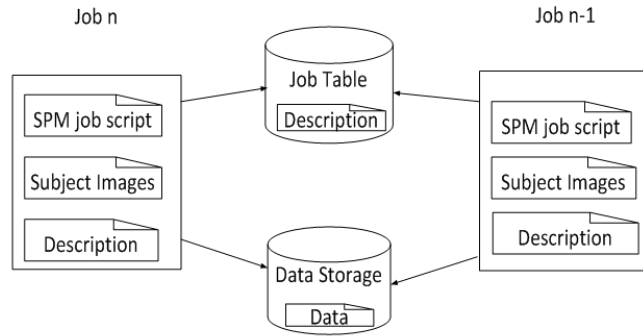


Figure 8.4: Job execution on a GW instance

Furthermore, the JM provides functions to check the status of submitted jobs and to terminate jobs at any stage of execution. To check the status of jobs, the JM periodically polls the GW with the job id returned from the earlier job submission steps. Thus, the JM can notify the job owner when a job is completed. Moreover, the JM is able to get the status of a list of jobs, or of all the jobs in the job table that belong to a specific user.

8.3.4 Application Management (AM)

Prior to accepting any requests from users, the cloud environment, has to set up any mandatory library and software. To successfully install an application in GW, the AM packs all the files that are necessary for running that application into a compressed file. The AM then creates an application description which defines the command line execution format for the application. Next, the AM uploads the application to the application repository that is based on the Azure blob storage. To do this, the AM has to serialize the application description into an XML structure that is uploaded to the application repository. Users need to remember the location of both the application and of the description for job submission purposes. As shown in Figure 8.4, the AM archives different user pre-requisites (for running a library-dependent job) into a single zip-file and uploads it to the pre-defined cloud storage.

8.4 Security and Privacy

To ensure confidentiality and integrity of brain images, strong authentication, RBAC and anonymization mechanisms have been implemented in our architecture.

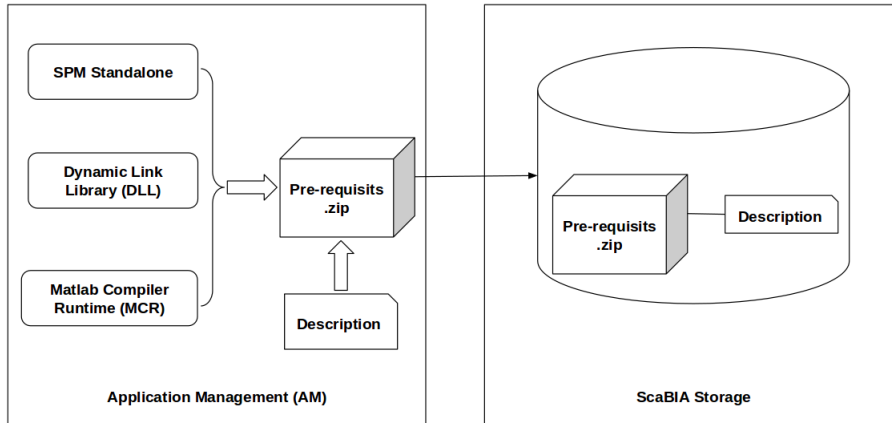


Figure 8.5: Installing the application requirements

8.4.1 Authentication

The GW operates in two modes: security with STS and security without STS. To ensure that only authorized users are allowed to register applications and submit jobs we used the security with the STS mode. We used the STS and added a thumbprint of a Secure Hash Algorithm (SHA-1) 2048-bits certificate with key exchange enabled in the Azure Portal.

We divided the certificates into two different categories: one for managing the deployment and another for securing the endpoints. In principle, user certificates are used when invoking commands to submit jobs to the GW or when establishing secure channels and encrypted communication SSL/STS to the Microsoft Azure platform, i.e., using management certificates for scaling the number of VMs running at the same time. Another certificate is needed for managing the Azure deployment for scaling the number of VMs. Azure uses PKI for managing the deployment. This way, a public key is configured in the Azure portal and whoever has the private key is able to manage the Azure resources. The users can obtain an X.509 certificate from a trusted certification authority or create a custom certificate through OpenSSL.

Microsoft Azure includes another layer of protection for storage. Therefore, users are required to create a storage account in the Azure portal. This will generate a pair of storage primary and secondary keys. The primary key is a mandatory 512-bits symmetric key (i.e., *AD5052E7C5A41C1B8862071555E91A3E157DD2BD*) that will be used as the main authentication key. The secondary storage key is optional to have and it can be generated independently from the primary key to act as a backup with similar access rights. A designated host with a management certificate is allowed to add or remove users belonging to different domains.

8.4.2 Authorization

There are two roles for setting up and using the platform: Administrator and Researcher. An administrator will set up the platform and have full access to the resources and adding/removing users while researcher has the possibility to upload and run SPM jobs. More precisely:

- **Administrator:** This role will assign new users to the Research role category or will revoke existing members for the group belong to the Researcher role through the Microsoft Azure management portal to a project, and
- **Researcher:** Users in the role are able to submit analytical SPM jobs to the GW and access the results.

Prior to any attempts by users to access the storage, the data owner must grant access privileges to the storage according to users storage access keys through Microsoft Azure management portal. The GW STS enforce authorized access after successful authentication of users.

8.5 Implementation

The GW provides a Microsoft.NET API for applications to implement all functionality that is needed to execute SPM job successfully. We integrated the GW and Software Development Kit (SDK) assemblies (DLLs) into the MATLAB R2011a environment. To relax the licensing issues, we used the MATLAB Compiler Runtime (MCR) Windows 64-bits version 7.15 associated with MATLAB R2011a [218].

8.5.1 Anonymization

To comply with the privacy regulations that demand protection of patient data such as HIPAA [9], we anonymized the brain images by removing any explicit identifier such as name, SSN or other QIDs including date of birth and ZIP code from the metadata. We used the open source Dicom toolkit³ (*dcmk*) to anonymize brain images by modifying the open source code of *dcmk*. Dicom tags including the identity and the name of the participants were removed from the datasets.

8.5.2 Secure Deployment of the Generic Worker

As a preliminary step, we deployed SPM using GW on top of Azure to facilitate deployment, initialization, and invoking SPM stand-alone over Azure without any modification of the SPM API. The Azure management portal provides user interfaces to upload security credentials, along with the GW and configurations. The GW is provided in different deployable packages as extra small, small, medium, large and extra-large [219] as summarized in Table 8.1.

³<http://dicom.offis.de/dcmk>

Instance	Cores	RAM	Disk Size
Extra Small	1	0.75 GB	20 GB
Small	1	1.75 GB	40 GB
Medium	2	3.5 GB	60 GB
Large	4	7 GB	120 GB
Extra Large	8	14 GB	240 GB

Table 8.1: Microsoft Azure basic tier general purpose compute

We deployed a production-hosted service composed of 20 medium-sized dual core machines and 3.5 Gigabyte memory with a bandwidth of 200 Mbit/s.

This hosted service also required an XML service configuration file that defines how the hosted service should run, for example, specifying the number of running instances or user and management X.509 certificate thumbprints, or other information (such as the job submission to GW end-points). Using OpenSSL we issued two self-signed X.509 certificates, one for management and one for job submission: both based on their distinguished names. We added the thumbprint of these certificates to the hosted service configuration file and uploaded the certificates to the management portal. The uploaded certificate contains both public and private keys in a single file, with a protected private key defined by the user's secret.

8.5.3 Building the Application

GW instances search for the MCR and standalone SPM dependencies during the initialization process. Therefore, prior to running the GW instances, we have to upload the MCR v7.15 and standalone SPM for Microsoft Windows 64-bits platform [220].

We implemented an MATLAB function to act on behalf of the user to compress the SPM dependencies and to upload them to the application repository using the following command line:

```
install_application(mcr_path,spm_path, app_rep)
```

The first argument of this command is the location of the MCR. The second and third arguments are the SPM standalone location and the name on Azure of the application repository name where the user pre-requisites should be stored. After successful installation of the application pre-requisites, the GW will load and install the pre-requisites for all the new VMs.

8.5.4 Job Submission

For job submission purposes, we developed two MATLAB functions. The user enters these on the command line to submit a job, either for a single subject or for

several subjects to run in parallel to ensure scalability for scenarios with a large number of subjects. The signature of the function to submit SPM jobs is as below:

```
submit_job(job_script, data_path, output_name, flag)
```

This function needs the following arguments: the real SPM script as the execution job, the path to the brain images, the name to be used for the output results, and a flag that is set in cases of multiple job submission (where an SPM job will be iterated over several subjects within different running GWs). To clarify, the SPM job is a script where the user creates an iteration of a set of instructions over a number of brain images for a group of one or more subjects. The second argument specifies the directory path where the images from that subject reside. Those images need to be uploaded with the submission of the SPM job. This command adapts the local file system names according to the GW standard by eliminating the root directory and replacing that with the current working path in GW. Figure 8.6 illustrates the process of creating SPM scripts by the user through the GUI and making them compatible to run on Azure by JM, in respect to the GW file addressing conventions.

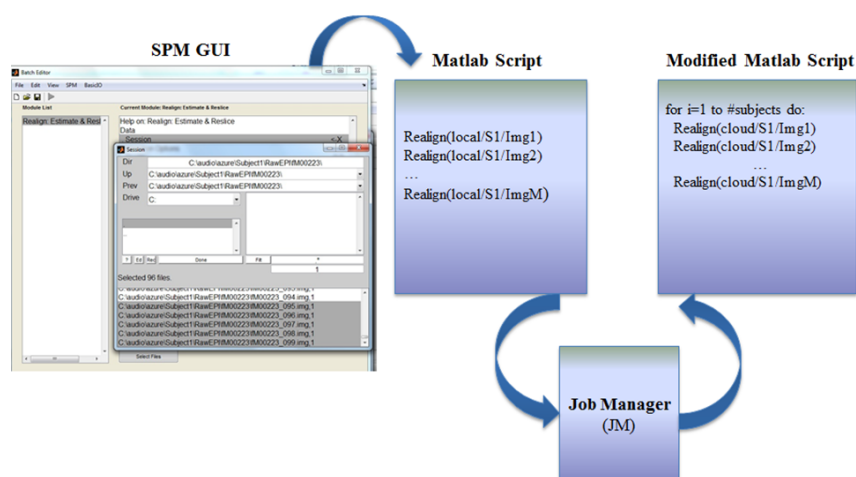


Figure 8.6: Process of creating SPM scripts and making them compatible with GW

Moreover, to store the result of the executions in the cloud storage, the user should define a name in the third argument that will be used for the output. There are some cases that do not require uploading of data when the clients submit jobs, for instance, in a specific scenario where job $n - 1$ produces results that are required by the next job n as input. In such cases, the user can pass an explicit “no_data”-signal in the flag argument. This notifies the application to fetch the input data from the storage.

To fulfill other job management tasks, we implemented the following functions

to track the status of submitted tasks, and to cancel a specifically submitted job on GW when required by user:

```
poll_status(id) and terminate_job(id)
```

The first function resolves a job identifier that is returned from the `submit_job` function and periodically polls the status of a defined job. The status could be Pending, Running, Finished or Canceled. Furthermore, the client is also able to terminate any job using its job identifier.

These two additional commands are very useful when an SPM user needs to submit a large number of jobs in parallel together on different running instances of the GW. All the submitted jobs can easily be tracked, and terminated instantly if needed, using these commands.

8.5.5 Data Management

In order to upload, download or eliminate brain images on the cloud storage, we developed a set of functions to enable SPM users to run those functions from the MATLAB command line. In the first step, the DM acquires a container reference from the storage using a GW API according to the container's name:

```
GetContainerReference(container)
```

The argument of this function refers to the container that the user provided to invoke the data transfer functions. We also increased the default time-out of the client (by setting the `TimeSpan` to a reasonable value) so that large amounts of data could be transferred successfully even if there were delays in the underlying communication networks or storage server timeouts.

To upload a file (that is, a set of brain images as a single file) to a specific container, the client should issue the command:

```
upload_subject(container, file_name, local_path)
```

Moreover, to facilitate the distributed access of data for a set of results, the client can query the container that stores a particular file. For this purpose, the user needs the file name that is stored in the container and the local path to save the result. The user must define at least a 512-bits storage access key as primary key, based on Azure management portal in combination with the storage account name to authorize user requests to storage services:

```
download_subject(container, file_name, local_path)
```

SPM users can issue another command to remove the storage contents associated with file names in distinguished containers. We implemented the following function with the same interaction pattern, with upload/download provided by the GW API:

```
delete_subject(container_name, file_name)
```

8.6 Summary

Our prototype demonstrated that cloud computing, and specifically Microsoft Azure, can enable neuroscientists to exploit the benefits of cloud computing while ensuring

security and privacy.

Users and organizations no longer need to be concerned about hardware requirements or storage capacities as the cloud provides scalability on demand. Also, our implementation shows that it is feasible to share anonymized data securely between users belonging to different organizations to avoid limitations of data privacy laws to process sensitive data in the cloud.

Part IV

**Secure Multi-Tenancy in the
Cloud**

Chapter 9

Quantifying and Minimizing the Risk of Kernel Exploitation

9.1 Introduction

As discussed earlier the underpinning for the clouds are infrastructures that virtualize the operating system. The kernel is the core software of the operating system that makes hardware usable and it provides the software that provides basic services for all other parts of the operating system. Kernel size in terms of LOC has been growing during the past decades which unfortunately has introduced or will introduce new bugs constantly over time [134, 221, 222, 223]. These vulnerabilities can be exploited by malicious attackers mainly to get privileged access to the underlying systems through executing untrusted code in userspace. An exhaustive list of such publicly known information security flaws can be found in the Common Vulnerabilities and Exposures (CVE)¹ database.

To achieve secure multi-tenancy through isolating untrusted code from the privileged code, the security community has introduced several protection mechanisms with the aim to restrict untrusted code to access unauthorized resources on the user machine such as user's data or hardware devices. For example, OS virtualization (e.g., Xen, KVM, VirtualBox), system call filtering [131, 129], and library OSes [126, 125]. Common security wisdom is that by running software in a virtual machine, one can prevent the attacker from exploiting flaws in the underlying kernel. However, the security of virtualization itself is a challenging issue [224] because some vulnerabilities on the Linux kernel are not prevented from being exploited.

Limiting access to kernel code by itself is insufficient to build a secure virtualization system because of two issues: 1) if a complex program can not access a part of the kernel, its functionality must exist somewhere else or the program will not work, 2) it is typical for virtualization systems to add new privileged code as their TCB. As a result, a vulnerability in the privileged codebase is as much of

¹<https://cve.mitre.org/>

a security risk as a flaw in the kernel. CVE-2008-2100 and CVE-2011-1751 are two examples of weaknesses in the virtualized system, where exploits from guest to host OS have been possible. In CVE-2008-2100, a vulnerability in VMWare's codebase was caused by buffer overflows. This could allow local users to bypass the guest VM and gain privilege escalation to execute arbitrary code in the host OS. In CVE-2011-1751, missing check in KVM's QEMU emulation of PCI-ISA bridge could allow an attacker for root exploit in the host OS being triggered from a guest [225].

This chapter presents a metric that helps to identify likely locations within the kernel where vulnerabilities may exist, based on examination of 40 kernel bug fixing patches [226, 227, 228]. We aim to validate the proposed metric using a dual sandbox that is implemented using the concept of safely-reimplement for most common kernel functionalities. Lind contains the POSIX implementation to limit access to the kernel. This implementation is integrated with the NaCl for software fault isolation through memory safety of the application.

The remainder of this Chapter is organized as follows. Section 9.2 presents the dual-layer sandbox architecture of Lind. Section 9.3 describes the key hypothesis and how the security of Lind is tested against other virtualization systems. Finally, in Section 9.4, we summarize our findings and concluding remarks.

9.2 Lind Dual-Layer Sandbox

Lind consists of two main modules: NaCl [137] as the computation module for efficient execution of legacy code in the form of x86 or Advanced RISC Machine (ARM) binaries and Seattle's Repty [226] as the library OS. The userspace application system calls are either dispatched through NaCl or directly invoked by the system call interface as shown in Figure 9.1 according to the policy definitions. The core functionalities of NaCl and Repty modules are described in the following.

9.2.1 Native Client (NaCl)

The NaCl module is used to isolate the user application activities from the underlying OS kernel in Lind. NaCl provides APIs that allow Lind to deploy and run legacy code by compiling the programs to produce a binary with the SFI property. This step prevents the majority of the application from performing system calls or executing arbitrary instructions.

When the compiled application is executed the invoked system calls jump into a small privileged part of the NaCl TCB that directs system calls to the OS for processing. In Lind, the system call forwarding is slightly different and the NaCl TCB forwards the call to the Lind library OS that we call SafePOSIX for the actual execution. The issued system calls then reach the kernel that is divided into several subsystems. For example in Linux there are mainly these subsystems: `ipc` (process management), `net` (network), `mm` (memory management), `fs` (file system),

security (security module), and `drivers` (device drivers). As we discussed earlier in Section 9.1, kernel modules might contain undiscovered bugs that can be triggered through executing untrusted code in userspace.

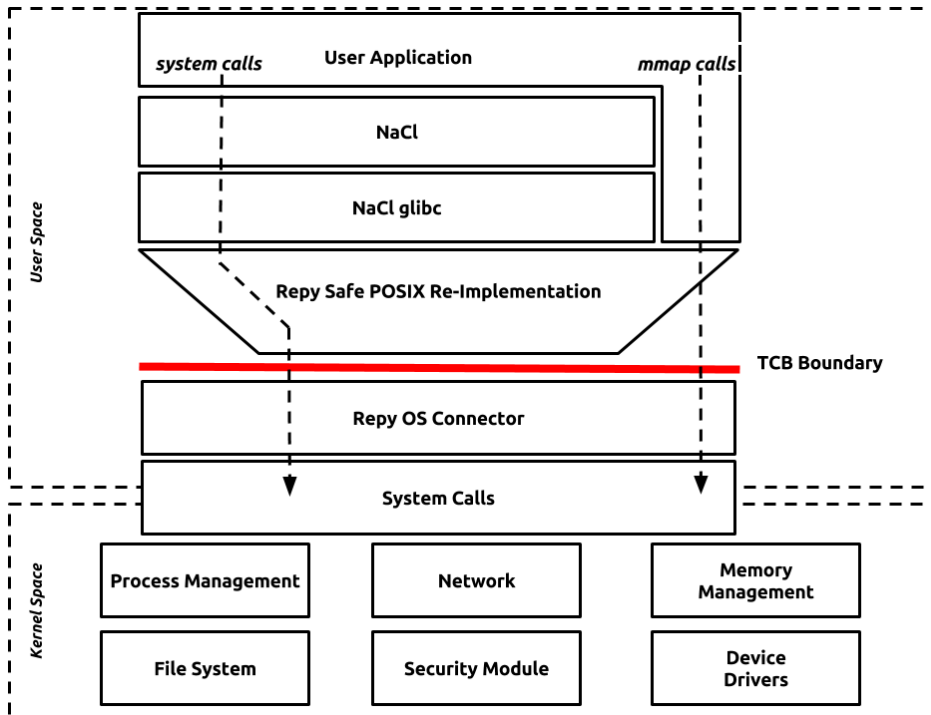


Figure 9.1: Architecture of Lind including various components such as NaCl, NaCl' glibc, and Repy Sandbox. User level applications will issue system calls that are dispatched through the Repy OS connector that bridges the Lind system to the OS Kernel.

9.2.2 Seattle's Repy

Lind isolates computation and only allows access to commonly used kernel paths in order to build an API to access the safe parts of the underlying kernel. Seattle's Repy [226] sandbox is the key component in Lind that provides this functionality. The POSIX API is used to run applications within the sandbox. Lind integrates Seattle's Repy system API that is a tiny sandbox comprised of around 8K LOC. Repy provides Lind with the ability of access to a minimal set of the system call API needed to build general computational functionality.

Repy provides access only to the safe portions of the OS kernel with 33 basic API functions, including 13 network functions, 6 file functions, 6 threading functions, and 8 miscellaneous functions [226, 229]. The code is written using style guidelines designed to ease security auditing of the code [230], as shown in Table 9.1.

Table 9.1: Repy sandbox kernel capabilities that supports NaCl functions, such as networking, file I/O operations and threading.

Repy Function	Available System Calls
Networking	<i>gethostbyname, openconnection, getmyip, socket.send, socket.receive, socket.close, listenforconnection, tcpserversocket.getconnection, tcpserversocket.close, sendmessage, listenformessage, udpserversocket.getmessage, and udpserversocket.close.</i>
I/O Operations	<i>openfile, file.close, file.readat, file.writeat, listfiles, and removefile.</i>
Threading	<i>createlock, sleep, lock.acquire, lock.release, createthread, and getthreadname.</i>
Miscellaneous Functions	<i>getruntime, randombytes, log, exitall, createvirtualnamespace, virtualnamespace.evaluate, getresources, and getlasterror.</i>

The Repy functionalities have been used since 2010 in real-life applications in the Seattle’s [231] testbed. It has been also audited through penetration testing and unit testing for each system call. The system call requests are issued from the userspace code are received by NaCl and then will be redirected to the system API module that includes a POSIX API to serve those requests. The POSIX API contains a set of standard interfaces that deliver the OS functionalities to the userspace code. The POSIX API could become big and complex enough which makes it hard to assure as a bug-free piece of code. To resolve this issue, Repy as a programming language sandbox has been used in Lind to provide the ideal isolation needed to construct the custom POSIX API. In Lind, complex system functions are reimplemented using the Repy code, based on the “safely-reimplement” principle.

9.3 Quantitative Evaluation

This section describes our approach for quantitatively evaluation risk of kernel exploitation and measuring the effectiveness of Lind compared to other systems.

9.3.1 Hypothesis

The key hypothesis of our approach is that executed kernel paths by popular applications are likely to contain fewer exploitable bugs than uncommonly used paths.

Because if there are bugs in such common kernel paths they are most probably identified before being exploited by malicious adversaries.

To test our hypothesis we performed an analysis of two different versions of the Linux kernel, 3.13.0 and 3.14.1 at the level of lines of code against existing kernel bugs for a better understanding of the security features of the OS kernel. The kernel coverage safety metric provide us information about the potential risks and consequently, it helps us to safely-reimplement those risky kernel interface (or system calls) to avoid the potential exploits. We also selected a variety of CVE bug reports to experiment and understand which parts of the kernel have been used in common (Section 9.3.2).

9.3.2 Data Sources and Experiments

To compare the efficiency of security in various virtualization mechanisms compared to our approach in Lind, we selected several other popular systems such as VirtualBox, VMWare Workstation, Docker, LXC, QEMU, KVM and Graphene [127]. Naturally, we also included measuring native Linux as a baseline for our evaluation due to lack of any virtualization. These experiments were decided to be conducted under Linux kernel 3.14.1. Figure 9.2 demonstrate the activities from trace generation performed in target virtualization systems and transforming the collected traces for code-level analysis.

Data sources We decided to set up our experiments in three major environments: Lind platform, virtualization platforms, and native Linux. For each category, we ran a group of legacy applications, comprehensive fuzzing using Trinity system call fuzzer, a group of Linux Test Project (LTP) programs and 35 CVE exploits.

- **Legacy applications:** Common paths were captured by running popular user applications or open-source libraries. Most notably, these experiments were conducted in data collection phase included running several applications such as two large-scale browsers (Mozilla Firefox and Google Chrome), in addition to running 50 packages among the top 200 popular Debian packages [232]. To port these packages to Lind, first we compiled them with NaCl Pepper version 39² and then run the executables under Lind. Listing C.1 in Appendix C shows an example script to port a curl library into Lind through the NaCl toolchain.

During this step, we conducted several other operations needed to access common paths, including intensive file management tasks to create/read/update or delete files and directories from the underlying file system. These tests were completed during a discrete period of 20 hours over 5 days.

- **Trinity Fuzzer:** The system call fuzzing experiments were designed to utilize the Trinity system call fuzz tester [233]. This included sequential execution

²<https://developer.chrome.com/native-client/sdk/download>

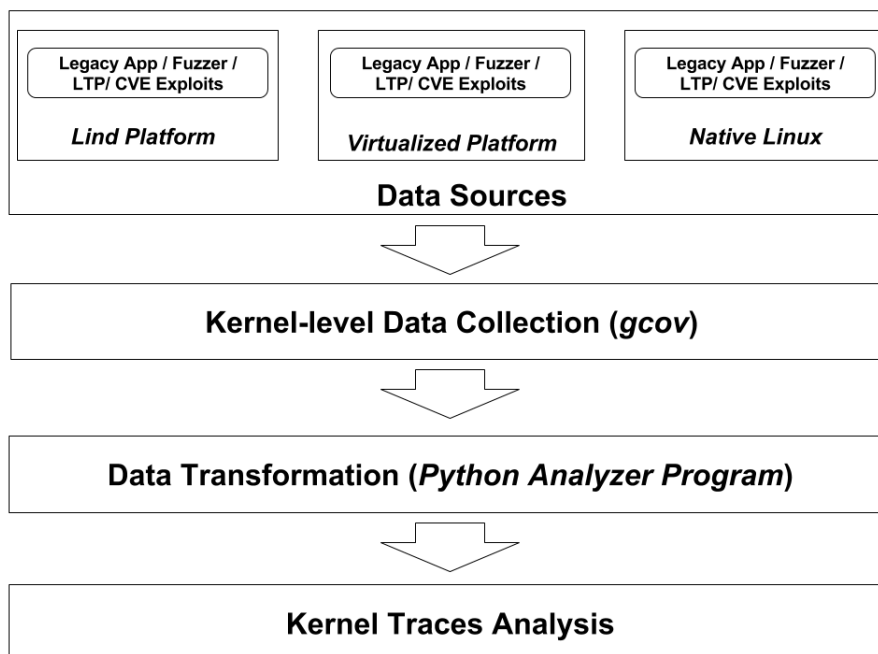


Figure 9.2: Various activities performed to capture and analyze the kernel traces generated by legacy applications, system fuzzers, LTP, and CVE bug reports. The traces are collected using `gcov` and a Python-based program that transforms the `gcov` data to macrodata-level information of each traversed path for final data analysis.

of more than 300 system calls with 1 million iterations for executing each system call by 16 child processes as Trinity workers. Trinity traces were used as an indication of the total reachable paths to be captured by comprehensive system call fuzzing and running the Linux kernel test suite.

- **LTP:** We also used LTP [234] to capture the total reachable kernel trace as it provides a mature and well-maintained tool to test the Linux kernel. We tested all the available system calls in LTP, in addition to 23 test cases: *accept*, *bind*, *chown*, *close*, *connect*, *date*, *exit*, *exit_group*, *getegid*, *geteuid*, *getgid*, *getpeername*, *getpid*, *getsockname*, *getsockopt*, *getuid*, *listen*, *rmdir*, *setsockopt*, *sleep*, and *socketpair*³. This traces helped us to validate the kernel trace generated by Trinity and also catch any possible missing paths. More precisely, the LTP was able to capture approximately 20% of kernel trace that

³https://github.com/gholamiali/ltp_tests.git

was ignored by Trinity. While Trinity traces included 15% of kernel trace that was not available in the LTP traces.

- **Linux Kernel Bug Test and Evaluation:** We compiled and ran the exploit C code under each virtualization system to obtain their kernel traces, and then used our kernel trace safety metric to determine if a specific bug was triggered. To this end, we reused some existing C code that was available publicly on the Internet with the capability of exploiting each of the kernel bugs [235] as shown in Table 9.2.

Vulnerability	Description
CVE-2015-5706	allows local users to cause a denial of service
CVE-2015-0239	allows guest OS users to gain guest OS privileges or cause a denial of service (guest OS crash)
CVE-2014-9584	allows local users to obtain sensitive information from kernel memory
CVE-2014-9529	allows local users to cause a denial of service (memory corruption or panic)
CVE-2014-9322	allows local users to gain privileges
CVE-2014-9090	allows local users to cause a denial of service (panic)
CVE-2014-8989	allows local users to bypass intended file permissions by POSIX ACL
CVE-2014-8559	denial of service (deadlock and system hang)
CVE-2014-8369	allows guest OS users to cause a denial of service (host OS page unpinning)
CVE-2014-8160	allows remote attackers to bypass intended access restrictions
CVE-2014-8134	allows guest OS users to bypass the Address Space Layout Randomization (ASLR) ACL protection mechanism
CVE-2014-8133	allows local users to bypass the ASLR protection mechanism
CVE-2014-8086	allows local users to cause a denial of service (file unavailability)
CVE-2014-7975	allows local users to cause a denial of service (loss of writability)
CVE-2014-7970	allows local users to cause a denial of service (mount-tree loop)
CVE-2014-7842	allows guest OS users to cause a denial of service (guest OS crash)
CVE-2014-7826	allows local users to gain privileges or cause a denial of service (invalid pointer dereference)
CVE-2014-7825	allows local users to cause a denial of service (out-of-bounds read and OOPS)

CVE-2014-7283	allows local users to cause a denial of service (filesystem corruption, or panic)
CVE-2014-5207	allows local users to gain privileges
CVE-2014-5206	allows local users to bypass an intended read-only restriction and defeat certain sandbox protection mechanisms
CVE-2014-5045	allows local users to cause a denial of service (memory consumption or use-after-free)
CVE-2014-4943	allows local users to gain privileges
CVE-2014-4667	allows remote attackers to cause a denial of service (socket outage)
CVE-2014-4508	allows local users to cause a denial of service (system crash)
CVE-2014-4171	allows local users to cause a denial of service
CVE-2014-4157	allows local users to bypass intended access control restrictions
CVE-2014-4014	allows local users to bypass intended chmod restrictions by first creating a user namespace
CVE-2014-3940	allows local users to cause a denial of service (memory corruption or system crash)
CVE-2014-3917	allows local users to obtain potentially sensitive single-bit values from kernel memory or cause a denial of service
CVE-2014-3153	allows local users to gain privileges
CVE-2014-3144	allows local users to cause a denial of service (integer underflow and system crash)
CVE-2014-3122	allows local users to cause a denial of service (system crash)
CVE-2014-2851	allows local users to cause a denial of service (use-after-free and system crash)
CVE-2014-0206	allows local users to obtain sensitive information from kernel memory

Table 9.2: Exploitable CVEs that we triggered under VirtualBox, VMWare Workstation, Docker, LXC, QEMU, KVM and Graphene virtualization systems

9.3.3 Kernel-Level Data Collection

We used `gcov` [236] which is a standard utility with the GNU compiler collection suite⁴ to get the kernel footprints. `gcov` provides a profiling tool that indicates which lines of kernel code are executed while an application runs. However, `gcov` is not installed by default in the Linux kernel distributions and we had to recompile and install a new Linux kernel to enable this feature. The trace collection procedure was as the following.

⁴<https://gcc.gnu.org/>

1. Compile and boot kernel with *gcov-kernel* active
2. Use *lcov* to clear all coverage: *lcov -z*
3. Run the program inside Lind, virtualized environments and native Linux
4. Collect coverage using *lcov:lcov -c -o coverage.info and genhtml coverage.info -o coverage.out*

9.3.4 Data Transformation

We implemented a Python parser based on the **gcov** data to identify the usage of each line of code for each kernel module C.2. This program generates a report from kernel areas that are triggered by an application (i.e. *coverage.out* from Section 9.3.3). It prints out the lines of each source file that are used according to the coverage info from *gcov-kernel*. Each source file might have been run several times and therefore, we aggregate all the executions in one source file.

9.3.5 Kernel Traces Analysis and Evaluation

We evaluated Lind's efficiency for the confinement of untrusted code and protecting the OS kernel and subsequently other processes within the same host. To this end, we conducted numerous experiments designed to proof how Lind assure isolation to achieve better multi-tenancy compared to other virtualization systems against historical Linux kernel bugs or how much of the underlying kernel code is exposed and, therefore, vulnerable to different virtualization systems?

The preliminary evaluations suggest that Lind outperforms other isolation mechanisms due to applying a dual-layer sandbox. The first security layer of Lind contains a tiny TCB and SafePOSIX API based on the concept of safely-reimplement which makes it an excellent choice to prevent untrusted programs to exploit any i.e., zero-day bugs. Adding the NaCl layer makes it nearly impossible for the userspace applications to violate the pre-define policies, i.e., restricting a subset of system calls that are not permitted by the untrusted programs.

Furthermore, we collected the results for different systems and measured how much of the underlying kernel is reached during our experiments. As an example, Figure 9.3 demonstrate that 12.4 % of the kernel surface were reachable when performing the fuzzing experiments using LTP and Trinity, while 55.4% of the kernel were not reachable at all, which is a promising indication of Lind effectiveness to protect a significant portion of the kernel.

9.4 Summary

This chapter presented Lind - a novel confinement solution to achieve higher levels of multi-tenancy through isolation of userspace applications. In addition, we introduced a hypothesis to investigate the risky portions of the OS kernel that contains

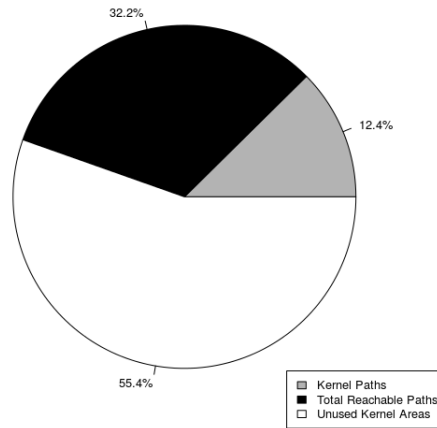


Figure 9.3: Percentage of different kernel areas that were reached during LTP and Trinity system call fuzzing experiments to measure the reachable kernel surface

higher rates of bugs. For this purpose, we designed a metric and performed extensive experiments within Lind and other existing virtualization systems that are widely used to isolate untrusted applications that share the same hardware.

The preliminary results show Lind can prevent zero-day bug exploits to a greater extent than other isolation approaches due to its architecture. This indicates a positive proof of our hypothesis that Lind can be efficient to restrict access to untrusted programs to the OS kernel. However, the evaluation part still is ongoing research and we would like to gather more data to strongly proof efficiency of our metric.

The next chapter of this thesis discusses the implementation of a reference monitor that was developed in the course of this work for debugging purposes. This reference monitor can be used to apply different policies to enable/disable specific system calls that interact with the Repty sandbox.

Chapter 10

Lind Reference Monitor

10.1 Introduction

There are several reference monitor systems that have been proposed to enforce security properties on programs they isolate. These systems employ trusted code which monitors and restricts the set of system calls a program is able to execute. We call such systems check-and-pass-through security systems. If an application issues an untrusted call, it is blocked and thus prevented from causing damages to the underlying infrastructures or other tenants in the same physical host.

Unfortunately, despite great efforts to develop research systems such as [130, 131] or Ostia [129], we could not apply them to the Lind architecture to enforce custom system call filtering policies. The main reason was compatibility of these approaches with the POSIX API inside Repty.

In this chapter, we discuss design and implementation of a reference monitor to be applied in Lind. This enables us to verify the intended functionalities of Lind and also facilitate debugging of potential bugs in the SafePOSIX implementations, i.e., allowing or denying a specific set of system calls to monitor the Lind core components behavior.

The rest of this chapter is organized as follows. Section 10.2 presents an overview of the proposed reference monitor architecture. Section 10.3 describes implementation of each components such policy management or system call filtering. Section 10.4 discusses the validation process to verify the proposed solution. Finally in Section 10.5, we present our findings and conclusions.

10.2 System Call Interposition Model

The Lind reference monitor is a standalone module that aims to intercept system calls that are issued by a client program for isolation purpose. A client program is an untrusted program that our reference monitor wishes to execute without negatively impacting the security of the executing system.

Figure 10.1 presents the proposed architecture for system call interposition over Repy connector. Each system call will be categorized as allowed by Lind, allowed by OS or denied.

- If a system call is allowed to be executed by Lind, then the associated system call in Safe POSIX in Repy will be invoked.
- If a system call is allowed to be executed by OS, then it will be forwarded directly to the underlying kernel to be executed.
- If a specific system call is denied to be invoked, then the reference monitor aborts the execution of that call immediately.

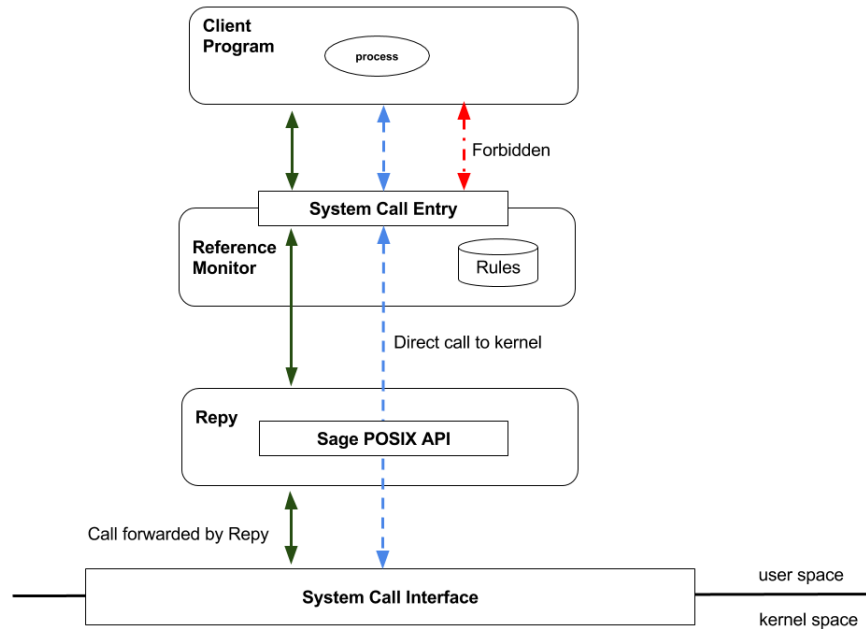


Figure 10.1: Reference monitor architecture

These functionalities are delivered through two components: policy configurations and system call filtering inside the reference monitor.

10.2.1 Policy Configurations

The policy configuration component includes management of system call interposition rules. There is a definition of permitted system calls in each module and such

information will be loaded in the runtime to be accessed by the system call filtering component.

All the available system calls in Lind have API defined in this component to ensure appropriate marshaling or unmarshaling of the arguments needed to be passed to the underlying API. Such arguments are originated from the intermediary layer that intercepts the calls and redirects them to the appropriate system.

10.2.2 System Call Filtering

This component enforces the pre-defined policies from the rules database to allow only permitted system calls to be executed by Lind or by the underlying OS kernel. The system call filtering intercepts the incoming calls and determines the number of the system call. This way, the associated number of the system call can be correlated with the SafePOSIX implementation which can have different values depending on the hardware, i.e. x86_64 or ARM.

10.3 Implementation

To provide an efficient implementation that is capable of manipulating the system call arguments we decided to use C. However, the reference monitor should be capable of communicating with the SafePOSIX API that are a safe subset system calls implemented in Python. For this purpose, we used a Python extension library in C, as it has been illustrated in Listing 10.1.

Listing 10.1: Invoking SafePOSIX system calls through Python extension in C

```
PyObject* MakeLindSysCall(int syscall, char* format, ...) {
    PyObject* callandarg = NULL;
    PyObject* response = NULL;
    PyObject* args = NULL;
    PyGILState_STATE gstate;
    va_list varg;

    gstate = PyGILState_Ensure();
    va_start(varg, format);
    args = Py_VaBuildValue(format, varg);
    va_end(varg);
    callandarg = Py_BuildValue("(i0)", syscall, args);
    response = CallPythonFunc(context, "LindSyscall", callandarg);
    Py_XDECREF(callandarg);
    Py_XDECREF(args);
    PyGILState_Release(gstate);
    return response;
}
```

}

To intercept and manipulate the system calls we developed our solution using a standard Linux system call that is called *ptrace*¹. As the first step, we initialize a process to be traced using *fork()* system call. The monitor checks if the fork was successful otherwise aborts the execution. In the case of success, the parent process starts to trace the child by initiating a *execve()* system call (see Appendix D Listing D.2).

We implemented a Linux shell script that depending on the underlying OS generates the list of available system calls. These system calls are fed into a configuration file for the monitor to define the rules to be enforced for the incoming calls from the client program (see Appendix D Listing D.1).

Furthermore, we implemented all the system calls in the configuration management component to forward incoming calls to the SafePOSIX API. Listings D.3 and D.4 in Appendix D demonstrate our approach to convert the incoming calls to be compatible with Lind and forwarding them to the Repy server.

Client programs can be easily run inside the reference monitor by passing the name of the executable binary as an argument. For example *\$reference_monitor myapp* will execute *myapp* which is an executable binary that is available in Repy file system.

10.4 Validation

We executed the reference monitor for several popular Debian libraries and packages such as editors, text processing and also apache web server. These experiments enabled us to discover few bugs within the Repy server implementation through applying different system call forwarding policies. In addition, we implemented unit tests for all the available system calls in Lind and we experimented them to verify the behavior of each system call when communicating with the Repy server. However, we did not perform any overhead measurement of applying the *ptrace*-based reference monitor because our intention was to only debug the possible bugs in the Lind TCB.

10.5 Summary

This chapter described design and implementation of a reference monitor for Lind that can be used for debugging purposes in the Lind dual-sandbox. For this aim, we developed a solution using the *ptrace* system call to intercept all the incoming calls from the client program and apply appropriate rules.

Our solution demonstrated to be able to run real life programs and to be used inside Lind for debugging and getting insight into various functionalities that are implemented in the TCB.

¹<http://man7.org/Linux/man-pages/man2/ptrace.2.html>

Part V

Epilogue

Chapter 11

Discussion

In this chapter, the work described in the previous parts (I, II, III, and IV) of this thesis are summarized and discussed. Let us start with the following research questions from Chapter 1.

- **Q1:** Can we develop a methodology to formulate privacy requirements and threats to facilitate compliance with data protection regulations?
- **Q2:** How to build privacy-preserving cloud-based systems from existing approaches in security and privacy?
- **Q3:** How to increase the safety of an OS by reducing the risk of kernel exploits?

We briefly discuss what strategies were used to answer the first question (Q1) in Section 11.1. Our approach to answering the second question (Q2) is discussed in Section 11.2. Finally, Section 11.3 discusses the last research question (Q3).

11.1 Discussion on Formulating the Cloud Privacy Requirements

Privacy of individuals in cloud computing encompasses several contexts depending on the nature of the sensitive data to be protected or the applicable data protection regulation. In this thesis, we emphasized the security and privacy issues of medical health care clouds that process NGS, clinical or brain imaging data.

For this purpose, we surveyed the literature in software engineering methodologies that offer comprehensive guidelines for building software correctly. In this step several existing security threat modeling approaches such as STRIDE [179] or OCTAVE [184] were studied but, unfortunately, we could not find any evidence of their usage to build privacy-preserving cloud computing in the healthcare domain. Additionally, privacy-preservation was not emphasized in those methodologies which cause substantial overhead when building cloud systems. To this end,

we built a new Cloud Privacy Threat Modeling (CPTM) methodology according to the Method Engineering (ME) method [35] that can efficiently facilitate the process of privacy threat modeling.

11.1.1 Cloud Privacy Threat Modeling

To build the CPTM, we first applied the ME guidelines to identify the characteristics that satisfy the requirements of such methodology. This included applying an ‘Extension-based’ paradigm for enhancing the process of identifying privacy threats by applying meta-models/patterns and predefined requirements. The demanded features to build the threat model were 1) privacy legislation support, 2) support for technical deployment and service models, 3) customer needs, 4) usability, and 5) traceability.

Afterward, the CPTM was proposed consisting of five products: privacy regulatory compliance, cloud environment specification, privacy threat identification, risk evaluation, and threat mitigation. These products will be used to compose a threat modeling artifact that can be used in the design step by the cloud software development teams to deploy a privacy-preserving solution.

This new methodology was applied to an emerging BioBankCloud in the context of the EU DPD. As the first step, key requirements of the DPD were identified. Physical and logical architectures of the target cloud environments were sketched to highlight any possible threat that poses a risk to the identified privacy requirements. As the next step, a risk evaluation of the identified threats was conducted to prioritize the threats according to their overall impact on the developing cloud environment. Furthermore, several countermeasures were proposed to mitigate the possibility of realizing the platform threats through adversarial attacks. The outcomes of this proof of concept were 8 key privacy requirements and mitigation recommendations for 26 critical threats to the privacy of genomic data in a PaaS. These facts enabled us to build and validate the prototype in part II.

11.2 Discussion on Building Privacy-Preserving Cloud Solutions

To explore the possibility of using privacy-preserving clouds, we proposed three usable privacy-preserving cloud-based architectures to process genomics, clinical and brain imaging datasets. For this purpose, we conducted a comprehensive survey on cloud provider activities to identify the state-of-the-art and existing approaches that could benefit us in this research.

We realized that the requirements of cloud based solutions vary from the context of the applicable privacy laws to the technical limitations of existing security solutions. We could not find a common set of security standards or technologies that fit the three architectures proposed in parts II and III. The main reason was the ethical and regulatory considerations when processing different types of sensitive

data. The existing laws and legislation are complex enough by nature that require appropriate considerations prior to the development of cloud-based solutions. The second reason was the architectural trade-offs in each cloud environment that make it complicated to integrate different pieces of security standards or existing technologies. We faced several shortcomings of existing security solutions to achieve strong security while offering a practical level of usability in our proposed solutions. For example, to offer several authentication mechanisms from the simplest approach based on username/password to more secure mechanisms such as two-factor authentication or X.509 certificates for different classes of users. To offer these authentication mechanisms there is no yet a single solution available that we are aware of, in best of our knowledge. Therefore, we extended and implemented our custom libraries and plugins to achieve our architectural goals for each target system.

However, a common balance in our proposed solutions was between usability vs. security and privacy. With a security mindset, we offered trade-off architectures that take into consideration a reasonable level of usability. But when we validated the BioBankCloud prototype by end-users we experienced a considerable demand for usability. Otherwise the proposed solutions were yet another piece of software out there with no direct usage.

11.3 Discussion on Quantifying and Minimizing the Risk of Kernel Exploits

The OS kernel is one of the most critical components of cloud computing because all other layers of the cloud stack rely on this part. Ensuring strong isolation between user processes in PaaS and SaaS layers are crucial to offering secure multi-tenancy in cloud-based solutions.

For this purpose, we made a hypothesis that most kernel exploits reside in the portion of the OS kernel that is not reachable by most popular applications. To verify this hypothesis, we designed a metric to quantify the risk of kernel exploits based on the collection and analyzing the kernel traces from various existing virtualization technologies against Lind - a dual layer sandbox based on the Repty SafePOSIX and NaCl that aim to deliver strong isolation in multi-tenant cloud environments.

Our experiments to collect the kernel traces were composed of reproducing approximately 40 CVE exploits in a Linux Kernel, in addition to running existing fuzzers such as LTP or Trinity to collect a fair amount of kernel usage. As another source of kernel traces, we developed and experimented with a set of unit tests for the supported system calls in Repty and NaCl. All these data sources provided us evidence to verify our hypothesis.

The preliminary results of our evaluations suggest that most existing kernel bugs are located in areas of the kernel that are not reachable by common applications and, therefore, Lind SafePOSIX API that is built on the concept of “safely-reimplement”

can significantly isolate the untrusted client programs in the user space from accessing the underlying kernel space functions. That is if an attacker tries to exploit a zero-day vulnerability in the host OS, it will be restricted in the Lind dual-layer of protection.

Chapter 12

Future Work

In this chapter, work from the previous parts (I, II, III, and IV) of this thesis that leads to many other interesting future research questions is discussed. Section 12.1 outlines the future research in the area of privacy by design. Then in Section 12.2, new dimensions to be explored in the area of trustworthy privacy-preserving cloud models are discussed. Finally, Section 12.3 discusses the new questions and future research in the area of secure multi-tenant cloud environments.

12.1 Privacy by Design for Cloud Computing

The future research related to the part II are categorized into three dimensions. First, studying the feasibility of the CPTM for other sectors. Second, validating CPTM against emerging data protection legislation and third, enhancing the security and usability of the BioBankCloud.

12.1.1 Applications of the CPTM in Other Domains

As discussed earlier, we developed the CPTM as an applicable methodology to different ecosystems such as healthcare, telecoms or finance. This thesis presented a proof of concept for the healthcare clouds as an appropriate choice. Because healthcare clouds usually process medical data which contains highly sensitive information.

An interesting research question that can be investigated in future work is the feasibility of using CPTM in other domains. For example in IoT where devices produce a huge amount of traffic data events including Call Detail Record (CDR)s that contain sensitive information. For organizational cost reduction benefits, such data can be stored in the cloud. The proposed model can be applied to identify the strengths or shortcomings for further improvements.

12.1.2 Emerging Data Protection Laws

Data protection regulations and laws are evolving. For example, in May 2016, there will be a new law to replace the EU DPD. The new regulation contains more than 130 recitals and 91 articles in addition to including new roles such as Data Protection Officer [187]. Hence, further expanding the CPTM for compatibility with this new law, indeed, would be useful in designing privacy-preserving cloud systems.

12.1.3 Security and Usability of the BioBankCloud

Another aspect that can be interesting for future research is to provide more usability and enhancing the existing functionalities e.g., resilience against SQL injection attacks for some internal queries. We already built a fully functional prototype, however, it will be interesting to add more features in addition to performing more bug analysis against the existing implementation to discover any potential bugs in the software. This includes:

- **PPH:** We designed a password protection mechanism in the later stage of the BioBankCloud project. We implemented a library using Java [237] to provide support in the IaaS level. Therefore, this integration was out of the scope of the BioBankCloud that was built on a PaaS cloud. This implementation can be integrated to efficiently protect user passwords and answer to security questions.
- **Federated Authentication:** As we described earlier, there were technical limitations for using federation to outsource the authentication to the users home organizations. The main issue was the fact that Shibboleth as a popular SAML implementation did not support Glassfish that was tightly integrated with the BioBankCloud infrastructure. In future work, it will be interesting to replace the Glassfish with a more mature software such as Apache Tomcat in the BioBankCloud. This makes it possible to authenticate users from their home institutions to achieve better security and usability.
- **Secure Data Discovery:** Providing the functionality of secure browsing of the platform for external users to discover the public datasets via REST calls is another area to be further explored.

12.2 Trustworthy Privacy-Preserving Cloud Models

The proposed eCPC toolkit in part III was able to anonymize sample availability clinical data efficiently and upload the anonymized data to a trusted third-party SaaS cloud. We were able to verify and validate the functionality of the client toolkit. We proposed a design for the integration server and partially implemented some basic functionalities such as uploading data to the server. However, we believe

that several components from the BioBankCloud can be reused in the integration server and this will be a straightforward engineering effort that is out of the scope of this thesis.

Other interesting dimensions to explore in this area are: 1) comparison of the eCPC approach with a secure multi-party computing to provide the functionality of cross-linking over multiple data sources, and 2) extending the RBAC model to a DAC model for more fine-grained sharing of the brain imaging data between researchers. In this way, data providers can directly grant privileges to other researchers with ought being concerned to have a wide level of permissions for a group of users.

12.3 Secure Multi-Tenancy in the Cloud

To provide meaningful insights into secure multi-tenant solutions, it would be very beneficial to perform a deep bug analysis for various virtualization systems such as Docker, QEMU, LXC, VMWare, and Graphene against Lind. This will make it possible to demonstrate the full quantitative measures for the security of Lind. This work is currently ongoing research and we anticipate to get the quantitative measures in a near future.

Lind has not been optimized for performance, so the results we present here should be taken as a baseline. We would like to explore what existing OS VM optimizations can be safely applied to enhance the overhead caused by the dual-layer sandbox compared to a virtualization system that performs emulation.

We would also like to test our metric in other popular operating systems, such as Windows and Mac OS. Our experiments were limited to Linux kernel 3.14.1 and some of the typical virtualization systems that existed in Linux. It would be interesting if similar tests could be run in other widely-used operating systems. In particular, it would be interesting to see if having the host and guest VM utilize different operating systems provides better protection.

Bibliography

- [1] R. A. S. NIST Big Data Public Working Group, “DRAFT NIST Big Data Interoperability Framework,” April 2015. Accessed July 2015.
- [2] F. Liu, J. Tong, J. Mao, R. Bohn, J. Messina, L. Badger, and D. Leaf, *NIST Cloud Computing Reference Architecture: Recommendations of the National Institute of Standards and Technology (Special Publication 500-292)*. USA: CreateSpace Independent Publishing Platform, 2012.
- [3] B. Russell, “Realizing Linux Containers (LXC).” <http://www.slideshare.net/BodenRussell/>. Accessed October 2015.
- [4] L. Sweeney, “Simple Demographics Often Identify People Uniquely,” *Carnegie Mellon University, Pittsburg, Working Paper 3*, 2000.
- [5] S. Rusitschka and A. Ramirez, “Big Data Technologies and Infrastructures.” <http://byte-project.eu/research/>, Sept. 2014. Deliverable D1.4, Version 1.1.
- [6] P. Mell and T. Grance, “The NIST Definition of Cloud Computing.” <http://www.csrc.nist.gov/groups/SNS/cloud-computing/>, July 2009.
- [7] M. Hogan, F. Liu, and A. Sokol, “Nist cloud computing standards roadmap,” 2011.
- [8] E. U. 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,” 1995.
- [9] U. States., “Health insurance portability and accountability act of 1996 [microform] : conference report (to accompany h.r. 3103).” <http://nla.gov.au/nla.cat-vn4117366>, 1996.
- [10] R.-M. Åhlfeldt, “Information security in distributed healthcare: Exploring the needs for achieving patient safety and patient privacy,” 2008.
- [11] S. Pearson, “Privacy, security and trust in cloud computing,” in *Privacy and Security for Cloud Computing* (S. Pearson and G. Yee, eds.), Computer Communications and Networks, pp. 3–42, Springer London, 2013.

- [12] A. Cavoukian, “The Security-Privacy Paradox: Issues, misconceptions, and Strategies.” <https://www.ipc.on.ca/images/Resources/sec-priv.pdf>, 2003. Accessed November 2015.
- [13] United Nations, “The Universal Declaration of Human Rights.” <http://www.un.org/en/documents/udhr/index.shtml>, 1948. Accessed August 2015.
- [14] A. Westin, *Privacy and Freedom*. New York Atheneum, 1967.
- [15] U. States., “Gramm-leach-bliley act.” <http://www.gpo.gov/fdsys/pkg/PLAW-106publ102/pdf/PLAW-106publ102.pdf>, November 1999.
- [16] U. S. F. Law, “Right to financial privacy act of 1978.” <https://epic.org/privacy/rfpa/>, 1978.
- [17] “Telecommunications Act of 1996.” <http://transition.fcc.gov/Reports/tcom1996.pdf>, 1996. No. 104-104, 110 Stat. 56.
- [18] D. Bigo, G. Boulet, C. Bowden, S. Carrera, J. Jeandesboz, and A. Scherrer, “Fighting cyber crime and protecting privacy in the cloud.” European Parliament, Policy Department C: Citizens’ Rights and Constitutional Affairs, <http://www.europarl.europa.eu/committees/en/studiesdownload.html?languageDocument=EN&file=79050>, Oct. 2012.
- [19] S. Stalla-Bourdillon, “Liability exemptions wanted! internet intermediaries’ liability under uk law,” *Journal of International Commercial Law and Technology*, vol. 7, no. 4, 2012.
- [20] “Scalable, secure storage biobank.” <http://www.biobankcloud.eu>. EU FP7 Framework, Grant Agreement No: 317871, Accessed January 2015.
- [21] M. Janitz, ed., *Next-generation genome sequencing: towards personalized medicine*. John Wiley & Sons, 2011.
- [22] R. Weissleder and M. Y. Pittet, “Imaging in the era of molecular oncology,” *Nature*, vol. 452, no. 7187, 2008.
- [23] F. F. Costa, “Big data in biomedicine,” *Drug discovery today*, vol. 19, no. 4, 2014.
- [24] M. Swan, “The quantified self: fundamental disruption in big data science and biological discovery,” *Big Data*, vol. 1, no. 2, 2013.
- [25] K. Shvachko, H. Kuang, S. Radia, and R. Chansler, “The hadoop distributed file system,” in *Mass Storage Systems and Technologies, 2010*, pp. 1–10, May 2010.
- [26] M. Ronström and J. Orelund, “Recovery Principles of MySQL Cluster 5.1,” in *Proc. of VLDB’05*, pp. 1108–1115, VLDB Endowment, 2005.

- [27] K. Hakimzadeh, H. Peiro Sajjad, and J. Dowling, "Scaling hdfs with a strongly consistent relational model for metadata," in *Distributed Applications and Interoperable Systems*, pp. 38–51, 2014.
- [28] S. Niazi, M. Ismail, G. Berthou, and J. Dowling, "Leader election using newsqldb database systems," in *DAIS*, pp. 158–172, 2015.
- [29] I. S. Reed and G. Solomon, "Polynomial Codes Over Certain Finite Fields," *Journal of the Society for Industrial and Applied Mathematics*, vol. 8, no. 2, pp. 300–304, 1960.
- [30] S. Pabinger, A. Dander, M. Fischer, R. Snajder, M. Sperk, M. Efremova, B. Krabichler, M. R. Speicher, J. Zschocke, and Z. Trajanoski, "A survey of tools for variant analysis of next-generation genome sequencing data," *Briefings in Bioinformatics*, vol. 15, pp. 256–278, Mar. 2014.
- [31] M. Bux, J. Brandt, C. Lipka, K. Hakimzadeh, J. Dowling, and U. Leser, "SAASFEE: Scalable Scientific Workflow Execution Engine," in *VLDB Demonstrations Track, forthcoming*, (Hawaii, USA), 2015.
- [32] "VENUS-C FP7 Project, (2010). Grant Agreement No. 261565." <http://www.venus-c.eu>. Accessed January 2015.
- [33] J. Dean and S. Ghemawat, "Mapreduce: A flexible data processing tool," *Commun. ACM*, vol. 53, pp. 72–77, Jan. 2010.
- [34] I. Foster and A. Grimshaw, "OGSA basic execution service version 1.0." <https://www.ogf.org/documents/GFD.108.pdf>.
- [35] S. Brinkkemper, "Method engineering: engineering of information systems development methods and tools," *Information and Software Technology*, vol. 38, no. 4, pp. 275–280, 1996.
- [36] A. Gholami, A.-S. Lind, J. Reiche, J.-E. Litton, A. Edlund, and E. Laure, "Privacy threat modeling for emerging biobankclouds," in *The 5th International Conference on Emerging Ubiquitous Systems and Pervasive Networks (EUSPN-2014)/ The 4th International Conference on Current and Future Trends of Information and Communication Technologies in Healthcare (ICTH 2014)/ Affiliated Workshops, September 22-25, 2014, Halifax, Nova Scotia, Canada*, vol. 37 of *Procedia Computer Science*, pp. 489–496, Elsevier, 2014.
- [37] A. Gholami and E. Laure, "Advanced cloud privacy threat modeling," *Jan Zizka et al. (Eds) : CCSIT, SIPP, AISC, CMCA, SEAS, CSITEC, DaKM, PDCTA, NeCoM*, p. 229–239, 2016.
- [38] A. Gholami, A.-S. Lind, J. Reiche, J.-E. Litton, A. Edlund, and E. Laure, "Design and implementation of the advanced cloud privacy threat modeling," *International Journal of Network Security & Its Applications (IJNSA)*, 2016.

- [39] A. Bessani, J. Brandt, M. Bux, V. Cogo, L. Dimitrova, J. Dowling, A. Gholami, K. Hakimzadeh, M. Hummel, M. Ismail, *et al.*, “Biobankcloud: a platform for the secure storage, sharing, and processing of large biomedical data sets,” *the First International Workshop on Data Management and Analytics for Medicine and Healthcare (DMAH 2015)*, 2015.
- [40] A. Gholami, J. Dowling, and E. Laure, “A security framework for population-scale genomics analysis,” in *2015 International Conference on High Performance Computing & Simulation, HPCS 2015, Amsterdam, Netherlands, July 20-24, 2015*, pp. 106–114, IEEE, 2015.
- [41] A. Gholami, E. Laure, P. Somogyi, O. Spjuth, NiaziSalman, and J. Dowling, “Privacy-preservation for publishing sample availability data with personal identifiers,” *Journal of Medical and Bioengineering*, vol. 4-2, pp. 117–125, April 2014.
- [42] A. Gholami, G. Svensson, E. Laure, M. Eickhoff, and G. Brasche, “Scabia: Scalable brain image analysis in the cloud,” in *CLOSER 2013 - Proceedings of the 3rd International Conference on Cloud Computing and Services Science, Aachen, Germany, 8-10 May, 2013* (F. Desprez, D. Ferguson, E. Hadar, F. Leymann, M. Jarke, and M. Helfert, eds.), pp. 329–336, SciTePress, 2013.
- [43] A. Szalay and J. Gray, “2020 Computing: Science in an exponential world,” *Nature*, vol. 440, pp. 413–414, Mar. 2006.
- [44] C. Lynch, “Big data: How do your data grow?,” *Nature*, vol. 455, pp. 28–29, Sept. 2008.
- [45] S. Ghemawat, H. Gobioff, and S. Leung, “The google file system,” in *Proceedings of the 19th ACM Symposium on Operating Systems Principles 2003, SOSP 2003, Bolton Landing, NY, USA, October 19-22, 2003*, pp. 29–43, 2003.
- [46] A. S. Foundation, “Apache hadoop 2.7.1 documentation,” 2015. Accessed July 2015.
- [47] T. White, *Hadoop: The Definitive Guide*. O’Reilly Media, Inc., 1st ed., 2009.
- [48] IBM, “Big data and analytics hub.” <http://www.ibmbigdatahub.com>. Accessed July 2015.
- [49] Oracle, “Big data in the cloud.” <https://www.oracle.com/big-data/index.html>, 2015. Accessed July 2015.
- [50] Microsoft, “Data and insights.” <http://www.microsoft.com/enterprise/it-trends/big-data/>, 2015. Accessed July 2015.

- [51] Hewlett-Packard, “Big data solutions.” <http://www8.hp.com/us/en/business-solutions/big-data.html>, 2015. Accessed July 2015.
- [52] CISCO, “Big data.” <http://www.cisco.com/c/en/us/solutions/data-center-virtualization/big-data/index.html>, 2015. Accessed July 2015.
- [53] SAP, “Sap makes big data real.” <http://go.sap.com/docs/download/2014/04/2e777923-0a7c-0010-82c7-eda71af511fa.pdf>, 2015. Accessed July 2015.
- [54] N. B. Data, “Nist big data group.” <http://bigdatawg.nist.gov/home.php>, 2015. Accessed July 2015.
- [55] S. E. Madnick and J. J. Donovan, “Application and analysis of the virtual machine approach to information system security and isolation,” in *Proceedings of the Workshop on Virtual Computer Systems*, (New York, NY, USA), pp. 210–224, ACM, 1973.
- [56] “Timeline of virtualization development.” https://en.wikipedia.org/wiki/Timeline_of_virtualization_development. Accessed June 2015.
- [57] G. J. Popek and R. P. Goldberg, “Formal requirements for virtualizable third generation architectures,” *Commun. ACM*, vol. 17, pp. 412–421, July 1974.
- [58] “Hypervisors, virtualization, and the cloud: Learn about hypervisors, system virtualization, and how it works in a cloud environment.” <http://www.ibm.com/developerworks/cloud/library/cl-hypervisorcompare/>. Accessed June 2015.
- [59] M. Portnoy, *Virtualization Essentials*. Alameda, CA, USA: SYBEX Inc., 1st ed., 2012.
- [60] AMD, “Secure virtual machine architecture reference manual,” tech. rep., Advanced Micro Devices, May 2005.
- [61] “Intel® Virtualization Technology Specification for the Intel® Itanium® Architecture (VT-i),” April 2005. Accessed October 2015.
- [62] R. Dua, A. Raja, and D. Kakadia, “Virtualization vs containerization to support paas,” in *Cloud Engineering (IC2E), 2014 IEEE International Conference on*, pp. 610–614, March 2014.
- [63] D. Bernstein, “Containers and cloud: From lxc to docker to kubernetes,” *Cloud Computing, IEEE*, vol. 1, pp. 81–84, Sept 2014.
- [64] “FreeBSD Jails.” <https://wiki.freebsd.org/Jails>. Accessed September 2015.

- [65] R. Pike, D. Presotto, K. Thompson, H. Trickey, and P. Winterbottom, “The use of name spaces in plan 9,” *SIGOPS Oper. Syst. Rev.*, vol. 27, pp. 72–76, Apr. 1993.
- [66] D. M’Raihi, M. Bellare, F. Hoornaert, D. Naccache, and O. Ranen, “HOTP: An HMAC-Based One-Time Password Algorithm.” RFC 4226 (Informational), Dec. 2005.
- [67] D. M’Raihi, S. Machani, M. Pei, and J. Rydell, “TOTP: Time-Based One-Time Password Algorithm.” RFC 6238 (Informational), May 2011.
- [68] L. Sweeney, “k-anonymity: a model for protecting privacy,” *Int. J. Uncertain. Fuzziness Knowl.-Based Syst.*, vol. 10, pp. 557–570, Oct. 2002.
- [69] G. on Statistical Disclosure Control, “Quasi Identifier.” <http://stats.oecd.org/glossary/detail.asp?ID=6961>. Accessed January 2013.
- [70] L. Sweeney, “Achieving k-anonymity privacy protection using generalization and suppression,” *Int. J. Uncertain. Fuzziness Knowl.-Based Syst.*, vol. 10, pp. 571–588, Oct. 2002.
- [71] P. Samarati and L. Sweeney, “Generalizing data to provide anonymity when disclosing information (abstract),” in *Proceedings of the Seventeenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, PODS ’98, (New York, NY, USA), pp. 188–, ACM, 1998.
- [72] A. Machanavajjhala, D. Kifer, J. Gehrke, and M. Venkatasubramanian, “L-diversity: Privacy beyond k-anonymity,” *ACM Trans. Knowl. Discov. Data*, vol. 1, Mar. 2007.
- [73] C. L. Liu, *Introduction to combinatorial mathematics*. New York, St Louis, San Francisco: McGraw-Hill, 1968.
- [74] A. Shamir, “How to share a secret,” *Commun. ACM*, vol. 22, pp. 612–613, Nov. 1979.
- [75] J. Cappos and S. Torres, “PolyPasswordHasher: Protecting Passwords In The Event Of A Password File Disclosure.” <https://github.com/PolyPasswordHasher/>. Accessed May 2015.
- [76] M. Himmel and F. Grossman, “Security on distributed systems: Cloud security versus traditional it,” *IBM Journal of Research and Development*, vol. 58, pp. 3:1–3:13, Jan 2014.
- [77] M. Zhou, R. Zhang, W. Xie, W. Qian, and A. Zhou, “Security and privacy in cloud computing: A survey,” in *Semantics Knowledge and Grid (SKG), 2010 Sixth International Conference on*, pp. 105–112, Nov 2010.

- [78] L. Kaufman, "Data security in the world of cloud computing," *Security Privacy, IEEE*, vol. 7, pp. 61–64, July 2009.
- [79] D. Chen and H. Zhao, "Data security and privacy protection issues in cloud computing," in *Computer Science and Electronics Engineering (ICCSEE), 2012 International Conference on*, vol. 1, pp. 647–651, March 2012.
- [80] M. Shankarwar and A. Pawar, "Security and privacy in cloud computing: A survey," in *Proceedings of the 3rd International Conference on Frontiers of Intelligent Computing: Theory and Applications (FICTA) 2014* (S. C. Satapathy, B. N. Biswal, S. K. Udgata, and J. K. Mandal, eds.), vol. 328 of *Advances in Intelligent Systems and Computing*, pp. 1–11, Springer International Publishing, 2015.
- [81] H. Li, X. Tian, W. Wei, and C. Sun, "A deep understanding of cloud computing security," in *Network Computing and Information Security* (J. Lei, F. Wang, M. Li, and Y. Luo, eds.), vol. 345 of *Communications in Computer and Information Science*, pp. 98–105, Springer Berlin Heidelberg, 2012.
- [82] G.-J. Ahn and D. Shin, "Towards scalable authentication in health services," in *Enabling Technologies: Infrastructure for Collaborative Enterprises, 2002. WET ICE 2002. Proceedings. Eleventh IEEE International Workshops on*, pp. 83–88, 2002.
- [83] R. D. Pietro, G. Me, and M. A. Strangio, "A two-factor mobile authentication scheme for secure financial transactions," in *2005 International Conference on Mobile Business (ICMB 2005), 11-13 July 2005, Sydney, Australia*, pp. 28–34, 2005.
- [84] D. W. Chadwick, D. P. Mundy, and J. P. New, "Experiences of using a PKI to access a hospital information system by high street opticians," *Computer Communications*, vol. 26, no. 16, pp. 1893–1903, 2003.
- [85] H. Gomes, J. a. P. Cunha, and A. Zúquete, "Authentication architecture for ehealth professionals," in *Proceedings of the 2007 OTM Confederated International Conference on On the Move to Meaningful Internet Systems: CoopIS, DOA, ODBASE, GADA, and IS - Volume Part II, OTM'07*, (Berlin, Heidelberg), pp. 1583–1600, Springer-Verlag, 2007.
- [86] H. Kim and S. Timm, "X.509 authentication and authorization in fermi cloud," in *Utility and Cloud Computing (UCC), 2014 IEEE/ACM 7th International Conference on*, pp. 732–737, Dec 2014.
- [87] M. J. Atallah, K. B. Frikken, M. T. Goodrich, and R. Tamassia, "Secure biometric authentication for weak computational devices," in *Proceedings of the 9th International Conference on Financial Cryptography and Data Security, FC'05*, (Berlin, Heidelberg), pp. 357–371, Springer-Verlag, 2005.

- [88] R. Snelick, U. Uludag, A. Mink, M. Indovina, and A. Jain, “Large-scale evaluation of multimodal biometric authentication using state-of-the-art systems,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 27, pp. 450–455, March 2005.
- [89] N. Mimura Gonzalez, M. Torrez Rojas, M. Maciel da Silva, F. Redigolo, T. Melo de Brito Carvalho, C. Miers, M. Naslund, and A. Ahmed, “A framework for authentication and authorization credentials in cloud computing,” in *Trust, Security and Privacy in Computing and Communications (TrustCom), 2013 12th IEEE International Conference on*, pp. 509–516, July 2013.
- [90] R. Banyal, P. Jain, and V. Jain, “Multi-factor authentication framework for cloud computing,” in *Computational Intelligence, Modelling and Simulation (CIMSIm), 2013 Fifth International Conference on*, pp. 105–110, Sept 2013.
- [91] R. Lomotey and R. Deters, “Saas authentication middleware for mobile consumers of iaas cloud,” in *Services (SERVICES), 2013 IEEE Ninth World Congress on*, pp. 448–455, June 2013.
- [92] B. Tang, R. Sandhu, and Q. Li, “Multi-tenancy authorization models for collaborative cloud services,” in *Collaboration Technologies and Systems (CTS), 2013 International Conference on*, pp. 132–138, May 2013.
- [93] D. F. Ferraiolo and D.R.Kuhn, “Role-Based Access Control,” in *Proc. of the 15th National Computer Security Conference*, pp. 554–563, 1992.
- [94] L. Zhou, V. Varadharajan, and M. Hitchens, “Integrating trust with cryptographic role-based access control for secure cloud data storage,” in *Trust, Security and Privacy in Computing and Communications (TrustCom), 2013 12th IEEE International Conference on*, pp. 560–569, July 2013.
- [95] J. Sendor, Y. Lehmann, G. Serme, and A. Santana de Oliveira, “Platform-level support for authorization in cloud services with oauth 2,” in *Proceedings of the 2014 IEEE International Conference on Cloud Engineering, IC2E '14*, (Washington, DC, USA), pp. 458–465, IEEE Computer Society, 2014.
- [96] M. A. Leandro, T. J. Nascimento, D. R. dos Santos, C. M. Westphall, and C. B. Westphall, “Multi-tenancy authorization system with federated identity for cloud-based environments using shibboleth,” in *Proceedings of the 11th International Conference on Networks, ICN 2012*, pp. 88–93, 2012.
- [97] M. Stihler, A. Santin, A. Marcon, and J. Fraga, “Integral federated identity management for cloud computing,” in *New Technologies, Mobility and Security (NTMS), 2012 5th International Conference on*, pp. 1–5, May 2012.
- [98] H. Li, Y. Dai, L. Tian, and H. Yang, “Identity-based authentication for cloud computing,” in *Cloud Computing* (M. Jaatun, G. Zhao, and C. Rong, eds.),

- vol. 5931 of *Lecture Notes in Computer Science*, pp. 157–166, Springer Berlin Heidelberg, 2009.
- [99] E. Carlini, M. Coppola, P. Dazzi, L. Ricci, and G. Righetti, “Cloud federations in contrail,” in *Euro-Par 2011: Parallel Processing Workshops* (M. Alexander, P. D’Ambra, A. Belloum, G. Bosilca, M. Cannataro, M. Danelutto, B. Di Martino, M. Gerndt, E. Jeannot, R. Namyst, J. Roman, S. Scott, J. Traff, G. Vallée, and J. Weidendorfer, eds.), vol. 7155 of *Lecture Notes in Computer Science*, pp. 159–168, Springer Berlin Heidelberg, 2012.
- [100] J. Gouveia, P. Crocker, S. Melo De Sousa, and R. Azevedo, “E-id authentication and uniform access to cloud storage service providers,” in *Cloud Computing Technology and Science (CloudCom), 2013 IEEE 5th International Conference on*, vol. 1, pp. 487–492, Dec 2013.
- [101] W. Hummer, P. Gaubatz, M. Strembeck, U. Zdun, and S. Dustdar, “Enforcement of Entailment Constraints in Distributed Service-Based Business Processes,” *Information and Software Technology*, 2013.
- [102] G. Dreo, M. Golling, W. Hommel, and F. Tietze, “Iceman: An architecture for secure federated inter-cloud identity management,” in *Integrated Network Management (IM 2013), 2013 IFIP/IEEE International Symposium on*, pp. 1207–1210, May 2013.
- [103] G. Sipos, D. Scardaci, D. Wallom, and Y. Chen, “The user support programme and the training infrastructure of the egi federated cloud,” in *High Performance Computing Simulation (HPCS), 2015 International Conference on*, pp. 9–18, July 2015.
- [104] “Shibboleth support for glassfish.” <https://wiki.shibboleth.net/confluence/display/SHIB2/IdpGlassfishPrepare>. Accessed October 2013.
- [105] N. Santos, K. P. Gummadi, and R. Rodrigues, “Towards trusted cloud computing,” in *Proceedings of the 2009 Conference on Hot Topics in Cloud Computing*, HotCloud’09, (Berkeley, CA, USA), USENIX Association, 2009.
- [106] T. Garfinkel, B. Pfaff, J. Chow, M. Rosenblum, and D. Boneh, “Terra: A virtual machine-based platform for trusted computing,” in *Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles*, SOSP 2003, (New York, NY, USA), pp. 193–206, ACM, 2003.
- [107] R. A. Popa, J. R. Lorch, D. Molnar, H. J. Wang, and L. Zhuang, “Enabling security in cloud storage slas with cloudproof,” in *Proceedings of the 2011 USENIX Conference on USENIX Annual Technical Conference*, USENIX-ATC’11, (Berkeley, CA, USA), pp. 31–31, USENIX Association, 2011.

- [108] S. Zhu and G. Gong, “Fuzzy authorization for cloud storage,” *Cloud Computing, IEEE Transactions on*, vol. 2, pp. 422–435, Oct 2014.
- [109] D. Perez-Botero, J. Szefer, and R. B. Lee, “Characterizing hypervisor vulnerabilities in cloud computing servers,” in *Proceedings of the 2013 International Workshop on Security in Cloud Computing*, Cloud Computing ’13, (New York, NY, USA), pp. 3–10, ACM, 2013.
- [110] F. F. Brasser, M. Bucicoiu, and A.-R. Sadeghi, “Swap and play: Live updating hypervisors and its application to xen,” in *Proceedings of the 6th Edition of the ACM Workshop on Cloud Computing Security*, CCSW ’14, (New York, NY, USA), pp. 33–44, ACM, 2014.
- [111] C. Klein, A. Papadopoulos, M. Dellkrantz, J. Durango, M. Maggio, K.-E. Arzen, F. Hernandez-Rodriguez, and E. Elmroth, “Improving cloud service resilience using brownout-aware load-balancing,” in *Reliable Distributed Systems (SRDS), 2014 IEEE 33rd International Symposium on*, pp. 31–40, Oct 2014.
- [112] E. Lakew, L. Xu, F. Hernandez-Rodriguez, E. Elmroth, and C. Pahl, “A synchronization mechanism for cloud accounting systems,” in *Cloud and Automatic Computing (ICCAC), 2014 International Conference on*, pp. 111–120, Sept 2014.
- [113] “The programming language Lua.” <http://www.lua.org>. Accessed October 2015.
- [114] “Microsoft Silverlight.” <http://www.microsoft.com/silverlight/>. Accessed October 2015.
- [115] P. W. L. Fong, “Reasoning about safety properties in a jvm-like environment,” *Sci. Comput. Program.*, vol. 67, pp. 278–300, July 2007.
- [116] J. G. Politz, A. Guha, and S. Krishnamurthi, “Typed-based verification of Web sandboxes,” *Journal of Computer Security*, vol. 22, no. 4, pp. 511–565, 2014.
- [117] P. H. Phung, D. Sands, and A. Chudnov, “Lightweight self-protecting javascript,” in *Proceedings of the 4th International Symposium on Information, Computer, and Communications Security*, ASIACCS ’09, (New York, NY, USA), pp. 47–60, ACM, 2009.
- [118] “Learn about java technology.” <http://www.java.com/en/about/>. Accessed November 2014.
- [119] N. Paul and D. Evans., “Comparing java and .net security: Lessons learned and missed,” in *Computers and Security*, pp. 338–350, 2006.

- [120] “Linux Container (LXC).” <https://linuxcontainers.org>. Accessed September 2015.
- [121] “Qubes OS.” <http://www.qubes-os.org>. Accessed November 2015.
- [122] X. Chen, T. Garfinkel, E. C. Lewis, P. Subrahmanyam, C. A. Waldspurger, D. Boneh, J. Dwoskin, and D. R. Ports, “Overshadow: A virtualization-based approach to retrofitting protection in commodity operating systems,” *SIGPLAN Not.*, vol. 43, pp. 2–13, Mar. 2008.
- [123] C. Li, A. Raghunathan, and N. Jha, “Secure virtual machine execution under an untrusted management os,” in *Cloud Computing (CLOUD), 2010 IEEE 3rd International Conference on*, pp. 172–179, July 2010.
- [124] J. Yang and K. G. Shin, “Using hypervisor to provide data secrecy for user applications on a per-page basis,” in *Proceedings of the Fourth ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments, VEE '08*, (New York, NY, USA), pp. 71–80, ACM, 2008.
- [125] D. E. Porter, S. Boyd-Wickizer, J. Howell, R. Olinsky, and G. C. Hunt, “Rethinking the library os from the top down,” in *Proceedings of the ASPLOS'11*, (Newport Beach, California, USA), pp. 291–304, 2011.
- [126] A. Baumann, D. Lee, P. Fonseca, L. Glendenning, J. R. Lorch, B. Bond, R. Olinsky, and G. C. Hunt, “Composing os extensions safely and efficiently with bascule,” in *Proceedings of the Eurosys'13*, 2013.
- [127] C. C. Tsai, K. S. Arora, N. Bandi, B. Jain, W. Jannen, J. John, H. A. Kalodner, V. Kulkarni, D. Oliveira, and D. E. Porter, “Cooperation and security isolation of library oses for multi-process applications,” in *Proceedings of the EuroSys'14*, (Amsterdam, Netherlands), 2014.
- [128] A. Baumann, M. Peinado, and G. Hunt, “Shielding applications from an untrusted cloud with haven,” in *Proceedings of the OSDI'14*, 2014.
- [129] T. Garfinkel, B. Pfaff, and M. Rosenblum, “Ostia: A delegating architecture for secure system call interposition,” in *Proceedings of the NDSS'04*, 2004.
- [130] I. Goldberg, D. Wagner, R. Thomas, and E. Brewer, “A secure environment for untrusted helper applications (confining the wily hacker),” in *Proceedings of the USENIX UNIX Security Symposium'96*, 1996.
- [131] D. A. Wagner, “Janus: An approach for confinement of untrusted applications,” in *Tech. Rep. CSD-99-1056, University of California, Berkeley*, 1999.
- [132] T. Garfinkel *et al.*, “Traps and pitfalls: Practical problems in system call interposition based security tools,” in *NDSS*, vol. 3, pp. 163–176, 2003.

- [133] R. Wahbe, S. Lucco, T. E. Anderson, and S. L. Graham, "Efficient software-based fault isolation," in *SIGOPS Oper. Syst. Rev.* 27, 5, pp. 203–216, 1993.
- [134] A. Chou, J. Yang, B. Chelf, S. Hallem, and D. Engler, "An empirical study of operating systems errors," vol. 35, no. 5, 2001.
- [135] C. Small and M. Seltzer, "Misfit: constructing safe extensible systems," *Concurrency, IEEE*, vol. 6, pp. 34–41, Jul 1998.
- [136] M. M. Swift, B. N. Bershad, and H. M. Levy, "Improving the reliability of commodity operating systems," in *Proceedings of the SOSPO'03*, pp. 207–222, 2003.
- [137] B. Yee, D. Sehr, G. Dardyk, J. B. Chen, R. Muth, T. Ormandy, S. Okasaka, N. Narula, and N. Fullagar, "Native client: A sandbox for portable, untrusted x86 native code," in *Proceedings of the IEEE Symposium on Security and Privacy*, (Berkeley, CA, USA), pp. 79–93, 2009.
- [138] M. Anand, "Cloud monitor: Monitoring applications in cloud," in *Cloud Computing in Emerging Markets (CCEM), 2012 IEEE International Conference on*, pp. 1–4, Oct 2012.
- [139] A. Brinkmann, C. Fiehe, A. Litvina, I. Lück, L. Nagel, K. Narayanan, F. Ostermair, and W. Thronicke, "Scalable monitoring system for clouds," in *Proceedings of the 2013 IEEE/ACM 6th International Conference on Utility and Cloud Computing, UCC '13*, (Washington, DC, USA), pp. 351–356, IEEE Computer Society, 2013.
- [140] J. Nikolai and Y. Wang, "Hypervisor-based cloud intrusion detection system," in *Computing, Networking and Communications (ICNC), 2014 International Conference on*, pp. 989–993, Feb 2014.
- [141] C. Basescu, A. Carpen-Amarie, C. Leordeanu, A. Costan, and G. Antoniu, "Managing data access on clouds: A generic framework for enforcing security policies," in *Advanced Information Networking and Applications (AINA), 2011 IEEE International Conference on*, pp. 459–466, March 2011.
- [142] H. Takabi and J. Joshi, "Policy management as a service: An approach to manage policy heterogeneity in cloud computing environment," in *System Science (HICSS), 2012 45th Hawaii International Conference on*, pp. 5500–5508, Jan 2012.
- [143] K. W. Hamlen, L. Kagal, and M. Kantarcioglu, "Policy enforcement framework for cloud data management.," *IEEE Data Eng. Bull.*, vol. 35, no. 4, pp. 39–45, 2012.

- [144] S. Pearson, V. Tountopoulos, D. Catteddu, M. Sudholt, R. Molva, C. Reich, S. Fischer-Hubner, C. Millard, V. Lotz, M. Jaatun, R. Leenes, C. Rong, and J. Lopez, "Accountability for cloud and other future internet services," in *Cloud Computing Technology and Science (CloudCom), 2012 IEEE 4th International Conference on*, pp. 629–632, Dec 2012.
- [145] S. Fischer-Hubner, J. Angulo, and T. Pulls, "How can cloud users be supported in deciding on, tracking and controlling how their data are used?," in *Privacy and Identity Management for Emerging Services and Technologies* (M. Hansen, J.-H. Hoepman, R. Leenes, and D. Whitehouse, eds.), vol. 421 of *IFIP Advances in Information and Communication Technology*, pp. 77–92, Springer Berlin Heidelberg, 2014.
- [146] D. Das, O. O'Malley, S. Radia, and K. Zhang, "Adding Security to Apache Hadoop." Hortonworks Technical Report, 2010.
- [147] "Hortonworks technical report." <http://www.infoq.com/articles/HadoopSecurityModel>. Accessed February 2015.
- [148] A. Gholami, J. Dowling, L. Dimitrova, and R. Merino Martinez, "Security Toolset Implementation (alpha version) of the Scalable, Secure Storage BiobankCloud (D3.3)," tech. rep., Department of High Performance Computing and Visualization, KTH Royal Institute of Technology, Submitted to the European Commission, February 2015. Available at: <http://www.biobankcloud.eu/sites/default/files/deliverables/2014/D3.3-final.pdf>.
- [149] Y. Xianqing, P. Ning, and M. Vouk, "Enhancing security of hadoop in a public cloud," in *Information and Communication Systems (ICICS), 2015 6th International Conference on*, pp. 38–43, April 2015.
- [150] J. Dowling, S. Grohsschmiedt, M. Ismail, and S. Niazi, "Object model implementation the scalable, secure storage biobankcloud (d3.5)," tech. rep., KTH Royal Institute of Technology, Submitted to the European Commission, May 2015. Available at: <http://www.biobankcloud.com/sites/default/files/deliverables/2015/D3.5-final.pdf>.
- [151] E. Bertino, "Big data - security and privacy," in *2015 IEEE International Congress on Big Data, New York City, NY, USA, June 27 - July 2, 2015*, pp. 757–761, 2015.
- [152] S. Sharma, U. S. Tim, J. S. Wong, S. K. Gadia, and S. Sharma, "A brief review on leading big data models," *Data Science Journal*, vol. 13, pp. 138–157, 2014.
- [153] S. Sharma, U. S. Tim, S. K. Gadia, J. S. Wong, R. Shandilya, and S. K. Peddoju, "Classification and comparison of nosql big data models," *IJBDE*, vol. 2, no. 3, pp. 201–221, 2015.

- [154] E. S. Dove, Y. Joly, A.-M. Tassé, P. Burton, R. Chisholm, I. Fortier, P. Goodwin, J. Harris, K. Hveem, J. Kaye, *et al.*, “Genomic cloud computing: legal and ethical points to consider,” *European Journal of Human Genetics*, 2014.
- [155] E. Ayday, J. Raisaro, U. Hengartner, A. Molyneaux, and J.-P. Hubaux, “Privacy-preserving processing of raw genomic data,” in *Data Privacy Management and Autonomous Spontaneous Security* (J. Garcia-Alfaro, G. Lioudakis, N. Cuppens-Boualahia, S. Foley, and W. M. Fitzgerald, eds.), vol. 8247 of *Lecture Notes in Computer Science*, pp. 133–147, Springer Berlin Heidelberg, 2014.
- [156] E. Ayday, E. D. Cristofaro, J.-P. Hubaux, and G. Tsudik, “The chills and thrills of whole genome sequencing,” *Computer*, vol. 99, no. PrePrints, p. 1, 2013.
- [157] Y. Huang and I. Goldberg, “Outsourced private information retrieval,” in *Proceedings of the 12th ACM Workshop on Workshop on Privacy in the Electronic Society*, WPES ’13, (New York, NY, USA), pp. 119–130, ACM, 2013.
- [158] S. Sharif, J. Taheri, A. Y. Zomaya, and S. Nepal, “Mphc: Preserving privacy for workflow execution in hybrid clouds,” in *Parallel and Distributed Computing, Applications and Technologies (PDCAT), 2013 International Conference on*, pp. 272–280, Dec 2013.
- [159] S. Sharif, J. Taheri, A. Y. Zomaya, and S. Nepal, “Online multiple workflow scheduling under privacy and deadline in hybrid cloud environment,” in *Proceedings of the 2014 IEEE 6th International Conference on Cloud Computing Technology and Science*, CLOUDCOM ’14, (Washington, DC, USA), pp. 455–462, IEEE Computer Society, 2014.
- [160] K. Lauter, A. Lopez-Alt, and M. Naehrig, “Private computation on encrypted genomic data,” Tech. Rep. MSR-TR-2014-93, June 2014.
- [161] S. Nakamoto, “Bitcoin: A Peer-to-Peer Electronic Cash System.” <http://fastbull.dl.sourceforge.net/project/bitcoin/Design%20Paper/bitcoin.pdf/bitcoin.pdf>.
- [162] “Statement on auditing standards (sas) no. 70.” http://sas70.com/sas70_overview.html. Accessed January 2013.
- [163] S. D. Gantz and D. R. Philpott, *FISMA and the Risk Management Framework: The New Practice of Federal Cyber Security*. Syngress Publishing, 1st ed., 2012.
- [164] A. Pfitzmann and M. Hansen, “Anonymity, unobservability, and pseudonymity: A consolidated proposal for terminology,” July 2000.

- [165] M. Gostev, J. Fernandez-Banet, J. Rung, J. Dietrich, I. Prokopenko, S. Ripatti, M. I. McCarthy, A. Brazma, and M. Krestyaninova, "Sail - a software system for sample and phenotype availability across biobanks and cohorts.," *Bioinformatics*, vol. 27, no. 4, pp. 589–591, 2011.
- [166] M. Gymrek, A. L. McGuire, D. Golan, E. Halperin, and Y. Erlich, "Identifying personal genomes by surname inference.," *Science (New York, N.Y.)*, vol. 339, pp. 321–324, Jan. 2013.
- [167] A. Pearlgood, "The impact of mandatory data infringement reporting," *Computer Fraud & Security*, vol. 2012, pp. 11–13, May 2012.
- [168] K. Benitez and B. Malin, "Evaluating re-identification risks with respect to the hipaa privacy rule," *JAMIA*, vol. 17, no. 2, pp. 169–177, 2010.
- [169] J. Qian and N. Qamar, "An experimental evaluation of de-identification tools for electronic health records," *CoRR*, vol. abs/1211.3836, 2012.
- [170] A. J. Hundepool and L. C. R. J. Willenborg, "Mu-and tau-argus: Software for statistical disclosure control," 1996.
- [171] M. Templ, "Statistical disclosure control for microdata using the r-package sdcmicro," *Trans. Data Privacy*, vol. 1, pp. 67–85, Aug. 2008.
- [172] Data and Privacy Lab, The University of Texas at Dallas, "Manual for anonymization toolbox." <http://cs.utdallas.edu/dspl/cgi-bin/toolbox/anonManual.pdf>.
- [173] K. LeFevre, D. J. DeWitt, and R. Ramakrishnan, "Incognito: efficient full-domain k-anonymity," in *Proceedings of the 2005 ACM SIGMOD international conference on Management of data*, SIGMOD '05, (New York, NY, USA), pp. 49–60, ACM, 2005.
- [174] J. Litton, J. Muilu, A. Björklund, A. Leinonen, and N. Pedersen, "Data modeling and data communication in genomeutwin.," *Twin Res*, vol. 6, no. 5, pp. 383–90, 2003.
- [175] J. Muilu, L. Peltonen, and J. Litton, "The federated database—a basis for biobank-based post-genome studies, integrating phenome and genome data from 600,000 twin pairs in Europe.," *Eur J Hum Genet*, vol. 15, no. 7, pp. 718–723, 2007.
- [176] G. Ölund, P. Lindqvist, and J.-E. Litton, "Bims: An information management system for biobanking in the 21st century," *IBM Syst. J.*, vol. 46, pp. 171–182, Jan. 2007.
- [177] B. Riedl, V. Grascher, and T. Neubauer, "A secure e-health architecture based on the appliance of pseudonymization," *JSW*, vol. 3, no. 2, pp. 23–32, 2008.

- [178] “CATS.” <https://www.custodix.com/index.php/products>. Accessed December 2015.
- [179] F. Swiderski and W. Snyder, *Threat Modeling*. Redmond, WA, USA: Microsoft Press, 2004.
- [180] J. Ralyté, R. Deneckère, and C. Rolland, “Towards a generic model for situational method engineering,” in *Advanced Information Systems Engineering* (J. Eder and M. Missikoff, eds.), vol. 2681 of *Lecture Notes in Computer Science*, pp. 95–110, Springer Berlin Heidelberg, 2003.
- [181] V. Rahimian and R. Ramsin, “Designing an agile methodology for mobile software development: A hybrid method engineering approach,” in *Research Challenges in Information Science, 2008. RCIS 2008. Second International Conference on*, pp. 337–342, June 2008.
- [182] “The Finnish biobanks.” <http://www.biopankki.fi/en/suomalaiset-biopankit/>. Accessed January 2015.
- [183] “Biobanks, Ethics and Regulations.” <http://www.biobanks.se/ethics.html>. Accessed January 2015.
- [184] C. J. Alberts and A. Dorofee, *Managing Information Security Risks: The Octave Approach*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2002.
- [185] “Article 29 Working Party, Statement on the implementation of the judgment of the Court of Justice of the European Union of 6 October 2015 in the Maximilian Schrems v Data Protection Commissioner case (C-362-14).” http://ec.europa.eu/justice/data-protection/article-29/index_en.htm. Accessed January 2016.
- [186] A.-S. Lind and J. Reiche, “Administrating Data Protection - or the Fort Knox of the European Composite Administration.” *European Critical Quarterly for Legislation and Law*. 2014.
- [187] “Proposal for a Regulation of the European Parliament and of the Council on the protection of individuals with regard to the processing of personal data and on the free movement of such data (General Data Protection Regulation).” Interinstitutional File: 2012/0011 (COD). Accessed December 2015.
- [188] J. Reiche and A.-S. Lind, “Administrating Data Protection - or the Fort Knox of the European Composite Administration.” in Dorr, Dieter and Weaver, Russell L. (Eds), *Perpectives on Privacy*, de Gruyters publisher. 2014.
- [189] K. M. Shelfer and J. D. Procaccino, “Smart card evolution,” *Commun. ACM*, vol. 45, pp. 83–88, July 2002.

- [190] “Types of Smart Cards.” <http://www.smartcardbasics.com/smart-card-types.html>. Accessed February 2013.
- [191] L. Cranor and S. Garfinkel, *Security and Usability: Designing Secure Systems that People Can Use*. O’Reilly Media, 2005.
- [192] R. J. Anderson, *Security Engineering: A Guide to Building Dependable Distributed Systems*. New York, NY, USA: John Wiley & Sons, Inc., 1st ed., 2001.
- [193] L. O’Gorman, “Comparing passwords, tokens, and biometrics for user authentication,” *Proceedings of the IEEE*, vol. 91, pp. 2021–2040, Dec 2003.
- [194] C. Neuman, T. Yu, S. Hartman, and K. Raeburn, “The Kerberos Network Authentication Service (V5).” RFC 4120 (Proposed Standard), <http://www.ietf.org/rfc/rfc4120.txt>, July 2005. Updated by RFCs 4537, 5021.
- [195] P. J. Leach, K. Jaganathan, L. Zhu, and W. Ingersoll, “The Simple and Protected Generic Security Service Application Program Interface (GSS-API) Negotiation Mechanism.” IETF RFC 4178, <https://rfc-editor.org/rfc/rfc4178.txt>, Oct. 2015.
- [196] E. Baize and D. Pinkas, “The Simple and Protected GSS-API Negotiation Mechanism.” RFC 2478 (Proposed Standard), <http://www.ietf.org/rfc/rfc2478.txt>, December 1998. Obsoleted by RFC 4178.
- [197] J. Myers, “Simple authentication and security layer (sas).” RFC Editor, United States, 1997.
- [198] IETF, “RFC 4510 - lightweight directory access protocol (LDAP): Technical specification road map,” tech. rep., IETF, 2006.
- [199] E. Bertino and K. Takahashi, *Identity Management: Concepts, Technologies, and Systems*. Norwood, MA, USA: Artech House, Inc., 2010.
- [200] “Trust Models Guidelines.” <https://www.oasis-open.org/committees/download.php/6158/sstc-saml-trustmodels-2.0-draft-01.pdf>. Accessed December 2012.
- [201] D. Hardt, “The OAuth 2.0 Authorization Framework.” IETF RFC 6749, Oct. 2015.
- [202] R. Boyd, *Getting started with OAuth 2.0*. Beijing: O’Reilly, 2012.
- [203] S. Cantor, J. Kemp, R. Philpott, and E. Maler, “Assertions and Protocols for the OASIS Security Assertion Markup Language (SAML) V2.0,” tech. rep., Mar. 2005.

- [204] The OASIS technical committee, “XACML: eXtensible Access Control Markup Language.” <http://www.oasisopen.org/committees/xacml/repository/>, 2005.
- [205] “XDAS v2.” <https://collaboration.opengroup.org/projects/security/xdas/>. Accessed December 2015.
- [206] “SOC 1 Report (Service Organization Controls Report.” <http://www.ssaе-16.com/soc-1/>. Accessed June 2014.
- [207] C. Steven Hernandez, *Official (ISC)² Guide to the CISSP CBK, Third Edition*. (ISC)² Press, CRC Press, 2013.
- [208] “Custom JAAS realm.” <http://docs.oracle.com/cd/E19226-01/820-7695/6niugeskh/index.html>. Accessed December 2014.
- [209] “Yubikey Manual, YubiKey-Manual. V3-1.” <http://www.yubico.com/>. Accessed August 2013.
- [210] “Biobankcloud repository.” <https://github.com/biobankcloud/>. Accessed March 2016.
- [211] “Hops repository.” <https://github.com/hopshadoop/hopsworks>. Accessed March 2016.
- [212] “Webscarab getting started.” https://www.owasp.org/index.php/WebScarab_Getting_Started. Accessed October 2015.
- [213] T. Dalenius, “Towards a methodology for statistical disclosure control,” *Statistik Tidskrift*, vol. 15, no. 429-444, pp. 2-1, 1977.
- [214] C. A. Shoniregun, K. Dube, and F. Mtenzi, *Electronic Healthcare Information Security*, vol. 53 of *Advances in Information Security*. Springer, 2010.
- [215] NIST, *Secure Hash Standard*. Washington: National Institute of Standards and Technology, 2002. Federal Information Processing Standard 180-2.
- [216] N. I. of Standards and Technology, “Advanced encryption standard,” *NIST FIPS PUB 197*, 2001.
- [217] A. Savva and Others, “Job submission description language (JSDL) specification, version 1.0.” <http://www.ogf.org/documents/GFD.56.pdf>, Nov. 2005.
- [218] “MATLAB Runtime Compiler.” <http://www.mathworks.com/products/compiler>. Accessed October 2011.
- [219] “Microsoft Windows Azure.” <http://www.microsoft.com/windowsazure>. Accessed July 2012.
- [220] “Microsoft Azure Documentation Center.” <https://azure.microsoft.com/en-us/documentation/>. Accessed August 2015.

- [221] D. Engler, D. Y. Chen, S. Hallem, A. Chou, and B. Chelf, “Bugs as deviant behavior: A general approach to inferring errors in systems code,” *ACM*, vol. 35, no. 5, 2001.
- [222] D. Brumley, J. Newsome, D. Song, H. Wang, and S. Jha, “Towards automatic generation of vulnerability-based signatures,” in *Security and Privacy, 2006 IEEE Symposium on*, pp. 15–pp, IEEE, 2006.
- [223] A. C. Chou, “Static analysis for bug finding in systems software,” 2003.
- [224] T. Garfinkel and M. Rosenblum, “When virtual is harder than real: Security challenges in virtual machine based computing environments,” in *Proceedings of the 10th Conference on Hot Topics in Operating Systems - Volume 10, HOTOS’05*, (Berkeley, CA, USA), pp. 20–20, USENIX Association, 2005.
- [225] Nelson Elhage, “Virtunoid: A KVM Guest: Host privilege escalation exploit.” BlackHat, 2011.
- [226] J. Cappos, A. Dadgar, J. Rasley, J. Samuel, I. Beschastnikh, C. Barsan, A. Krishnamurthy, and T. Anderson, “Retaining sandbox containment despite bugs in privileged memory-safe code,” in *Proceedings of the CCS’10*, 2010.
- [227] Y. Li, C. Matthews, A. Gholami, Y. Zhuang, and J. Cappos, “Lind: A portable lightweight single-process sandbox.” <http://systems.cs.brown.edu/nens/2014>, October 2014. The First New England Networking and Systems (NENS) Day, Boston, MA.
- [228] Y. Li, C. Matthews, A. Gholami, Y. Zhuang, and J. Cappos, “Lind: A portable lightweight single-process sandbox.” <http://catt.poly.edu>, December 2014. Center for Advanced Technology in Telecommunications (CATT) Research Review, Brooklyn, NY.
- [229] “Seattle’s Repy V2 Library.” <https://seattle.poly.edu/wiki/RepyV2API>. Accessed September 2014.
- [230] “Code style guidelines for the seattle project.” <https://seattle.poly.edu/wiki/CodingStyle>. Accessed October 2015.
- [231] “Seattle testbed.” <https://seattle.poly.edu/>. Accessed September 2014.
- [232] “Debian Popularity Contest.” <http://popcon.debian.org/main/index.html>. Accessed December 2014.
- [233] “Trinity, a Linux System call fuzz tester.” <http://codemonkey.org.uk/projects/trinity/>. Accessed November 2014.
- [234] “Linux Test Project.” <https://linux-test-project.github.io/>. Accessed February 2015.

- [235] “Exploit Database.” <https://www.exploit-db.com>. Accessed October 2014.
- [236] “gcov(1) - Linux man page.” <http://linux.die.net/man/1/gcov>. Accessed October 2014.
- [237] “Polypasswordhasher repository.” <https://github.com/PolyPasswordHasher/PolyPasswordHasher-Java>. Accessed March 2016.

Appendix A

BioBankCloud

A.1 Identity and Access Management

Listing A.1: BioBankCloud identity and Access Management backend

```
CREATE TABLE 'bbc_group' (
  'group_name' VARCHAR(20) NOT NULL,
  'group_desc' VARCHAR(200) DEFAULT NULL,
  'gid' INT(11) NOT NULL,
  PRIMARY KEY ('gid')
) ENGINE=ndbcluster;

CREATE TABLE 'users' (
  'uid' INT(11) NOT NULL AUTO_INCREMENT,
  'username' VARCHAR(10) NOT NULL,
  'password' VARCHAR(128) NOT NULL,
  'email' VARCHAR(150) DEFAULT NULL,
  'fname' VARCHAR(30) DEFAULT NULL,
  'lname' VARCHAR(30) DEFAULT NULL,
  'activated' TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE
    CURRENT_TIMESTAMP,
  'title' VARCHAR(10) DEFAULT '-',
  'orcid' VARCHAR(20) DEFAULT '-',
  'false_login' INT(11) NOT NULL DEFAULT '-1',
  'status' INT(11) NOT NULL DEFAULT '-1',
  'isonline' INT(11) NOT NULL DEFAULT '-1',
  'secret' VARCHAR(20) DEFAULT NULL,
  'validation_key' VARCHAR(128) DEFAULT NULL,
  'security_question' VARCHAR(20) DEFAULT NULL,
  'security_answer' VARCHAR(128) DEFAULT NULL,
  'mode' INT(11) NOT NULL DEFAULT '0',
  'password_changed' TIMESTAMP NOT NULL DEFAULT '0000-00-00 00:00:00',
```

```

'notes' VARCHAR(500) DEFAULT '- ',
'mobile' VARCHAR(15) DEFAULT '- ',
'max_num_projects' INT(11) NOT NULL,
PRIMARY KEY ('uid'),
UNIQUE KEY 'username' ('username'),
UNIQUE KEY 'email' ('email')
) ENGINE=ndbcluster AUTO_INCREMENT=10000;

CREATE TABLE 'yubikey' (
'yubidnum' INT(11) NOT NULL AUTO_INCREMENT,
'uid' INT(11) NOT NULL,
'public_id' VARCHAR(40) DEFAULT NULL,
'created' TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
'aes_secret' VARCHAR(100) DEFAULT NULL,
'accessed' TIMESTAMP NULL DEFAULT NULL,
'status' INT(11) DEFAULT '-1',
'counter' INT(11) DEFAULT NULL,
'low' INT(11) DEFAULT NULL,
'high' INT(11) DEFAULT NULL,
'session_use' INT(11) DEFAULT NULL,
'notes' VARCHAR(100) DEFAULT NULL,
PRIMARY KEY ('yubidnum'),
FOREIGN KEY ('uid') REFERENCES 'users' ('uid') ON DELETE CASCADE ON
UPDATE NO ACTION
)ENGINE=ndbcluster;

CREATE TABLE 'address' (
'address_id' INT(11) NOT NULL AUTO_INCREMENT,
'uid' INT(11) NOT NULL,
'address1' VARCHAR(100) DEFAULT '- ',
'address2' VARCHAR(100) DEFAULT '- ',
'address3' VARCHAR(100) DEFAULT '- ',
'city' VARCHAR(40) DEFAULT '- ',
'state' VARCHAR(50) DEFAULT '- ',
'country' VARCHAR(40) DEFAULT '- ',
'postalcode' VARCHAR(10) DEFAULT '- ',
PRIMARY KEY ('address_id'),
FOREIGN KEY ('uid') REFERENCES 'users' ('uid') ON DELETE CASCADE ON
UPDATE NO ACTION
) ENGINE=ndbcluster;

CREATE TABLE 'people_group' (
'uid' INT(11) NOT NULL,
'gid' INT(11) NOT NULL,
PRIMARY KEY ('uid','gid'),
FOREIGN KEY ('gid') REFERENCES 'bbc_group' ('gid') ON DELETE NO ACTION
ON UPDATE NO ACTION,

```



```

FOREIGN KEY ('uid') REFERENCES 'users' ('uid') ON DELETE CASCADE ON
UPDATE NO ACTION
) ENGINE=ndbcluster;

CREATE TABLE 'account_audit' (
  'log_id' BIGINT(20) NOT NULL AUTO_INCREMENT,
  'initiator' INT(11) NOT NULL,
  'target' INT(11) NOT NULL,
  'action' VARCHAR(45) DEFAULT NULL,
  'time' TIMESTAMP NULL DEFAULT NULL,
  'message' VARCHAR(100) DEFAULT NULL,
  'outcome' VARCHAR(45) DEFAULT NULL,
  'ip' VARCHAR(45) DEFAULT NULL,
  'browser' VARCHAR(45) DEFAULT NULL,
  'os' VARCHAR(45) DEFAULT NULL,
  'mac' VARCHAR(45) DEFAULT NULL,
  'email' VARCHAR(254) DEFAULT NULL,
  PRIMARY KEY ('log_id'),
  KEY 'initiator' ('initiator'),
  KEY 'target' ('target'),
  FOREIGN KEY ('initiator') REFERENCES 'users' ('uid') ON DELETE NO ACTION
ON UPDATE NO ACTION,
  FOREIGN KEY ('target') REFERENCES 'users' ('uid') ON DELETE NO ACTION ON
UPDATE NO ACTION
) ENGINE=ndbcluster;

CREATE TABLE 'project' (
  'id' INT(11) NOT NULL AUTO_INCREMENT,
  'inode_pid' INT(11) NOT NULL,
  'inode_name' VARCHAR(255) NOT NULL,
  'projectname' VARCHAR(100) NOT NULL,
  'username' VARCHAR(150) NOT NULL,
  'created' TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
  'retention_period' DATE DEFAULT NULL,
  'ethical_status' VARCHAR(30) DEFAULT NULL,
  'archived' TINYINT(1) DEFAULT '0',
  'deleted' TINYINT(1) DEFAULT '0',
  'description' VARCHAR(2000) DEFAULT NULL,
  PRIMARY KEY ('id'),
  UNIQUE KEY('projectname'),
  UNIQUE KEY('inode_pid', 'inode_name'),
  FOREIGN KEY ('username') REFERENCES 'users' ('email') ON DELETE NO
ACTION ON UPDATE NO ACTION,
  FOREIGN KEY ('inode_pid', 'inode_name') REFERENCES
  'hops', 'hdfs_inodes' ('parent_id', 'name') ON DELETE CASCADE ON UPDATE
NO ACTION
) ENGINE=ndbcluster;

```

```

CREATE TABLE 'activity' (
  'id' INT(11) NOT NULL AUTO_INCREMENT,
  'activity' VARCHAR(128) NOT NULL,
  'user_id' INT(10) NOT NULL,
  'created' TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
  'flag' VARCHAR(128) DEFAULT NULL,
  'project_id' INT(11) NOT NULL,
  PRIMARY KEY ('id'),
  FOREIGN KEY ('project_id') REFERENCES 'project' ('id') ON DELETE CASCADE
    ON UPDATE NO ACTION,
  FOREIGN KEY ('user_id') REFERENCES 'users' ('uid') ON DELETE NO ACTION
    ON UPDATE NO ACTION
) ENGINE=ndbcluster;

```

```

CREATE TABLE 'userlogins' (
  'login_id' BIGINT(20) NOT NULL AUTO_INCREMENT,
  'ip' VARCHAR(45) DEFAULT NULL,
  'os' VARCHAR(30) DEFAULT NULL,
  'browser' VARCHAR(40) DEFAULT NULL,
  'action' VARCHAR(80) DEFAULT NULL,
  'outcome' VARCHAR(20) DEFAULT NULL,
  'mac' VARCHAR(45) DEFAULT NULL,
  'uid' INT(11) NOT NULL,
  'email' VARCHAR(150) DEFAULT NULL,
  'login_date' TIMESTAMP NULL DEFAULT NULL,
  PRIMARY KEY ('login_id'),
  KEY ('login_date'),
  FOREIGN KEY ('uid') REFERENCES 'users' ('uid') ON DELETE CASCADE ON
    UPDATE NO ACTION
) ENGINE=ndbcluster;

```

```

CREATE TABLE 'organization' (
  'id' INT(11) NOT NULL AUTO_INCREMENT,
  'uid' INT(11) DEFAULT NULL,
  'org_name' VARCHAR(100) DEFAULT '-',
  'website' VARCHAR(2083) DEFAULT '-',
  'contact_person' VARCHAR(100) DEFAULT '-',
  'contact_email' VARCHAR(150) DEFAULT '-',
  'department' VARCHAR(100) DEFAULT '-',
  'phone' VARCHAR(20) DEFAULT '-',
  'fax' VARCHAR(20) DEFAULT '-',
  PRIMARY KEY ('id'),
  FOREIGN KEY ('uid') REFERENCES 'users' ('uid') ON DELETE CASCADE
) ENGINE=ndbcluster;

```

```

CREATE TABLE 'consents_audit' (

```

```
'log_id' BIGINT(20) NOT NULL AUTO_INCREMENT,  
'initiator' INT(11) NOT NULL,  
'consent_id' int(11) NOT NULL,  
'action' VARCHAR(45) DEFAULT NULL,  
'time' TIMESTAMP NULL DEFAULT NULL,  
'message' VARCHAR(500) DEFAULT NULL,  
'outcome' VARCHAR(45) DEFAULT NULL,  
'ip' VARCHAR(45) DEFAULT NULL,  
'browser' VARCHAR(45) DEFAULT NULL,  
'os' VARCHAR(45) DEFAULT NULL,  
'mac' VARCHAR(45) DEFAULT NULL,  
PRIMARY KEY ('log_id'),  
KEY 'initiator' ('initiator'),  
FOREIGN KEY ('initiator') REFERENCES 'users' ('uid') ON DELETE NO ACTION  
ON UPDATE NO ACTION,  
FOREIGN KEY ('consent_id') REFERENCES 'consents' ('id') ON DELETE NO  
ACTION ON UPDATE NO ACTION  
) ENGINE=ndbcluster;
```

A.2 Auditing Users Actions

Listing A.2: Mapping of backend relations with the role auditing model inside the security framework

```

package se.kth.bbc.security.audit.model;

@Entity
@Table(name = "hopsworks.roles_audit")
@XmlRootElement
@NamedQueries({
    @NamedQuery(name = "RolesAudit.findAll",
        query = "SELECT r FROM RolesAudit r"),
    @NamedQuery(name = "RolesAudit.findByLogId",
        query = "SELECT r FROM RolesAudit r WHERE r.logId = :logId"),
    @NamedQuery(name = "RolesAudit.findByInitiator",
        query = "SELECT r FROM RolesAudit r WHERE r.initiator =
            :initiator"),
    @NamedQuery(name = "RolesAudit.findByAction",
        query = "SELECT r FROM RolesAudit r WHERE r.action = :action"),
    @NamedQuery(name = "RolesAudit.findByTime",
        query = "SELECT r FROM RolesAudit r WHERE r.time = :time"),
    @NamedQuery(name = "RolesAudit.findByMessage",
        query = "SELECT r FROM RolesAudit r WHERE r.message = :message"),
    @NamedQuery(name = "RolesAudit.findByIp",
        query = "SELECT r FROM RolesAudit r WHERE r.ip = :ip"),
    @NamedQuery(name = "RolesAudit.findByOs",
        query = "SELECT r FROM RolesAudit r WHERE r.os = :os"),
    @NamedQuery(name = "RolesAudit.findByOutcome",
        query = "SELECT r FROM RolesAudit r WHERE r.outcome = :outcome"),
    @NamedQuery(name = "RolesAudit.findByBrowser",
        query = "SELECT r FROM RolesAudit r WHERE r.browser = :browser"),
    @NamedQuery(name = "RolesAudit.findByMac",
        query = "SELECT r FROM RolesAudit r WHERE r.mac = :mac"))})
public class RolesAudit implements Serializable {

    private static final long serialVersionUID = 1L;
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Basic(optional = false)
    @Column(name = "log_id")
    private Long logId;
    @Size(max = 45)
    @Column(name = "action")
    private String action;
    @Column(name = "time")
    @Temporal(TemporalType.TIMESTAMP)

```

```
private Date time;
@Size(max = 45)
@Column(name = "message")
private String message;
@Size(max = 45)
@Column(name = "ip")
private String ip;
@Size(max = 45)
@Column(name = "outcome")
private String outcome;
@Size(max = 45)
@Column(name = "browser")
private String browser;
@Size(max = 45)
@Column(name = "os")
private String os;
@Size(max = 254)
@Column(name = "email")
private String email;
@Size(max = 45)
@Column(name = "mac")
private String mac;
@JoinColumn(name = "target",
            referencedColumnName = "uid")
@ManyToOne
private Users target;

@JoinColumn(name = "initiator",
            referencedColumnName = "uid")
@ManyToOne
private Users initiator;

public RolesAudit() {
}

public RolesAudit(Long logId) {
    this.logId = logId;
}

public Long getLogId() {
    return logId;
}

public void setLogId(Long logId) {
    this.logId = logId;
}
```

```
public Users getInitiator() {
    return initiator;
}

public void setInitiator(Users initiator) {
    this.initiator = initiator;
}

public String getAction() {
    return action;
}

public void setAction(String action) {
    this.action = action;
}

public Date getTime() {
    return time;
}

public void setTime(Date time) {
    this.time = time;
}

public String getMessage() {
    return message;
}

public void setMessage(String message) {
    this.message = message;
}

public String getIp() {
    return ip;
}

public void setIp(String ip) {
    this.ip = ip;
}

public String getOutcome() {
    return outcome;
}

public void setOutcome(String outcome) {
    this.outcome = outcome;
}
```

```
}

public String getBrowser() {
    return browser;
}

public void setBrowser(String browser) {
    this.browser = browser;
}

public String getMac() {
    return mac;
}

public void setMac(String mac) {
    this.mac = mac;
}

public Users getTarget() {
    return target;
}

public void setTarget(Users target) {
    this.target = target;
}

public String getOs() {
    return os;
}

public void setOs(String os) {
    this.os = os;
}

public String getEmail() {
    return email;
}

public void setEmail(String email) {
    this.email = email;
}

@Override
public int hashCode() {
    int hash = 0;
    hash += (logId != null ? logId.hashCode() : 0);
}
```

```
    return hash;
}

@Override
public boolean equals(Object object) {
    if (!(object instanceof RolesAudit)) {
        return false;
    }

    RolesAudit other = (RolesAudit) object;
    if ((this.logId == null && other.logId != null) || (this.logId != null
        && !this.logId.equals(other.logId))) {
        return false;
    }
    return true;
}

@Override
public String toString() {
    return "se.kth.bb.security.audit.model.RolesAudit[ logId=" + logId + "
        ]";
}
}
}
```

Appendix B

eCPC Toolkit

B.1 k-anonymity

Listing B.1: k-anonymization in sdcMicro integration

```
public void anonymizeCSV(String csvFilePath, String keyColumns, int k)
    throws IOException, InterruptedException {

    String cmd0 = "library('sdcMicro')";
    String cmd1 = "library('sdcMicroGUI')";
    String cmd2 = "csvfile <- read.csv(' + csvFilePath + "', sep='\" +
        CSVColumnSeparator + "\", na.strings = '')";
    String cmd3 = "kanonym <- localSuppression(csvfile, keyVars = c(\" +
        keyColumns + \"),\"
        + \" k=\" + k + \", importance=NULL)";
    String cmd4 = "plot(kanonym)";
    String cmd5 = "write.table(kanonym$xAnon, quote = FALSE, row.names
        = FALSE, \"
        + \"sep =\" + CSVColumnSeparator + \", eol = '\\n', file=\" +
        AnonymizedCSVFile + \"')";

    String jointCmd = cmd0 + "\\n\" + cmd1 + "\\n\" + cmd2 + "\\n\" + cmd3 +
        "\\n\" + cmd4 + "\\n\" + cmd5;
    String inFile = ECPCUtils.createBatch(jointCmd);

    String cmd = "R CMD BATCH \" + inFile;

    Process pr = Runtime.getRuntime().exec(cmd);
    pr.waitFor();
}
```

B.2 l-diversirty

Listing B.2: l-Diversity in sdcMicro integration

```

public void checkLDiversity(String csvFilePath, String keyColumns,
String sensitive, int k, int l)
    throws IOException, InterruptedException {
    String cmd0 = "library('sdcMicro')";
    String cmd1 = "library('sdcMicroGUI')";
    String cmd2 = "csvfile <- read.csv('" + csvFilePath + "', "
        + "sep='" + CSVColumnSeperator+"', na.strings = '')";
    String cmd3 = "kanonym <- localSuppression(csvfile, keyVars =
        c(" + keyColumns+ "), "
        + " k=" + k + ", importance=NULL)";
    String ldivCmd= "ldiv<-
        ldiversity(kanonym$xAnon,keyVars=c("+keyColumns+"),"
        + "ldiv_index=c("+ sensitive+"))";
    String ldivPlot= "print(ldiv) \n plot(ldiv)";

    String ldivCmd1= "TFfk <- apply(ldiv,1,function(x)any(x<"+l+"))";

    String ldivCmd2= "print(TFfk)";
    String jointCmd = cmd0 + "\n" + cmd1 + "\n" + cmd2 + "\n" + cmd3
        + "\n" +
        ldivCmd+ "\n" +ldivPlot + "\n"+ ldivCmd1+ "\n"+ ldivCmd2;

    String inFile = ECPCUtils.createBatch(jointCmd);

    String cmd = "R CMD BATCH " + inFile;

    Process pr = Runtime.getRuntime().exec(cmd);
    pr.waitFor();
}

```

B.3 Reidentification Risk

Listing B.3: Measuring the reidentification risk using sdcMicro

```
public void calcReidentificationRisk(String csvFile, String sensitive)
    throws IOException, InterruptedException {

    CSVUtils.cleanup();
    String libCmd = "library('sdcMicroGUI')";
    String cmd0 = "library('sdcMicro')";
    String cmd1 = "csvfile <- read.csv('" + csvFile + "', sep='\" +
        CSVColumnSeparator + "\", na.strings='')";
    String cmd2 = "freq <- freqCalc( csvfile, keyVars= c(" + sensitive
        + "))";
    String cmd3 = "indRisk <- indivRisk(freq, survey=TRUE)";
    String cmd4 = "plot(indRisk)";
    String jointCmd = libCmd + "\\n" + cmd0 + "\\n" + cmd1 + "\\n" + cmd2
        + "\\n"
        + cmd3 + "\\n" + cmd4;

    String inFile = ECPCUtils.createBatch(jointCmd);

    String cmd = "R CMD BATCH " + inFile;

    Process pr = Runtime.getRuntime().exec(cmd);
    pr.waitFor();

}
```

Appendix C

Lind Dual Sandbox

C.1 Porting Applications in NaCl and Repy

Listing C.1: Porting popular open source packages in NaCl to be used in Lind

```
#!/bin/bash

export NAME=Grep
export VERSION=2.14
export NACL_ARCH=x86_64
export TOOLCHAIN=glibc
export NACLPORTS_PREFIX=${REPY_PATH}/usr
export NACLPORTS_PREFIX_BIN=${REPY_PATH}/bin
source /usr/lind_project/lind/naclports/src/build_tools/common.sh

export NACL_SEL_LDR
export NACL_IRT
export NACL_SDK_LIB
export CC=${NACLCC}
export CXX=${NACL_CXX}
export AR=${NACLAR}
export RANLIB=${NACL_RANLIB}
export PKG_CONFIG_PATH=${NACLPORTS_LIBDIR}/pkgconfig
export PKG_CONFIG_LIBDIR=${NACLPORTS_LIBDIR}
export FREETYPE_CONFIG=${NACLPORTS_PREFIX_BIN}/freetype-config
export PATH=${NACL_BIN_PATH}:${PATH};

export NETTLE_CFLAGS=${NACLPORTS_PREFIX}/include
export GMP_CFLAGS=${NACLPORTS_PREFIX}/include

conf_host=${NACL_CROSS_PREFIX}
```

```
Banner "Configure grep-2.14"
ChangeDir "grep-2.14"
./configure\
  --prefix=${NACLPORTS_PREFIX} --enable-shared --host=x86_64-linux
make clean
make
```

C.2 Lind's Parser for Gcov

Listing C.2: Gcov parser to collect the data usage in the target platforms

```
import sys, os, string, getopt

def main(argv):
    inputfile = ''
    outputfile = ''
    try:
        opts, args = getopt.getopt(argv,"hi:o:",["ifile=", "ofile="])
    except getopt.GetoptError:
        print 'analyzer.py -i <coverage.info> -o <outputfile>'
        sys.exit(2)

    for opt, arg in opts:
        if opt == '-h':
            print 'analyzer.py -i <coverage.info> -o <outputfile>'
            sys.exit()
        elif opt in ("-i", "--ifile"):
            inputfile = arg
        elif opt in ("-o", "--ofile"):
            outputfile = arg

    if os.path.isfile(inputfile):
        infile = open(inputfile, 'r')
        outfile = open(outputfile, 'w')
        kernel_usage(infile, outfile)
        infile.close()
        outfile.close()
    else:
        print "File: ", inputfile, " does not exist!"
        sys.exit(2)

"""
This function generates the lines that are used used by each source file in
the kernel.
"""

def kernel_usage(infile, outfile):

    """
    A set of lines used in the SF. Each SF might be run several times
    and therefore we need a way to make each run identical:
    SF:/usr/src/linux-3.13.0/arch/x86/include/asm/atomic.h
    DA:38,3
    """
```

```

DA:117,3
LF:2
LH:2
SF:/usr/src/linux-3.13.0/arch/x86/include/asm/atomic.h:
DA:38,3
DA:40:0
DA 117,30
LF:2
LH:2
Result will be as follow which duplicated lines are removed:
SF:/usr/src/linux-3.13.0/arch/x86/include/asm/atomic.h
DA:38
DA:117
"""
da_values = set()

"Create a unique mapping of SFs with the executed lines-"
usage_dict = {}

"Put the SF path in a tmp variable to test if it already exist not to
be duplicated."
tmp_path= ""

"Generate the executed lines for each SF and write the unique results
in an outputfile."
for lines in infile:
    if ("SF:" in lines):
        tmp_path = lines.rstrip(os.linesep)

    if ("DA:" in lines) and ("FNDA" not in lines):
        lines = lines.replace("DA:", "")
        line = lines.split(",")

        if ("====" not in line[1]):
            if(int(line[1]) > 0):
                da_values.add(int(line[0]))

    if ("end_of_record" in lines):
        tmp_set = usage_dict.get(tmp_path)
        if tmp_set is not None:
            union = set().union(da_values,tmp_set)
            usage_dict[tmp_path] = union.copy()
        else:
            usage_dict[tmp_path]= da_values.copy()
            da_values.clear()

for sf,da in usage_dict.iteritems():

```



```
if(da):
    outfile.write(sf+'\n')
    da_sorted = sorted(da)
    for lines in da_sorted:
        outfile.write('%d\n'%lines)

    outfile.write('\n')
```

pass

Appendix D

Lind Reference Monitor

D.1 Policy Definition in Lind

Listing D.1: Lind's reference monitor policy definition

```
// Define the syscalls that are file system related
access=ALLOW_LIND
close=ALLOW_LIND
chdir=ALLOW_LIND
chmod=ALLOW_LIND
chown=ALLOW_LIND
creat=ALLOW_LIND
// dup* is not only for FS calls...
dup=ALLOW_LIND
dup2=ALLOW_LIND
dup3=ALLOW_LIND
fstat=ALLOW_LIND
fstatfs=ALLOW_LIND
//Lots of uses for fcntl...
fcntl=ALLOW_LIND
flock=ALLOW_LIND
stat=ALLOW_LIND
statfs=ALLOW_LIND
open=ALLOW_LIND
openat=ALLOW_LIND
read=ALLOW_LIND
mkdir=ALLOW_LIND
rmdir=ALLOW_LIND
rename=ALLOW_LIND
link=ALLOW_LIND
lseek=ALLOW_LIND
ioctl=ALLOW_LIND
```

```
pread64=ALLOW_LIND
pwritev=ALLOW_LIND
getdents=ALLOW_LIND
unlink=ALLOW_LIND
write=ALLOW_LIND

// Define network related system calls
accept=ALLOW_LIND
bind=ALLOW_LIND
connect=ALLOW_LIND
getpeername=ALLOW_LIND
getsockname=ALLOW_LIND
getsockopt=ALLOW_LIND
listen=ALLOW_LIND
setsockopt=ALLOW_LIND
socketpair=ALLOW_LIND
sendto=ALLOW_LIND
recvfrom=ALLOW_LIND
sendmsg=ALLOW_LIND
recvmsg=ALLOW_LIND
socket=ALLOW_LIND
shutdown=ALLOW_LIND

// Define generic user ID - gid syscalls
getegid=ALLOW_LIND
geteuid=ALLOW_LIND
getgid=ALLOW_LIND
getuid=ALLOW_LIND
getgroups=ALLOW_LIND
getpid=ALLOW_LIND

// Concurrency related
epoll_create=ALLOW_LIND
epoll_ctl=ALLOW_LIND
epoll_wait=ALLOW_LIND
epoll_create1=ALLOW_LIND
select=ALLOW_LIND
poll=ALLOW_LIND

// Define the syscalls that should go through OS
munmap=ALLOW_OS
mmap=ALLOW_OS
brk=ALLOW_OS
setgroups=ALLOW_OS
arch_prctl=ALLOW_OS
mprotect=ALLOW_OS
```

```
// TEMP CHANGES TO DEBUG APACHE
execve=ALLOW_OS
set_tid_address=ALLOW_OS
futex=ALLOW_OS
set_robust_list=ALLOW_OS
getrlimit=ALLOW_OS
rt_sigaction=ALLOW_OS
rt_sigprocmask=ALLOW_OS
exit_group=ALLOW_OS
exit=ALLOW_OS
clone=ALLOW_OS
kill=ALLOW_OS

// Calls that are not yet implemented
lstat=DENY_LIND
accept4=DENY_LIND
acct=DENY_LIND
...
```

D.2 System Call Filtering in Lind

Listing D.2: Lind's reference monitor system call filtering implementation

```

void monitor_ns()
{
    if (entering) {
        entering = 0;
    } else {
        regs.retval = EINVAL;
        fprintf(stdout, "%s () not supported by lind\n",
            syscall_names[regs.syscall]);

        set_args(&regs);
        entering = 1;
    }
}

void monitor_execve()
{
    if (entering) {
        entering = 0;

        char *execve_path;
        char **execve_args;

        execve_path = get_path(regs.arg1);
        char** p;
        int i = 0, argc;
        while (1) {
            p = (char**) get_mem(regs.arg2 + i * sizeof(char*), sizeof(char*));
            ++i;
            if (!*p)
                break;
        }
        argc = i;
        execve_args = malloc(sizeof(char*) * argc);
        execve_args[argc - 1] = 0;
        i = 0;
        fprintf(stdout, "[monitor] execve(%s, [", execve_path);
        for (i = 0; i < argc; ++i) {
            p = get_mem(regs.arg2 + i * sizeof(char*), sizeof(char*));
            if (*p) {
                execve_args[i] = (char *) get_path((uintptr_t)*p);
                fprintf(stdout, "%s ", execve_args[i]);
            }
        }
    }
}

```

```

    }
} else {
    fprintf(stdout, "]);
    fprintf(stdout, ") = %d\n", (int) regs.retval);
    entering = 1;
}
}

void monitor_deny()
{
    if (entering) {
        regs.retval = EINVAL;
        set_args(&regs);
        entering = 0;
        fprintf(stdout, "[monitor] BEFORE Deny call by Lind: %s() = %d\n",
                syscall_names[regs.syscall], (int) regs.retval);
        fprintf(stdout, "Aborting\n");
        exit(1);
    } else {
        fprintf(stdout, "[monitor] AFTER Deny call by Lind: %s() = %d\n",
                syscall_names[regs.syscall], (int) regs.retval);
        entering = 1;
    }
}

void monitor_close()
{
    if (entering) {
        entering = 0;
    } else {
        if ((int32_t) regs.arg1 >= 0) {
            regs.retval = lind_close(regs.arg1);
            set_args(&regs);
            fprintf(stdout, "[monitor] close(%d) = %d\n", (int) regs.arg1,
                    (int) regs.retval);
        }
        entering = 1;
    }
}

void monitor_getuid()
{
    if (entering) {
        entering = 0;
    } else {
        regs.retval = lind_getuid();
    }
}

```

```

        set_args(&regs);
        fprintf(stdout, "[monitor] getuid() = %d\n", (int) regs.retval);
        entering = 1;
    }
}

void monitor_read()
{
    if (entering) {
        entering = 0;
    } else {
        void *buff = malloc(regs.arg3);
        regs.retval = lind_read(regs.arg1, buff, regs.arg3);
        set_mem(regs.arg2, buff, regs.arg3);
        set_args(&regs);
        fprintf(stdout, "[monitor] read(%d, 0x%lx[], %d) = %d\n", (int)
            regs.arg1,
            (long) regs.arg2, (int) regs.arg3, (int) regs.retval);
        entering = 1;
    }
}

void monitor_open()
{
    if (entering) {
        entering = 0;
    } else {
        char *path = get_path(regs.arg1);
        int lind_fd = lind_open(path, regs.arg2, regs.arg3);

        if (lind_fd >= 0) {
            add_mapping(regs.retval, lind_fd);
            regs.retval = lind_fd;
            set_args(&regs);
            fprintf(stdout, "[monitor] open(%s, %d, %d) = %d\n", path,
                (int) regs.arg2, (int) regs.arg3, (int) regs.retval);
        }
        entering = 1;
    }
}

void monitor_fstat()
{
    if (entering) {
        entering = 0;
    }
}

```



```

} else {
    struct lind_stat st;
    regs.retval = lind_fstat(regs.arg1, &st);
    set_mem(regs.arg2, &st, sizeof(st));
    set_args(&regs);

    fprintf(stdout, "[monitor] fstat(%d, {st_mode = %d, st_size = %d}) =
        %d\n", (int) regs.arg1, (int) st.st_mode, (int) st.st_size,
            (int) regs.retval);
    entering = 1;
}
}

void monitor_openat()
{
    if (entering) {
        entering = 0;
    } else {
        char *path = get_path(regs.arg2);
        int lind_fd = lind_openat(regs.arg1, path, regs.arg3, regs.arg4);

        if (lind_fd >= 0) {
            add_mapping(regs.retval, lind_fd);
            regs.retval = lind_fd;
        } else {
            regs.retval = -1;
        }
        set_args(&regs);
        fprintf(stdout, "[monitor] openat(%d, %s, %d, %d) = %d\n",
            (int) regs.arg1, path, (int) regs.arg2, (int) regs.arg4,
            (int) regs.retval);
        entering = 1;
    }
}

void monitor_access()
{
    if (entering) {
        entering = 0;
    } else {
        char *path = get_path(regs.arg1);
        int lind_fd = lind_access(path, regs.arg2);
        if (lind_fd >= 0) {
            add_mapping(regs.retval, lind_fd);
            regs.retval = lind_fd;
        } else {

```

```

        regs.retval = -1;
    }
    set_args(&regs);
    fprintf(stdout, "[monitor] access(%s, %d, %d) = %d\n", path,
        (int) regs.arg2, (int) regs.arg3, (int) regs.retval);
    entering = 1;
}
}

void monitor_statfs()
{
    struct lind_statfs stfs;

    if (entering) {
        entering = 0;
    } else {
        char *path = get_path(regs.arg1);
        regs.retval = lind_statfs(path, &stfs);
        set_mem(regs.arg2, &stfs, sizeof(stfs));
        set_args(&regs);
        fprintf(stdout, "[monitor] statfs(%s) = %d\n", path,
            (int) regs.retval);
        entering = 1;
    }
}

void monitor_stat()
{
    struct lind_stat st;

    if (entering) {
        entering = 0;
    } else {
        char *path = get_path(regs.arg1);
        regs.retval = lind_stat(path, &st);
        set_mem(regs.arg2, &st, sizeof(st));
        set_args(&regs);
        fprintf(stdout, "[monitor] stat(%s) = %d\n", path, (int) regs.retval);
        entering = 1;
    }
}

void monitor_fstatfs()
{
    struct lind_statfs stfs;

    if (entering) {

```

```
    entering = 0;
} else {
    regs.retval = lind_fstatfs(regs.arg1, &stfs);
    set_mem(regs.arg2, &stfs, sizeof(stfs));
    set_args(&regs);
    fprintf(stdout, "[monitor] fstatfs(%d) = %d\n", (int) regs.arg1,
            (int) regs.retval);
    entering = 1;
}
}

void monitor_rmdir()
{
    if (entering) {
        entering = 0;
    } else {
        char* path = get_path(regs.arg1);
        regs.retval = lind_rmdir(path);
        set_args(&regs);
        fprintf(stdout, "[monitor] rmdir(%s) = %d\n", path, (int)
                regs.retval);
        entering = 1;
    }
}

void monitor_getpid()
{
    if (entering) {
        entering = 0;
    } else {
        regs.retval = lind_getpid();
        set_args(&regs);
        fprintf(stdout, "[monitor] getpid() = %d\n", (int) regs.retval);
        entering = 1;
    }
}

void monitor_geteuid()
{
    if (entering) {
        entering = 0;
    } else {
        regs.retval = lind_geteuid();
        set_args(&regs);
        fprintf(stdout, "[monitor] geteuid() = %d\n", (int) regs.retval);
        entering = 1;
    }
}
```

```
}

void monitor_getgid()
{
    if (entering) {
        entering = 0;
    } else {
        regs.retval = lind_getgid();
        fprintf(stdout, "[monitor] getgid() = %d\n", (int) regs.retval);
        set_args(&regs);
        entering = 1;
    }
}

void monitor_getegid()
{
    if (entering) {
        entering = 0;
    } else {
        regs.retval = lind_getegid();
        set_args(&regs);
        fprintf(stdout, "[monitor] getegid() = %d\n", (int) regs.retval);
        entering = 1;
    }
}

void monitor_write()
{
    if (entering) {
        entering = 0;
    } else {
        regs.retval = lind_write(regs.arg1, get_mem(regs.arg2, regs.arg3),
            regs.arg3);

        set_args(&regs);
        fprintf(stdout, "[monitor] write(%d, 0x%lx[], %d) = %d\n", (int)
            regs.arg1,
            (long) regs.arg2, (int) regs.arg3,
            (int) regs.retval);
        entering = 1;
    }
}

void monitor_unlink()
{
    if (entering) {
```

```
    entering = 0;
} else {
    char *path = get_path(regs.arg1);
    regs.retval = lind_unlink(path);
    set_args(&regs);

    fprintf(stdout, "[monitor] unlink(%s) = %d\n", path,
            (int) regs.retval);

    entering = 1;
}
}

void monitor_link()
{
    if (entering) {
        entering = 0;
    } else {
        char *path = get_path(regs.arg1);
        char *path1 = get_path(regs.arg2);
        regs.retval = lind_link(path, path1);
        set_args(&regs);
        fprintf(stdout, "[monitor] link(%s, %s) = %d\n", path, path1,
                (int) regs.retval);
        entering = 1;
    }
}

void monitor_fcntl()
{
    if (entering) {
        entering = 0;
    } else {
        regs.retval = lind_fcntl(regs.arg1, regs.arg2);
        set_args(&regs);
        fprintf(stdout, "[monitor] fcntl(%d, %d) = %d\n", (int) regs.arg1,
                (int) regs.arg2, (int) regs.retval);
        entering = 1;
    }
}

void monitor_listen()
{
    if (entering) {
        entering = 0;
    } else {
        regs.retval = lind_listen(regs.arg1, regs.arg2);
    }
}
```

```

        set_args(&regs);
        fprintf(stdout, "[monitor] listen(%d, %d) = %d\n", (int) regs.arg1,
                (int) regs.arg2, (int) regs.retval);
        entering = 1;
    }
}

void monitor_shutdown()
{
    if (entering) {
        entering = 0;
    } else {
        regs.retval = lind_shutdown(regs.arg1, regs.arg2);
        set_args(&regs);
        fprintf(stdout, "[monitor] shutdown(%d, %d) = %d\n", (int) regs.arg1,
                (int) regs.arg2, (int) regs.retval);
        entering = 1;
    }
}

void monitor_mkdir()
{
    if (entering) {
        entering = 0;
    } else {
        char *path = get_path(regs.arg1);
        regs.retval = lind_mkdir(path, regs.arg2);
        set_args(&regs);
        fprintf(stdout, "[monitor] mkdir(%s) = %d\n", path, (int)
                regs.retval);
        entering = 1;
    }
}

void monitor_chdir()
{
    if (entering) {
        entering = 0;
    } else {
        char *path = get_path(regs.arg1);
        regs.retval = lind_chdir(path);
        set_args(&regs);
        fprintf(stdout, "[monitor] chdir(%s) = %d\n", path, (int)
                regs.retval);
        entering = 1;
    }
}

```

```
void monitor_getcwd()
{
    if (entering) {
        entering = 0;
    } else {
        fprintf(stdout, "[monitor] getcwd(%s)\n", get_path(regs.arg1));
        entering = 1;
    }
}

void monitor_dup()
{
    if (entering) {
        entering = 0;
    } else {
        int lind_fd = lind_dup(regs.arg1);
        if (lind_fd >= 0) {
            add_mapping(regs.retval, lind_fd);
            regs.retval = lind_fd;
        } else {
            regs.retval = -1;
        }
        set_args(&regs);
        fprintf(stdout, "[monitor] dup(%d) = %d\n", (int) regs.arg1,
                (int) regs.retval);
        entering = 1;
    }
}

void monitor_dup2()
{
    if (entering) {
        entering = 0;
    } else {
        int lind_fd = lind_dup2(regs.arg1, regs.arg2);
        if (lind_fd >= 0) {
            add_mapping(regs.retval, lind_fd);
            regs.retval = lind_fd;
        } else {
            regs.retval = -1;
        }
        set_args(&regs);
        fprintf(stdout, "[monitor] dup2(%d, %d) = %d\n", (int) regs.arg1,
                (int) regs.arg2, (int) regs.retval);
        entering = 1;
    }
}
```

```
}

void monitor_dup3()
{
    if (entering) {
        entering = 0;
    } else {
        int lind_fd = lind_dup3(regs.arg1, regs.arg2, regs.arg3);
        if (lind_fd >= 0) {
            add_mapping(regs.retval, lind_fd);
            regs.retval = lind_fd;
        } else {
            regs.retval = -1;
        }
        set_args(&regs);
        fprintf(stdout, "[monitor] dup3(%d, %d, %d) = %d\n", (int) regs.arg1,
                (int) regs.arg2, (int) regs.arg3, (int) regs.retval);
        entering = 1;
    }
}

void monitor_flock()
{
    if (entering) {
        entering = 0;
    } else {
        regs.retval = lind_flock(regs.arg1, regs.arg2);
        set_args(&regs);
        fprintf(stdout, "[monitor] flock(%d, %d) = %d\n", (int) regs.arg1,
                (int) regs.arg2, (int) regs.retval);
        entering = 1;
    }
}

void monitor_epoll_create1()
{
    if (entering) {
        entering = 0;
    } else {
        int lind_fd = lind_epoll_create1(regs.arg1);
        if (lind_fd >= 0) {
            add_mapping(regs.retval, lind_fd);
            regs.retval = lind_fd;
        } else {
            regs.retval = -1;
        }
        set_args(&regs);
    }
}
```



```
    fprintf(stdout, "[monitor] epoll_create1(%d) = %d\n", (int) regs.arg1,
            (int) regs.retval);
    entering = 1;
}
}

void monitor_epoll_create()
{
    if (entering) {
        entering = 0;
    } else {
        int lind_fd = lind_epoll_create(regs.arg1);
        if (lind_fd >= 0) {
            add_mapping(regs.retval, lind_fd);
            regs.retval = lind_fd;
        } else {
            regs.retval = -1;
        }
        set_args(&regs);
        fprintf(stdout, "[monitor] epoll_create(%d) = %d\n", (int) regs.arg1,
            (int) regs.retval);
        entering = 1;
    }
}

void monitor_getdents()
{
    if (entering) {
        entering = 0;
    } else {
        char *buf = malloc (regs.arg3);
        regs.retval = lind_getdents(regs.arg1, buf,
            regs.arg3);
        set_mem(regs.arg2, buf, regs.arg3);
        set_args(&regs);
        fprintf(stdout, "[monitor] getdents(%d, %d) = %d\n", (int) regs.arg1,
            (int) regs.arg3, (int) regs.retval);
        free(buf);
        entering = 1;
    }
}

void monitor_lseek()
{
    if (entering) {
```

```

        entering = 0;
    } else {
        regs.retval = lind_lseek(regs.arg1, regs.arg2, regs.arg3);
        set_args(&regs);
        fprintf(stdout, "[monitor] lseek(%u, %d, %d) = %d\n", (int) regs.arg1,
                (int) regs.arg2, (int) regs.arg3, (int) regs.retval);
        entering = 1;
    }
}

void monitor_pwritev()
{
    if (entering) {
        entering = 0;
    } else {
        regs.retval = lind_pwrite(regs.arg1, get_mem(regs.arg2, regs.arg3),
                regs.arg3, regs.arg4);
        set_args(&regs);
        fprintf(stdout, "[monitor] pwritev(%d, 0x%lx[], %d) = %d\n",
                (int)regs.arg1,
                (long) regs.arg2, (int) regs.arg4,
                (int) regs.retval);
        entering = 1;
    }
}

void monitor_pread64()
{
    if (entering) {
        entering = 0;
    } else {
        regs.retval = lind_pread(regs.arg1, get_mem(regs.arg2, regs.arg3),
                regs.arg3, regs.arg4);
        set_args(&regs);
        fprintf(stdout, "[monitor] pread64(%d, 0x%lx[], %d) = %d\n",
                (int) regs.arg1, (long) regs.arg2,
                (int) regs.arg4, (int) regs.retval);
        entering = 1;
    }
}

void monitor_socket()
{
    if (entering) {
        entering = 0;
    } else {

```

```
int lind_fd = lind_socket(regs.arg1, regs.arg2, regs.arg3);

if (lind_fd >= 0) {
    add_mapping(regs.retval, lind_fd);
    regs.retval = lind_fd;
} else {
    regs.retval = -1;
}
set_args(&regs);
fprintf(stdout, "[monitor] socket(%d, %d, %d) = %d\n", (int)
    regs.arg1,
    (int) regs.arg2, (int) regs.arg3, (int) regs.retval);
entering = 1;
}
}

void monitor_bind()
{
    if (entering) {
        entering = 0;
    } else {
        regs.retval = lind_bind(regs.arg1,
            get_mem(regs.arg2, sizeof(struct lind_sockaddr)), regs.arg3);

        set_args(&regs);
        fprintf(stdout, "[monitor] bind(%d) = %d\n", (int) regs.arg1,
            (int) regs.retval);
        entering = 1;
    }
}

void monitor_connect()
{
    if (entering) {
        entering = 0;
    } else {
        regs.retval = lind_connect(regs.arg1,
            get_mem(regs.arg2, sizeof(struct lind_sockaddr)), regs.arg3);

        fprintf(stdout, "[monitor] connect(%d) = %d\n", (int) regs.arg1,
            (int) regs.retval);
        set_args(&regs);
        entering = 1;
    }
}

void monitor_accept()
```

```

{
    if (entering) {
        entering = 0;
    } else {
        int lind_fd = lind_accept(regs.arg1,
            get_mem(regs.arg2, sizeof(struct lind_sockaddr)),
            (lind_socklen_t*) regs.arg3);

        if (lind_fd >= 0) {
            add_mapping(regs.retval, lind_fd);
            regs.retval = lind_fd;
        } else {
            regs.retval = -1;
        }

        set_args(&regs);
        fprintf(stdout, "[monitor] accept(%d) = %d\n", (int) regs.arg1,
            (int) regs.retval);
        entering = 1;
    }
}

void monitor_sendto()
{
    if (entering) {
        entering = 0;
    } else {
        regs.retval = lind_sendto(regs.arg1, get_mem(regs.arg2, regs.arg3),
            regs.arg3, regs.arg4,
            get_mem(regs.arg2, sizeof(struct lind_sockaddr)), regs.arg5);
        set_args(&regs);
        fprintf(stdout, "[monitor] sendto(%d) = %d\n", (int) regs.arg1,
            (int) regs.retval);
        entering = 1;
    }
}

void monitor_recvfrom()
{
    if (entering) {
        entering = 0;
    } else {
        char *var = malloc(regs.arg3);
        struct lind_sockaddr * buff = malloc(regs.arg6);

        regs.retval = lind_recvfrom(regs.arg1, var, regs.arg3, regs.arg4,
            buff, (lind_socklen_t *) regs.arg6);
    }
}

```

```

    set_mem(regs.arg2, var, regs.arg3);
    set_mem(regs.arg5, buff, regs.arg6);

    free(var);
    free(buff);
    set_args(&regs);

    fprintf(stdout, "[monitor] recvfrom(%d) = %d\n", (int) regs.arg1,
            (int) regs.retval);
    entering = 1;
}
}

void monitor_recvmmsg()
{
    if (entering) {
        entering = 0;
    } else {
        struct lind_msghdr msg_orig;

        struct lind_msghdr* msg = get_mem(regs.arg2,
            sizeof(struct lind_msghdr));
        msg_orig = *msg;

        struct lind_iovec* iovs = get_mem((uintptr_t)msg->msg_iov->iov_base,
            sizeof(struct lind_iovec) * msg->msg_iovlen);

        struct lind_iovec* iovs_orig = malloc(
            sizeof(struct lind_iovec) * msg->msg_iovlen);

        memcpy(iovs_orig, iovs, sizeof(struct lind_iovec) * msg->msg_iovlen);

        for (int i = 0; i < msg->msg_iovlen; ++i) {
            iovs[i].iov_base = malloc(iovs[i].iov_len);
        }

        msg->msg_iov = iovs;
        msg->msg_name = malloc(msg->msg_namelen);
        msg->msg_control = malloc(msg->msg_controllen);

        regs.retval = lind_recvmmsg(regs.arg1, msg, regs.arg3);

        set_mem((uintptr_t)msg_orig.msg_name, msg->msg_name,
            msg->msg_namelen);
        set_mem((uintptr_t) msg_orig.msg_control, msg->msg_control,
            msg->msg_controllen);
    }
}

```

```

    for (int i = 0; i < msg->msg_iovlen; ++i) {
        set_mem((uintptr_t) iovs_orig[i].iov_base, iovs[i].iov_base,
               iovs_orig[i].iov_len);
    }
    free(iovs);
    free(iovs_orig);
    free(msg);

    set_args(&regs);

    fprintf(stdout, "[monitor] recvmsg(%d, %d) = %d\n", (int) regs.arg1,
           (int) regs.arg3, (int) regs.retval);
    entering = 1;
}
}

void monitor_sendmsg()
{
    if (entering) {
        entering = 0;
    } else {
        struct lind_msghdr* msg = get_mem(regs.arg2,
                                         sizeof(struct lind_msghdr));

        struct lind_iovec* iovs = get_mem((uintptr_t) msg->msg_iov->iov_base,
                                         sizeof(struct lind_iovec) * msg->msg_iovlen);
        struct lind_iovec* iovs_orig = (struct lind_iovec*) malloc(
            sizeof(struct lind_iovec) * msg->msg_iovlen);
        memcpy(iovs_orig, iovs, sizeof(struct lind_iovec) * msg->msg_iovlen);
        for (int i = 0; i < msg->msg_iovlen; ++i) {
            iovs[i].iov_base = get_mem((uintptr_t) iovs[i].iov_base,
                                       iovs[i].iov_len);
        }
        msg->msg_iov = iovs;
        msg->msg_name = malloc(msg->msg_namelen);
        msg->msg_control = malloc(msg->msg_controllen);

        regs.retval = lind_sendmsg(regs.arg1, msg, regs.arg3);

        free(msg);
        free(iovs);
        free(iovs_orig);
        set_args(&regs);

        fprintf(stdout, "[monitor] sendmsg(%d) = %d\n", (int) regs.arg1,
               (int) regs.retval);
        entering = 1;
    }
}

```

```

    }
}

lind_socklen_t* addrlen;
void monitor_getsockname()
{
    if (entering) {
        entering = 0;
    } else {

        struct lind_sockaddr *buff = malloc(sizeof (struct lind_sockaddr));
        regs.arg3 = (long)get_mem(regs.arg3, sizeof(lind_socklen_t));
        regs.retval = lind_getsockname(regs.arg1, buff,
            (lind_socklen_t*) regs.arg3);

        set_mem(regs.arg2, buff, sizeof(struct lind_sockaddr));
        set_args(&regs);
        fprintf(stdout, "[monitor] getsockname(%d, %d) = %d\n", (int)
            regs.arg1,
            (int) regs.arg2, (int) regs.retval);
        free(buff);
        entering = 1;
    }
}

void monitor_getsockopt()
{
    if (entering) {
        entering = 0;
    } else {
        struct lind_sockaddr *buff = malloc(sizeof(struct lind_sockaddr));
        regs.retval = lind_getsockopt(regs.arg1, regs.arg2, regs.arg3, buff,
            (lind_socklen_t*) regs.arg5);
        set_mem(regs.arg4, buff, sizeof(struct lind_sockaddr));
        set_args(&regs);

        fprintf(stdout, "[monitor] getsockopt(%d, %d, %d, %d) = %d\n", (int)
            regs.arg1,
            (int) regs.arg2, (int) regs.arg3, (int) regs.arg5, (int)
            regs.retval);

        free(buff);
        entering = 1;
    }
}

void monitor_setsockopt()

```

```

{
    if (entering) {
        entering = 0;
    } else {
        regs.retval = lind_setsockopt(regs.arg1, regs.arg2, regs.arg3,
            get_mem(regs.arg4, sizeof(struct lind_sockaddr)), regs.arg5);
        set_args(&regs);
        fprintf(stdout, "[monitor] setsockopt(%d, %d, %d, %d) = %d\n", (int)
            regs.arg1,
            (int) regs.arg2, (int) regs.arg3, (int) regs.arg5, (int)
            regs.retval);
        entering = 1;
    }
}

void monitor_socketpair()
{
    if (entering) {
        entering = 0;
    } else {
        int lind_fd = lind_socketpair(regs.arg1, regs.arg2,
            regs.arg3, get_mem(regs.arg4, 2 * sizeof(int)));

        if (lind_fd >= 0) {
            add_mapping(regs.retval, lind_fd);
            regs.retval = lind_fd;
        } else {
            regs.retval = -1;
        }
        set_args(&regs);
        fprintf(stdout, "[monitor] socketpair(%d, %d) = %d\n", (int)
            regs.arg1,
            (int) regs.arg2, (int) regs.retval);
        entering = 1;
    }
}

void monitor_getpeername()
{
    if (entering) {
        entering = 0;
    } else {
        fprintf(stdout, "[monitor] getpeername(%d) = %d\n", (int) regs.arg1,
            (int) regs.retval);
        entering = 1;
    }
}

```



```

}

void monitor_select()
{
    if (entering) {
        entering = 0;
    } else {

        void* set1 = get_mem(regs.arg2, sizeof(fd_set));
        void* set2 = get_mem(regs.arg3, sizeof(fd_set));
        void* set3 = get_mem(regs.arg4, sizeof(fd_set));
        void* tv = get_mem(regs.arg5, sizeof(struct timeval));

        regs.retval = lind_select(regs.arg1, set1, set2, set3, tv);
        set_mem(regs.arg2, set1, sizeof(fd_set));
        set_mem(regs.arg3, set2, sizeof(fd_set));
        set_mem(regs.arg4, set3, sizeof(fd_set));
        set_mem(regs.arg5, tv, sizeof(struct timeval));
        set_args(&regs);
        fprintf(stdout, "[monitor] select(%d) = %d\n", (int) regs.arg1,
                (int) regs.retval);
        entering = 1;
    }
}

void monitor_poll()
{
    if (entering) {
        entering = 0;
    } else {
        struct lind_pollfd * lpf = malloc(sizeof(struct lind_pollfd));
        regs.retval = lind_poll(lpf, regs.arg2, regs.arg3);
        set_mem(regs.arg1, lpf, sizeof(struct lind_pollfd));
        set_args(&regs);
        fprintf(stdout, "[monitor] poll(%d) = %d\n", (int) regs.arg1,
                (int) regs.retval);
        entering = 1;
    }
}

void monitor_epoll_ctl()
{
    if (entering) {
        entering = 0;
    } else {

        struct lind_epoll_event *event = malloc(

```

```

        sizeof(struct lind_epoll_event));
regs.retval = lind_epoll_ctl(regs.arg1, regs.arg2, regs.arg3,
    get_mem(regs.arg4, sizeof(struct lind_epoll_event)));

set_mem(regs.arg4, event, sizeof(struct lind_epoll_event));
set_args(&regs);
fprintf(stdout, "[monitor] epoll_ctl(%d) = %d\n", (int) regs.arg1,
    (int) regs.retval);
entering = 1;
}
}

void monitor_epoll_wait()
{
    if (entering) {
        entering = 0;
    } else {
        struct lind_epoll_event *event = malloc(
            sizeof(struct lind_epoll_event));
regs.retval = lind_epoll_wait(regs.arg1, event, regs.arg3, regs.arg4);
set_mem(regs.arg2, event, sizeof(struct lind_epoll_event));
set_args(&regs);
fprintf(stdout, "[monitor] epoll_wait(%d) = %d\n", (int) regs.arg1,
    (int) regs.retval);
entering = 1;
    }
}

void monitor_os()
{
    if (entering) {
        entering = 0;
    } else {
        fprintf(stdout, "[monitor] %s() = %d\n", syscall_names[regs.syscall],
            (int) regs.retval);
entering = 1;
    }
}

void monitor_gettid()
{
    if (entering) {
        entering = 0;
    } else {
        fprintf(stdout, "[monitor] gettid() = %d\n",
            (int) regs.retval);
entering = 1;
    }
}

```

```

    }
}

void monitor_arch_prctl()
{
    if (entering) {
        entering = 0;
    } else {
        if (((int32_t) regs.arg5) >= 0) {
            fprintf(stdout, "[monitor] arch_prctl() = %d\n",
                (int) regs.retval);
        }
        entering = 1;
    }
}

/* Handle the mmap call through the monitor */
void monitor_mmap()
{
    if (entering) {
        entering = 0;

        if (regs.arg5 >= 0){
            regs.arg5 = get_mapping(regs.arg5);
            set_args(&regs);
        }

    } else {
        if (!regs.arg1) {
            fprintf(stdout, "[monitor] mmap(NULL, %lu, %d, %d, %d, %#llx) =
                0x%lx\n",
                (long) regs.arg2, (int) regs.arg3, (int) regs.arg4, (int)
                regs.arg5, (long long unsigned int) regs.arg6,
                (long int) regs.retval);
        } else {
            fprintf(stdout, "[monitor] mmap(0x%lx, %lu, %d, %d, %d, %#llx) =
                0x%lx\n", (long) regs.arg1,
                (long) regs.arg2, (int) regs.arg3, (int) regs.arg4, (int)
                regs.arg5, (long long unsigned int) regs.arg6, (long
                int) regs.retval);
        }
        entering = 1;
    }
}

void monitor_munmap()
{

```

```
    if (entering) {
        entering = 0;
    } else {
        fprintf(stdout, "[monitor] munmap(%#lx, %lu) = %d\n", (long)
            regs.arg1, (long) regs.arg2, (int) regs.retval);
        entering = 1;
    }
}

void monitor_mprotect()
{
    if (entering) {
        entering = 0;
    } else {
        fprintf(stdout, "[monitor] mprotect(%#lx, %lu, %d) = %d\n", (long)
            regs.arg1, (long) regs.arg2, (int) regs.arg3, (int) regs.retval);
        entering = 1;
    }
}

void monitor_brk()
{
    if (entering) {
        entering = 0;
    } else {
        fprintf(stdout, "[monitor] brk(%#lx) = 0x%llx \n", regs.arg1,
            regs.retval);
        entering = 1;
    }
}

void monitor_exit_group()
{
    fprintf(stdout, "[monitor] exit_group(%d) \n", (int) regs.arg1);
}

void monitor_exit()
{
    fprintf(stdout, "[monitor] exit(%d) \n", (int) regs.arg1);
}

void monitor_tgkill()
{
    if (entering) {
        entering = 0;
    } else {
        fprintf(stdout, "[monitor] tgkill() = %d\n", (int) regs.retval);
    }
}
```

```
    entering = 1;
  }
}

/* Handle generic the generic system calls */
void monitor_gscall(int call_no)
{
  if (entering) {
    entering = 0;
  } else {
    fprintf(stdout, "[monitro] %s() = %d\n", syscall_names[call_no],
            (int) regs.retval);
    entering = 1;
  }
}

int main(int argc, char** argv)
{
  /* Check the command line arguments to see if a process is defined for
  trace */
  if (argc <= 0) {
    fprintf(stderr, "Usage %s <program> <options>\n", argv[0]);
    exit(-1);
  }

  init_ptrace(argc, argv);
  LindPythonInit();
  intercept_calls();
  return 0;
}

/* Initialize a process to be traced */
void init_ptrace(int argc, char** argv)
{
  char ** argv1 = malloc(sizeof(char*) * argc);
  memcpy(argv1, argv + 1, sizeof(char*) * (argc - 1));
  argv1[argc - 1] = NULL;

  load_config();
  tracee = fork();

  /* Check if fork was successful */
  if (tracee < 0) {
    fprintf(stderr, "No process could be monitored.\n");
    exit(-1);
  }
}
```

```

/* Trace the child */
if (tracee == 0) {
    /* Let the parent process to trace the child*/
    ptrace(PTRACE_TRACEME, tracee, 0, 0);

    /* Stop the current process*/
    kill(getpid(), SIGSTOP);

    extern char **environ;

    execve(argv[1], argv1, environ);

    fprintf(stderr, "Unknown command %s\n", argv[1]);
    exit(1);
}

}

/* Intercept the system calls issued by the tracee process */
void intercept_calls()
{
    int status = -1, syscall_num = -1;

    /* Wait for the child to stop */
    waitpid(tracee, &status, 0);

    ptrace(PTRACE_SETOPTIONS, tracee, 0,
           PTRACE_O_TRACESYSGOOD | PTRACE_O_TRACEEXIT);

    while (1) {

        /* Get every syscall and notify in the tracee stops */
        ptrace(PTRACE_SYSCALL, tracee, 0, 0);

        /* Wait for other syscalls */
        waitpid(tracee, &status, 0);

        /* Check if tracee is terminated */
        if (WIFEXITED(status))
            break;

        if (!WIFSTOPPED(status)) {
            fprintf(stderr, "wait(&status)=%d\n", status);
            exit(status);
        }
    }
}

```

```
get_args(&regs);
syscall_num = regs.syscall;

if (WSTOPSIG(status) == (SIGTRAP | 0x80)) {
    if (monitor_actions[syscall_num] == DENY_LIND) {
        monitor_deny();
        break;
    } else if (monitor_actions[syscall_num] == ALLOW_OS) {

        switch (syscall_num) {

        case __NR_execve:
            monitor_execve();
            break;

        case __NR_tgkill:
            monitor_tgkill();
            break;

        case __NR_getcwd:
            monitor_getcwd();
            break;

        case __NR_exit_group:
            monitor_exit_group();
            break;

        case __NR_exit:
            monitor_exit();
            break;

        case __NR_arch_prctl:
            monitor_arch_prctl();
            break;

        case __NR_munmap:
            monitor_munmap();
            break;

        case __NR_mmap:
            monitor_mmap();
            break;

        case __NR_mprotect:
            monitor_mprotect();
            break;
```

```
    case __NR_gettid:
        monitor_gettid();
        break;

    case __NR_brk:
        monitor_brk();
        break;

    default:
        monitor_os();
        break;
} /* switch*/

} else if (monitor_actions[syscall_num] == ALLOW_LIND) {

    switch (syscall_num) {

        case __NR_getuid:
            monitor_getuid();
            break;

        case __NR_read:
            monitor_read();
            break;

        case __NR_open:
            monitor_open();
            break;

        case __NR_openat:
            monitor_openat();
            break;

        case __NR_access:
            monitor_access();
            break;

        case __NR_close:
            monitor_close();
            break;

        case __NR_rmdir:
            monitor_rmdir();
            break;

        case __NR_stat:
```



```
    monitor_stat();
    break;

case __NR_statfs:
    monitor_statfs();
    break;

case __NR_fstat:
    monitor_fstat();
    break;

case __NR_fstatfs:
    monitor_fstatfs();
    break;

case __NR_write:
    monitor_write();
    break;

case __NR_mkdir:
    monitor_mkdir();
    break;

case __NR_chdir:
    monitor_chdir();
    break;

case __NR_getcwd:
    monitor_getcwd();
    break;

case __NR_dup:
    monitor_dup();
    break;

case __NR_dup2:
    monitor_dup2();
    break;

case __NR_dup3:
    monitor_dup3();
    break;

case __NR_getpid:
    monitor_getpid();
    break;
```

```
case __NR_geteuid:
    monitor_geteuid();
    break;

case __NR_getgid:
    monitor_getgid();
    break;

case __NR_getegid:
    monitor_getegid();
    break;

case __NR_unlink:
    monitor_unlink();
    break;

case __NR_link:
    monitor_link();
    break;

case __NR_fcntl:
    monitor_fcntl();
    break;

case __NR_listen:
    monitor_listen();
    break;

case __NR_shutdown:
    monitor_shutdown();
    break;

case __NR_flock:
    monitor_flock();
    break;

case __NR_getdents:
    monitor_getdents();
    break;

case __NR_lseek:
    monitor_lseek();
    break;

case __NR_pread64:
    monitor_pread64();
    break;
```

```
case __NR_pwritev:
    monitor_pwritev();
    break;

case __NR_socket:
    monitor_socket();
    break;

case __NR_bind:
    monitor_bind();
    break;

case __NR_connect:
    monitor_connect();
    break;

case __NR_accept:
    monitor_accept();
    break;

case __NR_sendto:
    monitor_sendto();
    break;

case __NR_recvfrom:
    monitor_recvfrom();
    break;

case __NR_recvmsg:
    monitor_recvmsg();
    break;

case __NR_sendmsg:
    monitor_sendmsg();
    break;

case __NR_getsockname:
    monitor_getsockname();
    break;

case __NR_getsockopt:
    monitor_getsockopt();
    break;

case __NR_setsockopt:
    monitor_setsockopt();
```

```

        break;

    case __NR_socketpair:
        monitor_socketpair();
        break;

    case __NR_getpeername:
        monitor_getpeername();
        break;

    case __NR_select:
        monitor_select();
        break;

    case __NR_poll:
        monitor_poll();
        break;

    case __NR_epoll_create:
        monitor_epoll_create();
        break;

    case __NR_epoll_create1:
        monitor_epoll_create1();
        break;

    case __NR_epoll_ctl:
        monitor_epoll_ctl();
        break;

    case __NR_epoll_wait:
        monitor_epoll_wait();
        break;

    default:
        monitor_ns();
        break;
} /* switch*/
} /* WSTOPSIG*/
} /* while */
}

/* Get the path of files required by a system call through the defined
address */
char *get_path(uintptr_t addr)
{

```

```

size_t len = PATH_MAX;
char *buffer = buffer = malloc(len);

uint32_t tmp;
int i = 0;

while (1) {
    if (i >= len) {
        len *= 2;
        buffer = realloc(buffer, len);
    }

    tmp = ptrace(PTRACE_PEEKDATA, tracee, addr + i, NULL);
    memcpy(buffer + i, (void *) &tmp, sizeof(tmp));

    if (memchr(&tmp, 0, sizeof(tmp)) != NULL) {
        break;
    }

    i += 4;
}

return buffer;
}

/* Set the memory from an address to a specific buffer */
void set_mem(uintptr_t addr, void * buff, size_t count)
{
    long ret = -1;
    int i;

    int fullblocks = count / sizeof(long);
    int remainder = count % sizeof(long);

    for (i = 0; i < fullblocks; i++) {
        ret = ptrace(PTRACE_POKEDATA, tracee,
            (char *) (addr + sizeof(long) * i),
            *(long*) ((char*) buff + sizeof(long) * i));
    }

    if (remainder) {
        unsigned long value = ptrace(PTRACE_PEEKDATA, tracee,
            (char *) (addr + sizeof(long) * fullblocks), 0);
        value = (ret & (ULONG_MAX << (remainder * 8)))
            | (*(long*) ((char*) buff + sizeof(long) * fullblocks)
                & ~(ULONG_MAX << (remainder * 8))));
        ret = ptrace(PTRACE_POKEDATA, tracee,

```

```

        (char *) (addr + sizeof(long) * fullblocks), value);
    }
}

/* Get count number of memory defined through an address */
void *get_mem(uintptr_t addr, size_t count) {

    long ret;
    int i;
    long *mem = malloc((count / sizeof(long) + 1) * sizeof(long));

    for (i = 0; i < count / sizeof(long) + 1; i++) {
        ret = ptrace(PTRACE_PEEKDATA, tracee,
            (char *) (addr + sizeof(long) * i), 0);
        mem[i] = ret;
    }

    return (void*) mem;
}

/* Return system call number by name */
int get_syscall_num(char *name)
{
    int i;

    for (i = 0; i < TOTAL_SYSCALLS; i++){
        if (syscall_names[i] && !strcmp(syscall_names[i], name))
            return i;
    }
    fprintf(stderr, "Syscall %s not found.\n", name);
    return -1;
}

/* Return the arguments of a system call by PTRACE */
void get_args(struct syscall_args *args)
{
    if (ptrace(PTRACE_GETREGS, tracee, 0, &args->user) < 0) {
        fprintf(stderr, "ptrace could not get the register arguments. \n");
        return;
    }
    args->syscall = args->user.regs.orig_rax;
    args->retval = args->user.regs.rax;

    args->arg1 = args->user.regs.rdi;
    args->arg2 = args->user.regs.rsi;
    args->arg3 = args->user.regs.rdx;
}

```

```

args->arg4 = args->user.regs.r10;
args->arg5 = args->user.regs.r8;
args->arg6 = args->user.regs.r9;

}

/* Set the arguments of a system call by PTRACE */
void set_args(struct syscall_args *args)
{
    args->user.regs.orig_rax = args->syscall;
    args->user.regs.rax = args->retval;
    args->user.regs.rdi = args->arg1;
    args->user.regs.rsi = args->arg2;
    args->user.regs.rdx = args->arg3;
    args->user.regs.r10 = args->arg4;
    args->user.regs.r8 = args->arg5;
    args->user.regs.r9 = args->arg6;

    if (ptrace(PTRACE_SETREGS, tracee, 0, &args->user) < 0) {
        fprintf(stderr, "ptrace could not set registers . \n");
        return;
    }
}

/* Load the config file containing the policies to dispatch the system
calls */
int load_config()
{
    char *str, buff[100];
    char *key, *value;
    enum monitor_action mact = ALLOW_LIND;
    const char * config_file = get_lind_config();

    FILE *fp = fopen(config_file, "r");
    if (fp == NULL) {
        fprintf(stderr, "[monitor] Config file %s could not be opened. \n ",
            config_file);
        exit(-1);
    }

    int var;
    for (var = 0; var < TOTAL_SYSCALLS; var++) {
        monitor_actions[var] = NO_VAL;
    }

    while ((str = fgets(buff, sizeof buff, fp)) != NULL) {

```

```
if (buff[0] == '\n' || buff[0] == '#')
    continue;

char* sep;
if ((sep = strchr(str, '=')) {
    *sep++ = 0;
    key = strdup(str);
    value = strdup(sep);

    if(strcmp(value, "ALLOW_LIND\n") == 0) {
        mact = ALLOW_LIND;
    } else if (strcmp(value, "DENY_LIND\n") == 0) {
        mact = DENY_LIND;
    } else if (strcmp(value, "ALLOW_OS\n") == 0) {
        mact = ALLOW_OS;
    }
}

int sys = get_syscall_num(key);

if (monitor_actions[sys] == NO_VAL){
    monitor_actions[sys] = mact;
} else {
    fprintf(stderr, "Duplicated call definitions %s %d \n",
            syscall_names[sys], sys);
    exit(-1);
}
}
}

fflush(fp);

if (fclose(fp) <0) {
    fprintf(stderr, "Could not close %s \n", config_file);
    return -1;
}

return 0;
}
```

D.3 System Call Forwarding in Lind

Listing D.3: Example of invoking Repy library system calls through the monitor to send a message

```

/* This function packs the msg_iovec to a string and passes it to be
   handled by Repy. */
ssize_t lind_sendmsg(int sockfd, const struct lind_msghdr *msg, int flags) {

    /* Strip the msg->iov vector and extract each individual vector to be
       joined in a string. */
    unsigned int i = 0;
    /* Total length of all iovec elements */
    int total = 0;
    /* Length of individual iovec elements */
    int * lengths;
    char * concatenated;
    char * final_message;

    /* Memory allocation to store the lengths of the individual elements */
    lengths = malloc(msg->msg_iov->iov_len * sizeof(int));

    concatenated = malloc(total + 1);

    for (i = 0; i < msg->msg_iov->iov_len; i++) {
        lengths[i] = msg->msg_iov[i].iov_len;
        total += lengths[i];
    }

    /* memory allocation for the concatenated strings */

    concatenated = malloc(total);
    final_message = concatenated;

    for (i = 0; i < msg->msg_iov->iov_len; i++) {
        int j;
        for (j = 0; j < lengths[i]; j++) {
            memcpy(final_message + j, msg->msg_iov[i].iov_base,
                msg->msg_iov[i].iov_len);
        }

        final_message += lengths[i];
    }
}

```

```
free(lengths);
free(concatenated);

return ParseResponse(
    MakeLindSysCall(LIND_safe_net_sendmsg, "[is#s#is#ii]", sockfd,
        msg->msg_name, msg->msg_namelen, final_message,
        msg->msg_iovlen, msg->msg_control, msg->msg_controllen,
        msg->msg_flags, flags), 0);
}
```

Listing D.4: Example of invoking Repy library system calls through the monitor to receive a message

```

/* Delivers the received message from a socket to Repy */
ssize_t lind_recvmmsg(int sockfd, struct lind_msghdr *msg, int flags) {

    /* Strip the msg->iov vector and extract each individual vector to be
       joined in a string. */
    unsigned int i = 0;
    /* Total length of all iovec elements. */
    int total = 0;
    /* Length of individual iovec elements. */
    int * lengths;
    char * concatenated;
    char * final_message;

    /* Memory allocation to store the lengths of individual elements. */
    lengths = malloc(msg->msg_iov->iiov_len * sizeof(int));

    concatenated = malloc(total + 1);

    for (i = 0; i < msg->msg_iov->iiov_len; i++) {
        lengths[i] = msg->msg_iov[i].iov_len;
        total += lengths[i];
    }

    /* Memory allocation for the concatenated strings. */

    concatenated = malloc(total);
    final_message = concatenated;

    for (i = 0; i < msg->msg_iov->iiov_len; i++) {
        int j;
        for (j = 0; j < lengths[i]; j++) {
            memcpy(final_message + j, msg->msg_iov[i].iov_base,
                   msg->msg_iov[i].iov_len);
        }

        final_message += lengths[i];
    }

    free(lengths);
    free(concatenated);

    return ParseResponse(
        MakeLindSysCall(LIND_safe_net_recvmmsg, "[is#s#is#ii]", sockfd,

```

```
msg->msg_name, msg->msg_namelen, final_message,  
msg->msg_iovlen, msg->msg_control, msg->msg_controllen,  
msg->msg_flags, flags), 0);
```

```
}
```

List of Abbreviations

ABAC	Attribute-Based Access Control
ACL	Access Control List
AES	Advanced Encryption Standard
AM	Application Management
API	Application Programming Interface
ARM	Advanced RISC Machine
ASLR	Address Space Layout Randomization
AWS	Amazon Web Services
BCR	Binding Corporate Rules
BioBankCloud	Scalable, Secure Storage Biobank
BIOS	Basic Input/Output System
CDR	Call Detail Record
CIA	Confidentiality, Integrity, Availability
CMS	Cambridge Monitor System
CP	Control Program
CPTM	Cloud Privacy Threat Modeling
CPU	Central Processing Unit
CRM	Customer Relationship Management
CSA	Cloud Security Alliance
CSV	Comma Separated Values
CVE	Common Vulnerabilities and Exposures
DAC	Discretionary Access Control
DM	Data Management
DNA	Deoxyribo Nucleic Acid
DNS	Domain Name System
DoS	Denial of Service
DPD	Data Protection Directive
EC2	Elastic Compute Cloud
eCPC	e-Science for Cancer Prevention and Control

EGI	European Grid Infrastructure
EJB	Enterprise JavaBeans
ESX	Elastic Sky X
ETE	External Trusted Entity
EU	European Union
fMRI	Functional Magnetic Resonance Imaging
GFS	Google File System
GLBA	Gramm-Leach-Bliley Act
GLM	General Linear Model
GSS-API	Generic Security Services Application Program Interface
GUI	Graphical User Interface
GW	Generic Worker
HDFS	Hadoop Distributed File System
HIPAA	US Health Insurance Portability and Accountability Act
HMAC	Keyed-Hashing for Message Authentication
HOPS	Hadoop Optimized File System
HOTP	HMAC-based One-time Password
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
I/O	Input/Output
IaaS	Infrastructure-as-a-Service
IAM	Identity and Access Management
IAPP	International Association of Privacy Professionals
IBE	Identity-Based Encryption
IBS	Identity-Based Signature
IETF	Internet Engineering Task Force
IoT	Internet of Things
IP	Internet Protocol
IPC	Inter Process Communication
JAAS	Java Authentication and Authorization Service
JDBC	Java Database Connectivity
JDT	Job Details Table
JM	Job Management
JSDL	Job Submission Description Language
JSF	Java Server Faces
JSON	JavaScript Object Notation

JVM	Java Virtual Machine
KDC	Key Distribution Center
KVM	Kernel Virtual Machine
LDAP	Lightweight Directory Access Protocol
LIMS	Lab Information Management System
LOC	Line of Code
LSM	Linux Security Module
LTP	Linux Test Project
LXC	Linux Container
MAC	Mandatory Access Control
MCR	MATLAB Compiler Runtime
MD5	Message-Digest algorithm 5
ME	Method Engineering
MPHC	Multiterminal Cut for Privacy in Hybrid Clouds
MR	Map/Reduce
NaCl	Google Native Client
NDB	Network Database Technology
NGS	Next-Generation Sequencing
NIST	National Institute of Standards
OATH	Open Authentication
OAUTH	Open Authorization
OGSA-BES	Open Grid Service Architecture - Basic Execution Services
OS	Operating System
OTP	One Time Password
PaaS	Platform-as-a-Service
PAP	Policy Administration Point
PC	Personal Computer
PCI	Peripheral Component Interconnect
PDP	Policy Decision Point
PEP	Policy Enforcement Point
PID	Personal Identifier
PII	Personally identifiable information
PKI	Public Key Infrastructure
PMES	Programming Model Enactment Services
POSIX	Portable Operating System Interface
PPH	PolyPasswordHashser

QEMU	Quick Emulator
QID	Quasi Identifier
QRC	Quick Response Code
RBAC	Role Based Access Control
REST	Representational State Transfer
RFC	Request for Comments
RFPA	Right to Financial Privacy Act
RNA	Ribo Nucleic Acid
RPC	Remote Procedure Call
S3	Simple Storage Service
SaaS	Software-as-a-Service
SAML	Security Assertion Markup Language
SASL	Simple Authentication and Security Layer
ScaBIA	Scalable Brain Image Analysis
SCIM	System for Cross-domain Identity Management
SCSI	Small Computer System Interface
SDK	Software Development Kit
SDLC	Secure Development Life Cycle
SFI	Software Fault Isolation
SHA-1	Secure Hash Algorithm
SHA-3	Secure Hash Algorithm
SIAC	Standard Information About Consent
SINC	Standard Information on Non-Consented
SLA	Service Level Agreement
SM	Security Management
SMS	Short Message Service
SNP	Single Nucleotide Polymorphism
SOAP	Simple Authentication and Security Layer
SOC1	Service Organization Controls
SPM	Statistical Parametric Mapping
SQL	Structured Query Language
SRPA	Standard Research Project Approval
SSL	Secure Socket Layer
SSN	Social Security Number
SSO	Single Sing-on
STS	Security Token Service
TCB	Trusted Code Base
TCCP	Trusted Cloud Computing Platform
TGS	Ticket Granting Server
TGT	Ticket Granting Ticket

TLS	Transport Layer Security
ToS	Terms of Service
TOTP	Time-based One-time Password
TPM	Trusted Platform Module
TTP	Trusted Third Party
URL	Unified Resource Locator
USB	Universal Serial Bus
UTD	University of Texas at Dallas
UTS	Unix Time Sharing
VENUS-C	Virtual Multidisciplinary EnvironMents USing Cloud Infrastructures
VLAN	Virtual LAN
VM	Virtual Machines
VMM	Virtual Machine Monitor
VO	Virtual Organization
VOMS	Virtual Organization Management Service
VT	Virtualization Technology
XACML	eXtensible Access Control Markup Language
XDAS	Distributed Audit Service
XML	Extensible Markup Language
XSS	Identity-Based Encryption
YOTP	Yubikey One-time Password
ZIP	Zone Improvement Plan