Thèse n° 8263

Privacy-Preserving Federated Analytics using Multiparty Homomorphic Encryption

Présentée le 1er octobre 2021

Faculté informatique et communications
Laboratoire pour la Sécurité des Données
Programme doctoral en informatique et communications

pour l'obtention du grade de Docteur ès Sciences

par

David Jules FROELICHER

Acceptée sur proposition du jury

Prof. C. González Troncoso, présidente du jury
Prof. J.-P. Hubaux, Prof. B. A. Ford, directeurs de thèse
Prof. B. Berger, rapporteuse
Dr W. Dai, rapporteur
Prof. M. Jaggi, rapporteur

École
polytechnique
fédérale
de Lausanne

2021

Abstract

Analyzing and processing data that are siloed and dispersed among multiple distrustful stakeholders is difficult and can even become impossible when the data are sensitive or confidential. Current data-protection and privacy regulations (e.g., GDPR) highly restrict the sharing and outsourcing of personal information among stakeholders that are in different jurisdictions. Sharing data is, however, required in many domains such as finance and medicine. The medical sector is a paradigmatic example: Privacy is paramount and data sharing is needed in numerous applications where data is scarce (e.g., patients with rare diseases) and scattered among multiple stakeholders around the world. Existing privacy-preserving solutions for federated analytics rely either (1) on data centralization or outsourcing to a limited number of entities, which incur multiple security and trust issues, or (2) on the iterative exchange of cleartext aggregated and optionally obfuscated data, which can leak personal information or introduce bias in the final result. In this thesis, our goal is (1) to propose privacy-preserving federated solutions for exploration, and for statistical and machine-learning analyses on data held by multiple distrustful stakeholders, and (2) to analyze and evaluate the proposed systems, thus showing that they provide an efficient, secure, scalable, and accurate alternative to existing solutions for federated analysis by proving their utility in real-world state-of-the-art biomedical studies. In order to do this, we rely on multiparty homomorphic encryption (MHE). MHE combines secure multiparty computation (SMC) techniques with homomorphic encryption (HE) by pooling the advantages of both SMC and HE, i.e., interactivity and flexibility, and by minimizing their disadvantages, i.e., difficulty in scaling to a large number of parties and computation complexity.

First, we design UNLYNX, a system that enables privacy-preserving federated data exploration on a distributed dataset held by multiple data-providers (DPs), where N-1 out of N of the nodes performing the computations can be malicious. To achieve this, we build interactive protocols by relying on ElGamal additive homomorphic encryption (AHE) and ensure that each untrusted-node operation can be publicly verified by means of zero-knowledge proofs (ZKPs). We then explore how statistics, e.g., standard deviation and variance, can be computed by relying on AHE and ZKPs through the design of another system named DRYNX. In DRYNX, we also explore how to limit the influence of an entity that inputs wrong data in the system, and we propose an efficient federated solution for correctness verification.

We also propose SPINDLE, a solution for secure cooperative gradient descent on federated data that we instantiate for the privacy-preserving training and oblivious evaluation of generalized linear models. SPINDLE covers the entire machine-learning workflow, as it enables oblivious predictions to be performed on a trained model that remains secret. It ensures both data and model confidentiality in a passive adversarial model in which N-1 out of N DPs can collude. Finally, we demonstrate that the solutions proposed in this thesis can be efficient enablers for large-scale, highly sensitive, multi-site biomedical studies. We design and test, by replicating recent state-of-the-art medical studies, secure workflows for the federated execution of computations that span from analyses with low computational complexity, such as non-parametric survival analyses often used in oncology, to analyses with high computational complexity such as one of the key tools for genomic studies, genome-wide association studies (GWAS) on millions of variants.

Keywords: federated analytics, federated machine learning, multiparty homomorphic encryption, differential privacy, decentralized systems.

Résumé

L'analyse de données sensibles, personnelles ou confidentielles qui sont isolées, car stockées par plusieurs entités qui ne se font pas confiance, est souvent difficile, voire impossible. Les règles régissant la protection des données et de la vie privée, telle que le RGPD (Règlement général de protection des données) actuellement en vigueur dans l'Union Européenne, rendent le partage de données entre des entités appartenant à des juridictions différentes particulièrement difficile. Pourtant, celui-ci est indispensable dans des domaines tels que la médecine et la finance. Le domaine médical est un exemple paradigmatique dans lequel la protection de la vie privée est primordiale et le partage des données crucial. En effet, les informations médicales d'un patient sont souvent éparpillées à travers plusieurs instituts. De plus, les études médicales nécessitent souvent un grand nombre de sujets et requièrent dès lors le partage d'informations entre un nombre important d'établissements. L'étude de maladies rares, par exemple, nécessite souvent la mise en place d'une collaboration internationale entre une multitude d'instituts.

Il existe deux types principaux de solutions permettant l'analyse de données distribuées tout en en protégeant la confidentialité. Dans le premier, les informations sont soit centralisées dans une base de données unique ou partagée sur un nombre limité d'entités. Dans le second, les données restent en possession de leurs propriétaires qui exécutent des calculs distribués en échangeant des résultats partiels (qui peuvent être légèrement modifiés à des fins de protection). Malheureusement, ces deux modèles ont plusieurs défauts. Le premier requiert que les propriétaires de données fassent confiance à une ou plusieurs entités afin de stocker leurs données, et ceci sans essayer de les déchiffrer. Dans le second, l'échange de résultats partiels peut révéler des informations sensibles sur les données utilisées. De plus, lorsque ces résultats sont partiellement modifiés à des fins de protection, le résultat final peut être trop différent du résultat original. Dans cette thèse, notre objectif est de (1) proposer des solutions permettant l'exploration, l'analyse statistique, ainsi que l'apprentissage automatisé sur des données qui sont distribuées entre plusieurs entités, tout en protégeant la confidentialité de ces données ; (2) évaluer et analyser ces solutions en démontrant qu'elles sont une alternative viable, efficace et sécurisée aux systèmes existants. Dans ce but, nous démontrons leur efficacité dans le cas d'études biomédicales réalisées en collaboration par un grand nombre d'instituts. Nous nous appuyons sur la cryptographie homomorphique distribuée, nous combinons des protocoles interactifs de calculs sécurisés et l'encryption homomorphique de façon à additionner leurs avantages, l'interactivité et la flexibilité, tout en minimisant leurs désavantages : le premier devient inefficace avec un grand nombre d'entités alors que le second peut avoir un coût de

calcul important.
Premièrement, nous concevons UnLynx, un système qui permet l'exploration sécurisée et distribuée de données stockées sur plusieurs entités, dont N-1 sur N peuvent être mal intentionnées. Pour cela, nous élaborons des protocoles interactifs en utilisant le système ElGamal d'encryption homomorphique pour l'addition (EHA). De plus, en nous appuyant sur les preuves à divulgation nulle de connaissance (PDNC), nous nous assurons que chaque opération exécutée dans le système peut être publiquement vérifiée. Ensuite, nous explorons comment nous pouvons calculer des statistiques, telles que la déviation standard ou la variance, en nous appuyant sur EHA et PDNC afin de construire un nouveau système appelé Drynx. Dans ce second chapitre, nous explorons aussi comment limiter l'influence sur le résultat final d'une entité qui entrerait, de façon volontaire ou non, de fausses valeurs dans le système. Nous proposons une solution permettant une vérification efficace et probabiliste des opérations et valeurs entrées dans le système.
Deuxièmement, nous présentons Spindle, une solution permettant l'exécution sécurisée d'une descente de gradient sur des données distribuées. Nous paramétrons ce système afin d'entraîner et évaluer de manière confidentielle des modèles linéaires généralisés. Spindle couvre l'entièreté du processus d'apprentissage automatisé car il permet de faire des prédictions sur un modèle qui a été entraîné sans jamais être révélé. Tout ceci est exécuté en garantissant la confidentialité des données et du modèle entraîné même lorsque toutes les entités sauf une complotent à l'encontre du système ou de l'entité restante.
Troisièmement, nous démontrons que les solutions proposées dans cette thèse permettent l'exécution efficace d'études biomédicales à large échelle sur des données distribuées très sensibles. Nous concevons et testons, en reproduisant des études médicales de pointe, des processus sécurisés servant à l'exécution fédérée de calculs qui vont d'analyses à faible complexité, comme des courbes de survie souvent utilisées en oncologie, à des opérations complexes, telles qu'une étude d'association pangénomique considérant des millions de variants.

Mots Clés : analyse distribuée, apprentissage automatique distribué, encryption homomorphique distribuée, confidentialité différentielle, systèmes distribués.

Zusammenfassung

Die Analyse von Daten, die verstreut, isoliert und von mehreren Stellen gespeichert werden, die sich gegenseitig nicht vertrauen, ist schwierig und kann sogar unmöglich werden, wenn es sich um persönliche, sensible oder vertrauliche Daten handelt. Die aktuellen Datenschutzbestimmungen (z. B. GDPR) schränken die gemeinsame Nutzung und das Outsourcing personenbezogener Daten zwischen Stakeholdern, die sich in unterschiedlichen Rechtsordnungen befinden, stark ein. Dennoch ist die gemeinsame Nutzung von Daten in vielen Bereichen wie der Medizin oder dem Finanzwesen erforderlich. Der medizinische Bereich ist ein besonders krasses Beispiel, wo der Datenschutz an erster Stelle steht und die gemeinsame Nutzung von Daten absolut notwendig ist. Medizinische Informationen sind z. B. bei seltenen Krankheiten schwer zu finden und liegen oft verstreut in mehreren Instituten auf der ganzen Welt.

Lösungen, die eine verteilte Datenanalyse bei gleichzeitigem Schutz der Vertraulichkeit ermöglichen, lassen sich in zwei Modelle einteilen. Im ersten Modell werden die Informationen in einer einzigen Datenbank zentralisiert oder in verschlüsselter Form für eine begrenzte Anzahl von Einheiten freigegeben. Diese Einheiten können dann zusammenarbeiten, um die Berechnungen der Daten durchzuführen. Im zweiten Modell werden die Daten von ihren Besitzern lokal gespeichert und sie tauschen dann Teilergebnisse aus (die zum Schutz leicht modifiziert sein können), um gemeinsam eine definierte Berechnung durchzuführen. Leider haben beide Modelle mehrere Unzulänglichkeiten. Das erste Modell erfordert, dass die Dateneigentümer einer oder mehreren Entitäten vertrauen, die Daten speichern, ohne zu versuchen, sie zu entschlüsseln, während im zweiten Modell der Austausch von Teilergebnissen sensible Informationen preisgeben kann. Wenn Zwischenergebnisse zu Schutzzwecken leicht verändert werden, kann das Endergebnis zu weit vom ursprünglichen Ergebnis entfernt sein. In dieser Arbeit ist es unser Ziel, (1) datenschutzfreundliche föderierte Lösungen, für die Exploration und für statistische und Machine-Learning-Analysen, von Daten vorzuschlagen, die von mehreren vertrauenswürdigen Akteuren gehalten werden, und (2) die vorgeschlagenen Systeme zu analysieren und zu evaluieren und so zu zeigen, dass sie eine effiziente, sichere, skalierbare und genaue Alternative zu bestehenden Lösungen für föderierte Analysen darstellen. Indem wir ihren Nutzen in realen biomedizinischen Studien auf dem neuesten Stand der Technik beweisen. Um dies zu erreichen, setzen wir auf die homomorphe Mehrparteienverschlüsselung (MHE). MHE kombiniert sichere Multiparty Computation (SMC)-Techniken mit homomorpher Verschlüsselung (HE), indem es die Vorteile von SMC und HE, d.h. Interaktivität und Flexibilität, vereint und ihre Nachteile, d.h. Schwierigkeiten bei der Skalierung auf eine große Anzahl von Parteien und Rechenkomplexität, minimiert.

Zunächst entwarfen wir UnLynx, ein System, das eine sichere und verteilte Exploration von Daten ermöglicht, die auf mehreren Entitäten gespeichert sind. UnLynx muss sicher bleiben, wenn alle bis auf eine der Entitäten bösartig sein könnte. Zu diesem Zweck entwickeln wir interaktive Protokolle unter Verwendung der homomorphen Verschlüsselung für Addition (HVA) von ElGamal und stellen sicher, dass jede im System ausgeführte Operation von jeder Entität mithilfe von Zero-Knowledge-Disclosure-Proofs (ZKDP) verifiziert werden kann. Als Nächstes untersuchen wir, wie man Statistiken, wie Standardabweichung oder Varianz, auf der Grundlage von HVA und ZKDP und durch den Aufbau eines neuen Systems namens Drynx berechnen kann. In diesem zweiten Kapitel untersuchten wir auch, wie wir den Einfluss einer Entität, die - freiwillig oder nicht - falsche Werte in das System eingibt, limitieren. Wir schlagen eine Lösung vor, die eine effiziente probabilistische Überprüfung aller in das System eingegebenen Operationen und Werte ermöglicht.
Zweitens schlagen wir Spindle vor, eine Lösung zur sicheren Durchführung des Gradientenabstiegs auf verteilten Daten. Wir parametrisieren dieses System, um generalisierte lineare Modelle vertraulich zu trainieren und auszuwerten. Spindle deckt den gesamten Prozess des maschinellen Lernens ab, da es auch erlaubt, Vorhersagen über ein trainiertes Modell zu treffen, ohne dass dieses jemals offengelegt wird. Der gesamte Prozess wird unter Wahrung der Daten- und Modell-Privatsphäre durchgeführt, selbst wenn alle bis auf eine Entität sich gegen das System oder die verbleibende Entität verschwören.
Schließlich zeigen wir, dass die in dieser Arbeit vorgeschlagenen Lösungen die effiziente Durchführung großer biomedizinischer Studien auf hochsensiblen Daten, die auf verschiedenen Entitäten gespeichert sind, ermöglichen können. Wir entwerfen und testen, indem wir modernste medizinische Studien replizieren. Wir schlagen sichere Prozesse vor, für die föderierte Ausführung von Berechnungen, die von Analysen mit geringer Komplexität reichen, wie z.B. Überlebenskurven, die oft in der Onkologie verwendet werden, bis hin zu komplexen Operationen, wie z.B. einer genomweiten Assoziationsstudie, die Millionen von Varianten berücksichtigt.

Schlüsselwörter: föderierte Analyse, föderiertes maschinelles Lernen, homomorphe Mehrparteienverschlüsselung, differenzielle Privatsphäre, dezentrales System.

Acknowledgements

First and foremost, I am extremely grateful to my advisor, Prof. Jean-Pierre Hubaux, and to my co-advisor, Prof. Bryan Ford. Jean-Pierre, I am grateful for the opportunity that you offered me to join the LCA1 (now LDS) laboratory and for your guidance without which this thesis would have not been possible. I learned a lot working with you and I will keep relying on many of your advice in the future. Bryan, thank you for having welcomed me in the DeDiS laboratory. Your ideas and vision inspired my research from the beginning of my thesis.

I would like to express my gratitude to my thesis committee members, Prof. Bonnie Berger, Dr. Wei Dai, Prof. Martin Jaggi, and Prof. Carmela Troncoso for their valuable feedback and efforts spent reviewing my work. It was a privilege to have them sit on my committee.

I am very thankful to all my co-authors for our fruitful collaborations and their contributions to this thesis. Dr. Juan R. Troncoso-pastoriza, thank you for your continuous, timeless and always kind help. I am deeply indebted for your support and guidance without which this thesis would have not been possible. Dr. Apostolos Pyrgelis, I would like to thank you for your precious and friendly support throughout my thesis. I learned a lot from your critical thinking on our work and research in general. Dr. Jean Louis Raisaro and Dr. Zhicong Huang, thank you for having helped me start my thesis in the right direction and for having remained always ready to help even after you left EPFL. Patricia Egger, thank you for having welcomed me and shown me the bright side of the PhD life from the start.

I had the chance to have great collaborators. I would like to particularly thank Prof. Jacques Fellay and Dr. Michel Cuendet for their valuable input and advice on the medical application side. Thank you, Dr. Hyunghoon Cho and Prof. Berger, I really learned a lot from collaborating with you. I would also like to congratulate all the students with whom I had the pleasure to work: Yiping, Max, Louis, Emily, John Stephan, Benoit, Andres, Augustin, Zahra, Adrien, Jean, Ksandros, Philippe, and Caroline. I am grateful to Holly for her energetic help in improving my writing in English, and to Patricia and Angela for their always friendly help.

I am thankful to all my colleagues and friends at LDS and DeDiS for sharing this journey with me: Ludovic, Sylvain, Sinem, Jean-Philippe, Francesco, Romain, Anh, Alexandra, Nicolas, and Linus. Special thanks to Kirill and Cey for their always energetic and funny support. I am also grateful to Christian (i.e., “PhD buddy”) to have shared many coffees, laughs, and experiences with me. Mickaël, thank you for having shared a part of your time at EPFL with me. I am very grateful for your continuous support and all the fun we had at work and around. Joao, I am immensely grateful for your endless help and the countless hours you spent working with me. This thesis would have simply not been possible without your crucial and always friendly

support.
To conclude, I am still amazed by the always friendly help that I have been privileged to receive anytime I needed it during my work at EPFL. I am grateful to all the people who participated in making this work environment a nice place. My PhD years would not have been the same without the support of my friends and family. “Merci, vielen Dank” to my parents, to Stefan and Emeline, and to my grandparents for their unconditional support and love.

Contents

Abstract i

Acknowledgements vii

Introduction 1

1 Background 7
- 1.1 Multiparty Additive Homomorphic Encryption . . . 7
- 1.2 Zero-Knowledge Proofs . . . 8
- 1.3 Verifiable Shuffle . . . 8
- 1.4 Multiparty Fully Homomorphic Encryption . . . 8
- 1.5 Differential Privacy . . . 10
- 1.6 Blockchains . . . 10
- 1.7 Notations . . . 11

2 Related Work 13
- 2.1 Centralized & Hardware-Based Solutions for Data Exploration and Statistical Analysis . . . 13
- 2.2 Federated Solutions for Data Exploration and Statistical Analysis . . . 14
- 2.3 Federated Solutions for Machine Learning . . . 15

3 Federated Data Exploration 19
- 3.1 Introduction . . . 19
- 3.2 System Overview . . . 21
  - 3.2.1 System Model . . . 21
  - 3.2.2 Threat Model . . . 22
- 3.3 UnLynx Design . . . 23
  - 3.3.1 Decentralized Data-Sharing Protocol . . . 23
  - 3.3.2 Sub-protocols . . . 25
- 3.4 Security Analysis . . . 31
- 3.5 Discussion & Extensions . . . 33
- 3.6 Performance Evaluation . . . 34
  - 3.6.1 UNLYNX Complexity . . . 34
  - 3.6.2 System Implementation . . . 35

3.6.3 System Evaluation 35
3.7 Example Application: Secure Survey 41
3.8 Conclusion 44

4 Federated Statistical Analysis 45
4.1 Introduction 45
4.1.1 Use Cases 45
4.2 System Overview 47
4.2.1 System Model 47
4.2.2 Threat Model 48
4.2.3 Functional Requirements 48
4.2.4 Security Requirements 49
4.3 Drynx Design 49
4.3.1 Drynx Security Design 50
4.3.2 Drynx Optimized Design 58
4.4 Security Analysis 60
4.5 Encodings 60
4.6 Discussion and Extensions 64
4.7 Performance Evaluation 66
4.7.1 System Implementation 66
4.7.2 System Evaluation 66
4.8 Conclusion 73

5 Federated Learning 75
5.1 Introduction 75
5.2 Secure Federated Training and Evaluation 77
5.2.1 Problem Statement 77
5.2.2 Solution Overview 78
5.3 SPINDLE Design 79
5.3.1 Background 80
5.3.2 SPINDLE Protocols 81
5.4 System Operations 84
5.4.1 Cryptographic Operations 84
5.4.2 MAP Vector-Matrix Multiplications 85
5.4.3 Optimized Activation Function 88
5.5 System Configuration 88
5.6 Security Analysis 89
5.7 System Evaluation 91
5.7.1 Theoretical Analysis 91
5.7.2 Empirical Evaluation 91
5.8 Conclusion 97

6 Federated Analytics for Precision Medicine 99

6.1 Introduction 99
6.2 System Overview & Design 100
  6.2.1 System Model & Proposed Workflow 100
  6.2.2 Threat Model 101
  6.2.3 FAMHE Design 102
6.3 Evaluation 107
  6.3.1 Multi-centric Kaplan-Meier Survival Analysis 108
  6.3.2 Multi-centric Genome-Wide Association Study (GWAS) 109
6.4 Comparison With Existing Works 113
6.5 Conclusion 117

7 Conclusion **119**

Conclusion **119**

I Appendix **123**

A Federated Data Exploration **125**
A.1 Zero-Knowledge Proofs 125
A.2 SELECT query 127

B Federated Statistical Analysis **129**
B.1 Error Probability 129

C Federated Learning **131**
C.1 Activation Functions 131
C.2 Security of DBootstrap 132
C.3 Evaluation Complement 133
C.4 Modular Extensions 136
  C.4.1 Malicious Adversaries 136
  C.4.2 More Complex ML Models. 137

D Federated Analytics for Precision Medicine **139**
D.1 Packing for the Secure & Federated Computation of a GWAS 139
D.2 Data Providers Independent Results 142

Bibliography **143**

Curriculum Vitae **163**

Introduction

In our increasingly connected and data-driven world, the need to share and analyze data among multiple entities in a privacy-conscious way has become critical in numerous contexts. Five concrete examples include (i) *medical research*, where patients' sensitive data, from multiple institutions, need to be shared to enable practitioners to better understand and treat complex diseases [54], (ii) *fraud detection*, where a tax authority needs to securely access foreign bank accounts in order to identify potential tax evaders [82], (iii) *public safety*, where security agencies from different countries need to protect their confidential information but also need to share it in order to design effective anti-terrorism strategies [215], (iv) *private surveys*, where institutions need to collect and analyze personal data from citizens or private companies [25], and (v) *commercial collaborations*, where corporations do not want to reveal confidential data but are willing to share some information for mutual benefit, i.e., a balance has to be found between collaboration and competition [164; 228].

Data analytics, e.g., machine learning, usually requires large and diverse datasets [266] to produce generalisable and unbiased results with enough statistical power. In all these domains, assembling sufficiently large datasets has been proven difficult [258] and often requires the sharing of data among many data providers. This is particularly true in medicine, where patients' data are spread among multiple entities. For example, for rare diseases, one hospital might have only a few patients, whereas a medical study requires hundreds (or thousands) of them to produce significant results. Data sharing among many entities, which can be located in multiple countries, is hence required. Recently, as COVID-19 vaccines were distributed across the world, serious but rare side effects have begun to manifest. Vaccine regulators are struggling to assess and study these side effects as they do not have an efficient way to access the victims' medical data. Cases have spread among multiple countries with different jurisdictions and medical data, even at the national level, are highly fragmented [54].

A conceptually simple approach to data sharing consists in aggregating large amounts of data in a centralized database and querying this central repository to analyze the gathered data (e.g., [8; 92; 236]). However, multiple examples show that a centralization of the data can have serious consequences, affecting hundreds of millions of individuals [1; 75]; this was the case with the Equifax breach [75], in which personal information (including social-security numbers and credit-card information) of more than 143 million consumers (about 40% of the US population) was compromised. Centralized solutions are subject to multiple threats as the central database, which stores data from multiple mutually untrusted sources, constitutes a high-value target for possible attackers and a single point of failure. Existing solutions for centralized secure databases [23; 113; 122; 194; 235] usually add a cryptographic layer on top of the query engine or focus exclusively on the data-release privacy, e.g., by using differential privacy [48]. Most of these solutions, however, have a significant performance overhead, do not protect the data during the query execution, and still suffer from a single point of failure. When the data are sensitive and/or personal, they are particularly difficult to collect from (or share across) multiples entities. Stringent data-protection and privacy regulations (e.g., GDPR [91] in Europe) strongly restrict the transfer of personal data, even pseudonymized, across jurisdictions. Hence, it is often impossible to obtain sufficient data to perform meaningful data analysis, e.g., to train ML models that are key enablers in many domains (e.g., medical research [164; 110], financial analysis [232]). We note that limited data inter-operability across different providers is another potential challenge in deploying federated analytics solutions; this, in practice, can be surmounted by harmonizing the data across institutions before performing the analysis.

In this context, decentralized data-sharing systems have raised considerable interest and are key enablers for privacy-conscious big-data analysis. These solutions, in which storage and computation are distributed among multiple entities, can be referred to as *Federated Analytics (FA)* (or federated learning, in the specific context of machine learning). Their core principle is that data providers keep control of their own data and collaborate, e.g., by exchanging local aggregated results or model updates, in order to perform statistical analyses and/or develop machine-learning models, without exchanging their underlying datasets. Thus, FA offers unprecedented opportunities for exploiting large and diversified volumes of data spread across multiple institutions. FA facilitates the development and validation of more generalizable ML algorithms that can yield more accurate and unbiased results.

Despite their clear advantages, the adoption of simple non-secure FA solutions [90; 130; 153; 168; 238] raises important questions about the actual advantage, in terms of eased compliance to regulations over more conventional data-analysis platforms that rely on data centralization [8; 92; 236]. In those, the exchanged information, i.e., partial aggregates and model updates, can still be considered as sensitive (e.g., personal identifying data) [157; 169; 210; 243; 264]. It has been shown that these exchanged data can, under certain circumstances, leak sensitive information about the data providers' datasets, thus lead to re-identification, membership inference, and feature reconstruction [112; 157; 158; 169; 243; 221; 251; 264; 265]. For example, Shokri et al. [221] were able to predict the main procedure that a patient underwent in an hospital, i.e., a highly sensitive fact, by using a dataset containing pseudonimized information about inpatient stays in several health facilities (e.g., age, stay duration, and location), that was publicly released by the Texas Department of State Health Services.

In this respect, the computer security and privacy community has proposed several advanced solutions for privacy-preserving FA based on differential privacy [29; 137], secure multiparty computation (SMC) [4; 21; 27; 49; 52; 73; 78; 89; 95; 111; 116; 117; 118; 142; 163; 170; 178; 203;

225; 257; 259], and homomorphic encryption (HE) [10; 30; 43; 55; 102; 121; 124; 126] in order to protect the exchanged information. Nevertheless, the use of these techniques in FA introduces important trade-offs among privacy, utility, and computational overheads.

To ensure differential privacy [72; 137; 154; 220] during the FA process, e.g., ML training, some recent solutions [29; 137] rely on obfuscation techniques to protect the aggregated data transferred out of data providers. Yet, the use of obfuscation techniques during the computation process introduces a significant trade-off between data privacy and the accuracy of the end results, which can be considered unacceptable in multiple domains [201]. The level of obfuscation needed to protect the aggregated data of each data provider, at each stage of the computation, often generates a final result that is too noisy to be used to derive useful conclusions. Moreover, in multiple settings, e.g., for iterative processes such as ML training, it remains unclear how to set privacy parameters that provide acceptable mitigation of inference risks [119].

Other approaches rely on SMC [4; 21; 27; 49; 52; 73; 78; 89; 95; 116; 117; 118; 142; 163; 170; 178; 203; 225; 257; 259] where the data are obliviously secret-shared and distributed among all participating sites or a set of computing parties. Some of these solutions assume that a majority of the computing parties are honest and do not collude among them, e.g., to retrieve the users' original data. These parties then collaborate to perform a joint analysis on the data without revealing intermediate values. However, the use of SMC-based solutions includes several drawbacks that hinder their adoption in real-world use-cases: for example, intrinsic limitations in terms of control over data, high network overheads, and the difficulty to scale to large numbers of data providers.

Finally, a few solutions [10; 30; 43; 55; 102; 121; 124; 126] rely on the use of HE to enable participating sites to securely centralize their data in an encrypted form to a third party (e.g., cloud provider), who can perform meaningful operations on them without learning anything about their content. Yet, standard fully homomorphic-encryption schemes, i.e., schemes enabling arbitrary computations on encrypted data, are still considered non-viable due to the high computational and storage overheads they introduce. Furthermore, these solutions introduce a single point of failure because only one entity holds the decryption key. Most of these systems (and many solutions mentioned before) rely on honest-but-curious or trusted third-party assumptions that might not provide sufficient guarantees when the data to be shared are highly sensitive, valuable, influential, or private.

As a result of the immaturity of these secure solutions, it is common practice to rely on only limiting legal agreements rather than on technical solutions. Stakeholders willing to share their sensitive data can stipulate data-use agreements with a centralized trusted third-party that becomes fully responsible for collecting, storing, and managing the data for the whole ecosystem. Yet, this approach has proven to not be future-proof and to be particularly difficult to realize on a large scale. Hence, it is more urgent than ever to develop new operational tools that enable many data providers to *protect* and *efficiently analyze* their sensitive data while maintaining control over them.

In this thesis, we explore and design privacy-preserving systems that enable a large number of untrusted data providers to efficiently and collaboratively perform data analytics on their joint datasets. To this end, we combine secure multiparty computation, HE, and differential privacy to build on their features while eliminating their respective disadvantages, i.e., difficulty in scaling to large numbers of data providers, computational overhead, by-design bias introduction on the final result, and restrictive trust assumptions.

Our main contributions are as follows:

Chapter 3 - Federated Data Exploration. We present UNLYNX, a decentralized system for efficient privacy-preserving data exploration. It enables end-users to perform SQL-like queries over encrypted distributed data in order to compute useful descriptive statistics (e.g., count/sum) on a selected subset of data records. The computations are undertaken by a set of computing nodes that constitute a collective authority whose goal is to verifiably compute on data held by multiple data providers. UNLYNX guarantees the confidentiality, unlinkability between data providers and their data, privacy of the end result, and the correctness of computations by the computing nodes. Furthermore, to support differentially private queries, UNLYNX can collectively add noise under encryption. All of this is achieved through a combination of a set of new distributed and secure protocols that are based on additive homomorphic cryptography, verifiable shuffling, and zero-knowledge proofs. UNLYNX is highly parallelizable and modular by design, as it enables multiple security/privacy vs. runtime tradeoffs. Our evaluation shows that UNLYNX can execute a secure survey on 400,000 personal-data records that contain 5 encrypted attributes, distributed over 20 independent databases, for a total of 2,000,000 ciphertexts, in 24 minutes on commercial servers.

Chapter 4 - Federated Statistical Analysis. By improving upon and relying on certain techniques introduced in the previous chapter, we propose DRYNX, an operational, decentralized, and secure system that enables queriers to compute statistical functions and to train and evaluate simple machine-learning models, e.g., with a small number of features, on data hosted at different sources (i.e., on distributed datasets). DRYNX not only ensures the same security properties as UNLYNX (Chapter 3) (i.e., data confidentiality, data providers' (*DPs*) privacy, differential privacy, and computation correctness). But it also ensures that strong outliers, either maliciously or erroneously input by data providers, cannot influence the results beyond a certain limit (which we denote by *result robustness*). Furthermore, DRYNX enables an efficient and decentralized verification of the input data and of all the system's computations, thus it provides auditability in a strong adversarial model in which no entity has to be individually trusted. DRYNX is highly modular, dynamic, and parallelizable. Our evaluation shows that it enables the computation of the variance on a dataset (containing 600,000 records) distributed among 10 data providers in less than 1 second. The computations are distributed among 6 computing nodes, and DRYNX enables the verification of the query execution's correctness in less than 4.5 seconds.

Chapter 5 - Federated Learning. In this chapter, we address the problem of privacy-preserving distributed learning and the evaluation of machine-learning models by analyzing it in the widespread MapReduce abstraction that we extend with privacy constraints. We design SPINDLE (Scalable Privacy-preservINg Distributed LEarning), the first distributed and privacy-preserving system that covers the complete ML workflow by enabling the execution of a cooperative gradient-descent and the evaluation of the obtained model and by preserving data and model confidentiality in a passive-adversary model with up to all-but-one colluding parties. SPINDLE uses multiparty fully homomorphic encryption to execute parallel high-depth computations on encrypted data without significant overhead. We rely here on a fully homomorphic scheme, whereas in Chapters 3 and 4 we use an additive scheme. We notably build on this evolution to explore the secure and federated execution of more complex computations with higher dimensions. We instantiate SPINDLE for the training and evaluation of generalized linear models on distributed datasets. And we show that it is able to accurately (on par with non-secure centrally-trained models) and efficiently (due to a multi-level parallelization of the computations) train models that require a high number of iterations on large input data with thousands of features, distributed among hundreds of data providers. For instance, it trains a logistic-regression model on a dataset of one million samples with 32 features distributed among 160 data providers in less than three minutes.

Chapter 6 - Federated Analytics for Precision Medicine. In this last chapter, we showcase the fact that the systems and techniques presented in the previous chapters can be efficient enablers in the medical domain where the sensitive data must be shared, and where the computed results must be precise. Using real-world evidence in biomedical research, an indispensable complement of clinical trials, requires access to large quantities of patient data that are typically held separately by multiple healthcare institutions. We propose FAMHE, a novel federated analytics system that, based on multiparty homomorphic encryption, enables privacy-preserving analyses of distributed datasets by yielding highly accurate results without revealing any intermediate data. We demonstrate the applicability of FAMHE to essential biomedical analysis tasks, including Kaplan-Meier survival analysis in oncology and genome-wide association studies in medical genetics. Using our system, we accurately and efficiently reproduce two published centralized studies in a federated setting, enabling biomedical insights that are not possible from individual institutions alone. Our work represents a necessary key step towards overcoming the privacy hurdle in enabling multi-centric scientific collaborations.

Software. We designed and developed operational prototypes of the systems described in this thesis. UNLYNX (Chapter 3) is at the core of the MedCo software [155] that is being deployed across multiple hospitals in Switzerland and beyond [200]. UNLYNX provides the cryptographic operations and the interactive secure protocols for the MedCo privacy-preserving cohort exploration and analysis tool on distributed databases. This effort and the integration of the solutions presented in this thesis are at the core of and supported by two important collaborative and multi-disciplinary funded projects [63; 155]. SPINDLE (Chapter 5) is at the origin of the nascent TuneInsight startup that orchestrates collaborations, e.g., trains machine-learning models, on sensitive federated data.

Publications & Patents. Chapter 3 contains the findings of [84]. This paper was accepted and submitted at PETS'17 [189]. It then led to the filing of the patent *System and Method for Providing a Collective Decentralized Authority for Sharing Sensitive Data* [81]. In Chapter 4, we describe the contributions of [88], which was accepted in 2020 by the Journal *IEEE Transactions on Information Forensics and Security.* Chapter 5 is built on the results of [85]. This paper was accepted and presented at PETS'21 [190]. It set the framework and led to the publication of the paper entitled *POSEIDON: Privacy-Preserving Federated Neural Network Learning* [208], which was accepted at NDSS'21 [171] and which is not covered in this thesis. Both [85] and [208] resulted in the filing of patents, [86] and [209], respectively. Finally, in Chapter 6, we describe the conclusions of [87]; it is under revision by *Nature Communications* at the time of this writing.

Thesis Outline. In Chapter 1, we introduce multiparty homomorphic encryption and the other main building blocks of our work. In Chapter 2, we position this thesis with respect to the state of the art in privacy-preserving data analytics. We propose federated and privacy-preserving systems for data exploration (in Chapter 3), statistical analysis (in Chapter 4), and machine learning (in Chapter 5). Finally, we show that the proposed solutions can be used to enable efficient and accurate federated analytics for precision medecine (in Chapter 6).

1 Background

We introduce the main building blocks that we built upon in the subsequent chapters. We first describe the additive homomorphic encryption scheme and techniques defined upon it: zero-knowledge proofs of correctness and verifiable shuffle, which are used in Chapters 3 and 4. We then introduce the fully homomorphic encryption scheme and its multiparty construction. Finally, we introduce differential privacy before summarizing the main notations and symbols used in this thesis in Tables 1.1 to 1.4.

1.1 Multiparty Additive Homomorphic Encryption

In Chapters 3 and 4, data are encrypted using the probabilistic and additively-homomorphic ElGamal cryptosystem (ECEG) [74]. ECEG enables an efficient use of zero-knowledge proofs for correctness [38]. ECEG relies on the difficulty of computing a discrete logarithm in a finite field; in this case, an Elliptic Curve subgroup of $\mathbb{Z}_p$, with p a big prime. The encryption of a message $m \in \mathbb{Z}_p$ is $\mathrm{E}_\Omega(m) = (rB,\ mB + r\Omega)$, where r is a uniformly-random nonce in $\mathbb{Z}_p$, B is a base point on an elliptic curve $\mathcal{G}$ and Ω a public key. The additive homomorphic property states that $\mathrm{E}_\Omega(\alpha m_1 + \beta m_2) = \alpha \mathrm{E}_\Omega(m_1) + \beta \mathrm{E}_\Omega(m_2)$ for any messages m_1 and m_2 and for any scalars α and β. In order to decrypt a ciphertext $(rB,\ mB + r\Omega)$, the holder of the corresponding private key ω ($\Omega = \omega B$) multiplies rB and ω, yielding $\omega(rB) = r\Omega$ and subtracts this point from $mB + r\Omega$. The result mB is then mapped back to m, e.g., by using a hashtable. We rely on fixed-point representation to encrypt floating values.

Interactive protocols can be used to distribute the computations and the trust among multiple computing nodes CNs. Each CN_i possesses a private-public key pair (k_i, K_i) where k_i is a uniformly-random scalar in $\mathbb{Z}_p$ and $K_i = k_i B$ is a point on $\mathcal{G}$. The CNs' collective public key is $K = \sum_{i=1}^{\#CN} K_i$. The corresponding secret key $k = \sum_{i=1}^{\#CN} k_i$ is never reconstructed such that a message encrypted by using K can be decrypted only with the participation of all CNs. An attacker would have to compromise all CNs in order to decrypt a message.

1.2 Zero-Knowledge Proofs

Universally-verifiable zero-knowledge proofs (ZKPs) can be used to ensure computation integrity and to prove that encrypted data are within given ranges. In Chapters 3 and 4, we choose to verify computation integrity by using the proofs for general statements about discrete logarithms, introduced by Camenisch and Stadler [38]. These proofs enable a verifier to check that the prover knows the discrete logarithms y_1 and y_2 of the public values $Y_1 = y_1 B$ and $Y_2 = y_2 B$ and that they satisfy a linear equation

$$A_1 y_1 + A_2 y_2 = A, \tag{1.1}$$

where A, A_1, A_2 are public points on $\mathscr{G}$. This is done without revealing any information about y_1 or y_2.

The input-range validation is done by relying on the proofs proposed by Camenisch and Chaabouni [37], with which we can prove that a secret message m lies in a given range $[0, u^l)$ with u and l integers, without disclosing m. The prover writes the base-u decomposition of its secret value m and commits to the u-ary digits by using the verifier signatures on these digits. The l created commitments prove to the verifier that $m \in [0, u^l)$. We present this proof, adapted to our framework, in Chapter 4, Algorithm 4.1. Finally, both proofs can be made non-interactive through the Fiat-Shamir heuristic [80].

1.3 Verifiable Shuffle

To randomly select a value from a public list of noise values and to ensure differential privacy (see Section 1.5), we rely on a verifiable shuffle [18; 104; 172; 173]. We implemented and use the verifiable shuffle of ElGamal pairs, described by Neff [173]. This protocol takes as input a list of χ ElGamal pairs $(C_{1,i},\ C_{2,i})$ and outputs $(\bar{C}_{1,i},\ \bar{C}_{2,i})$ pairs such that for all $1 \leq i \leq \chi$, $(\bar{C}_{1,i}, \bar{C}_{2,i}) = (C_{1,v(i)} + r''_{v(i)} B, C_{2,v(i)} + r''_{v(i)} \Omega)$, where $r''_{v(i)}$ is a re-randomization factor, v is a permutation and Ω is a public key. The permutation v is used to change the order of the ElGamal pairs and $r''_{v(i)}$ is used to modify the value of the ciphertext encrypting a message m such that its decryption still outputs m. As a result, an adversary not knowing the decryption key v and the $r''_{v(i)}$ is unable to link back any ciphertext $(\bar{C}_{1,i}, \bar{C}_{2,i})$ with a ciphertext $(C_{1,i},\ C_{2,i})$. Neff provides a method for proving that such a shuffle is done correctly, i.e., that there exists a permutation v and re-randomization factors $r''_{i,j}$ such that $output = \textsc{shuffle}_{v, r''_{i,j}}(input)$, without revealing anything about v or $r''_{i,j}$. This is achieved by using honest-verifier zero-knowledge proofs, introduced by Neff [172; 173].

1.4 Multiparty Fully Homomorphic Encryption

Multiparty Homomorphic Encryption. In Chapter 5, we rely on a multiparty (or distributed) fully-homomorphic encryption scheme [165] in which the secret key is distributed among the parties, while the corresponding collective public key pk is known to all of them. Thus, each party can independently compute on ciphertexts encrypted under pk but all parties have to collaborate to decrypt a ciphertext. This enables, for example, the data providers (DPs) to train a collectively encrypted model, that cannot be decrypted as long as one DP is honest and refuses to participate in the decryption. As we show in Chapter 5, this multiparty scheme also enables DPs to collectively switch the encryption key of a ciphertext from pk to another public key without decrypting.

Mouchet et al. [165] propose a multiparty version of the Brakerski Fan-Vercauteren (BFV) lattice-based homomorphic cryptosystem [79] and introduce interactive (distributed) protocols: DKeyGen(·) for key generation, DDec(·) for decryption, and DBootstrap(·) for bootstrapping. We apply an adaptation of this multiparty scheme to the Cheon-Kim-Kim-Song cryptosystem (CKKS) [47] that enables approximate arithmetic, and whose security is based on the ring learning with errors (RLWE) problem [150]. CKKS enables arithmetic over $\mathbb{C}^{N/2}$; the plaintext and ciphertext spaces share the same domain $R_Q = \mathbb{Z}_Q[X]/(X^N+1)$, with N a power of 2. Both plaintexts and ciphertexts are represented by polynomials of N coefficients (degree $N-1$) in this domain. A plaintext/ciphertext encodes a vector $\boldsymbol{v}$ of up to $N/2$ values.

Parameters: The CKKS parameters are denoted by the tuple (N, Δ, η, mc), where N is the ring dimension, Δ is the plaintext scale, or precision, by which any value is multiplied before being quantized and encrypted/encoded, η is the standard deviation of the noise distribution, and mc represents a chain of moduli $\{q_0, \ldots, q_L\}$ such that $\Pi_{\iota\in\{0,\ldots,\tau\}} q_\iota = Q_\tau$ is the ciphertext modulus at level τ, with $Q_L = Q$, the modulus of fresh ciphertexts. Operations on a level-τ ciphertext $\langle \boldsymbol{v} \rangle$ are performed modulo Q_τ, with Δ always lower than the current Q_τ. Ciphertexts at level τ are simply vectors of polynomials in R_{Q_τ}, that we represent as $\langle \boldsymbol{v} \rangle$ when there is no ambiguity about their level, and use $\{\langle \boldsymbol{v} \rangle, \tau, \Delta\}$ otherwise. After performing operations that increase the noise and the plaintext scale, $\{\langle \boldsymbol{v} \rangle, \tau, \Delta\}$ has to be rescaled (see the ReScale(·) procedure defined in Table 1.4.2) and the next operations are performed modulo $Q_{\tau-1}$. When reaching level τ_b, $\langle \boldsymbol{v} \rangle$ has to be bootstrapped. The security of the cryptosystem depends on the choice of N, Q, and η, which in this work are parameterized to achieve at least 128-bit of security.

(Distributed) Operations: A vector $\boldsymbol{v}$ of cleartext values can be encrypted with the public collective key pk and can be decrypted with the collaboration of all DPs (DDec(·) protocol, in which each DP_i uses its secret key sk_i, where the set of all DP_is' secret keys is denoted by $\{sk_i\}$). evk is an evaluation key. The DPs can also change a ciphertext encryption from the public key pk to another public key pk' without decrypting the ciphertext, by relying on the DKeySwitch(·) protocol. A distributed decryption operation DDec(·) is a special case of the DKeySwitch(·) where there is no pk', i.e., pk' is 0. Each DP can independently add, multiply, rotate (i.e., inner-rotation of $\boldsymbol{v}$), rescale Rescale(·), or relinearize Relin(·) a vector encrypted with pk. When two ciphertexts are multiplied together, the result has to be relinearized with Relin(·) to preserve the ciphertext size. After multiple Rescale(·) operations, $\langle \boldsymbol{v} \rangle$ has to be refreshed by a collective protocol, i.e., DBootstrap(·), which returns a ciphertext at level L. The dot product $\bullet$ of two encrypted vectors of size a can be executed by a multiplication followed by $log_2(a)$ inner-left rotations and additions. We list all the operations that can be performed independently by the data providers in Scheme 1.4.1 and the interactive protocols in Scheme 1.4.2.

Encrypt: $\langle \boldsymbol{v}\rangle_{pk} = \{\{\langle \boldsymbol{v}\rangle, \tau, \Delta\}, L, \Delta\}_{pk} = \mathrm{Enc}(pk, \boldsymbol{v})$
Decrypt: $\boldsymbol{v} = \mathrm{Dec}(sk, \langle \boldsymbol{v}\rangle_{pk})$
Add: $\{\langle \boldsymbol{v}_1\rangle + \langle \boldsymbol{v}_2\rangle, \min(\tau, \tau'), \max(\Delta, \Delta')\} = \{\langle \boldsymbol{v}_1\rangle, \tau, \Delta\} + \{\langle \boldsymbol{v}_2\rangle, \tau', \Delta'\}$
Mult: $\{\langle \boldsymbol{v}_3\rangle, \min(\tau, \tau'), \Delta\Delta'\} = \mathrm{M}(\{\langle \boldsymbol{v}_1\rangle, \tau, \Delta\}, \{\langle \boldsymbol{v}_2\rangle, \tau', \Delta'\})$
Rot: $\{\langle \boldsymbol{v}'\rangle, \tau, \Delta\} = \mathrm{RotL/R}(\{\langle \boldsymbol{v}\rangle, \tau, \Delta\}, r, evk)$
Rescale: $\{\langle \boldsymbol{v}\rangle, \tau - 1, \Delta'\} = \mathrm{ReScale}(\{\langle \boldsymbol{v}\rangle, \tau, \Delta\})$
Relin: $\{\langle \boldsymbol{v}\rangle, \tau, \Delta\} = \mathrm{Relin}(\{\langle \boldsymbol{v}\rangle, \tau, \Delta\}, evk)$
Bootstrap: $\{\langle \boldsymbol{v}\rangle, \tau, \Delta\} = \mathrm{Bootstrap}(\{\langle \boldsymbol{v}\rangle, 0, \Delta\}, evk)$

Scheme 1.4.1 – CKKS operations.

Distrib. Key Gen: $pk, evks = \mathrm{DKeyGen}(\{sk_i\})$
Distrib. Bootstrap: $\{\langle \boldsymbol{v}\rangle, L, \Delta\} = \mathrm{DBootstrap}(\langle \boldsymbol{v}\rangle, \tau_b, \Delta, \{sk_i\})$
Distrib. Key Switch: $\langle \boldsymbol{v}\rangle_{pk'} = \mathrm{DKeySwitch}(\langle \boldsymbol{v}\rangle_{pk}, pk', \{sk_i\})$
Distrib. Decrypt: $\boldsymbol{v} = \mathrm{DDec}(\langle \boldsymbol{v}\rangle, \{sk_i\})$

Scheme 1.4.2 – Distributed CKKS operations.

1.5 Differential Privacy

Differential privacy is an approach for privacy-preserving reporting results on statistical datasets, introduced by Dwork [68]. This approach guarantees that a given randomized statistic, $\mathcal{M}(DS) = R$, computed on a dataset DS, behaves similarly when computed on a neighbor dataset DS' that differs from DS in exactly one element. More formally, (ϵ, δ)-differential privacy [71] is defined by $\Pr[\mathcal{M}(DS) = R] \leq \exp(\epsilon) \cdot \Pr[\mathcal{M}(DS') = R] + \delta$, where ϵ and δ are privacy parameters: the closer to 0 they are, the higher the privacy level is. (ϵ, δ)-differential privacy is often achieved by adding noise to the output of a function $f(DS)$. This noise can be drawn from the Laplace distribution with mean 0 and scale $\frac{\Delta f}{\epsilon}$, where Δf, the sensitivity of the original real valued function f, is defined by $\Delta f = \max_{D,D'} ||f(DS) - f(DS')||_1$. Other mechanisms, e.g., relying on a Gaussian distribution, have also been proposed [72; 94].

1.6 Blockchains

A *blockchain* is usually a public, append-only ledger that is distributively maintained by a set of nodes and serves as an immutable ledger [128; 166]. Blockchains are at the core of many innovations in a large variety of domains such as cryptocurrencies [166; 247] and health care [132]. Data are bundled into blocks that are validated through the consensus [40; 256] of the maintaining nodes. Each block contains a pointer (i.e., a cryptographic hash) to the previous valid block, a timestamp, a nonce, and application-specific data. The chain of these blocks forms the blockchain.

1.7 Notations

Symbol	**Description**
Q, DP, S,	Querier, Data Provider, set of DPs with cardinality $\|S\|$
DP_R	Root of the tree formed by the DPs
n, n_i	Nbr. of data samples total and per DP_i
FA	Federated Analytics
ML	Machine Learning
MHE	Multiparty Homomorphic Encryption
SMC	Secure Multiparty Computation
SIMD	Single Instruction, Multiple Data
ZKPs	Zero-Knowledge Proofs

Table 1.1 – Table of Recurrent Symbols. (Part 1)

	Chapters 3 & 4
CN	Computing Node
$\mathcal{G}$, B, $\mathbb{Z}_p$ p	Elliptic curve (EC), base point on $\mathcal{G}$, EC subgroup, prime
$\mathrm{E}_\Omega(m) = (C_1, C_2) = (rB,\ mB + r\Omega)$	ElGamal encryption of message m under key Ω, nonce r
K	CNs' public collective key
$(k_i$, $K_i)$	CN_i's private and public keys
A, A_1, A_2, Y_i, y_i	ZKPs public (uppercase) and discrete log. values
CDP, $(\epsilon, \delta, \theta)$	Collective Differential Privacy & privacy parameters
DDT, DRO	Distributed Deterministic Tagging, Random Obfuscation
	Chapter 4
HDS, PDS	Hospitals & Patients Data Sharing
VN, N_{VN}, f_h	Verifying Node, nbr. of VNs and threshold of honest VNs
π, ρ, $\bar{r}_i$	linear combination, *encoding*, records
$\mathbf{V_i} = [v_{i,1}, ..., v_{i,l}]$, c_i	Vector, count
CTA, CTO, $CTKS$	Collective Tree Aggregation, Obfuscation, Key Switch
$[b_l, b_u]$, $[0, u^l)$	Range, default range
p_{ver}, $p_{ver_{sub}}$	Probability of verification for proof and sub-proof
T, T_{sub}	Verification thresholds for proofs and sub-proofs
P_{f_h}	Probability that f_h honest VNs verify the proof

Table 1.2 – Table of Recurrent Symbols. (Part 2)

	Chapter 5
$\boldsymbol{X}_{n\times c}, \boldsymbol{y}_n$	Training dataset with c features, n samples, and label vector
$(\boldsymbol{X}^{(i)}, \boldsymbol{y}^{(i)})$, $(\boldsymbol{X}', \cdot)$	DP_i's part of the dataset and querier's evaluation data
$\boldsymbol{y}'$, cl, $\|cl\|$	Predictions' output, set of class labels and its cardinality
$\boldsymbol{X}[\phi, \cdot], \boldsymbol{X}[\cdot, \phi]$	ϕ^{th} line and column of $\boldsymbol{X}$
$\boldsymbol{y}[\phi]$	Element of vector y
$P(\cdot)$	Protection mechanism
$\boldsymbol{B} \in \boldsymbol{X}, b$	Random mini-batch of b rows
$\boldsymbol{W}_G^{(\cdot,j)}$	Global model at iteration j
$\boldsymbol{W}^{(i,j,l)}$	DP_i's local model at global iteration j, and local iteration l
$\boldsymbol{W}^{(i,j)}$	DP_i's local model at global iteration j
$\boldsymbol{w}_G, \boldsymbol{w}, \boldsymbol{w}^{(i,0)}$	Vector of global, local weights and initial local weights of DP_i
lp, g, z	Learning parameters, nbr. of global and local iterations
$\sigma(\cdot)$, d	Activation function, approximation degree
α, ρ	Learning and elastic rates
sk, pk, evk	Secret, public and evaluation keys
$\bullet$	Dot product
$I(\cdot)$	Model dependent function, e.g., identity
$\langle \boldsymbol{v} \rangle$, $\|ct\|$	Encrypted vector $\boldsymbol{v}$ and size of fresh ciphertext ct
$R_{Q_\tau} = \mathbb{Z}_{Q_\tau}[X]/(X^N + 1)$	Domain of a plaintext/ciphertext at level τ
N, Q, L	Ring dimension, fresh ciphertext modulus, nbr. of available levels
τ_b	Minimum level for Dbootstrap$(\cdot)$
η	Standard deviation of the noise distribution
Δ, mc, q_τ	Plaintext Scale, Chain of moduli variables, modulo of level τ

Table 1.3 – Table of Recurrent Symbols. (Part 3)

	Chapter 6
t_j	Time when at least one event happened
d_j	Nbr. of events at time t_j
n_j	Nbr. of individuals known to have survived (or at risk)
$\hat{S}(t)$	Kaplan-Meier estimator
T	Nbr. of data points
w, p	Model coefficients/weights, Nbr. patients
f, v	Nbr. of features, Nbr. of variants
$X \in \mathbb{R}^{p\times f}$, $y \in \mathbb{R}^{(p\times 1)}$	Covariates matrix, phenotype/label
$V \in \mathbb{R}^{p\times v}$	Variants' matrix
$u \in \mathbb{R}^{p\times 1}$	Vector of 1 variant value for all patients
p_{val}	P-value
$pnorm$	Cumulative distribution function (CDF) of the standard normal distribution
$w[f+2]$, y', MSE	Weight corresponding to the variant
y'	Prediction vector
MSE	Mean-squared error
GJ	Gauss-Jordan method

Table 1.4 – Table of Recurrent Symbols. (Part 4)

2 Related Work

In this chapter, we discuss the state of the art in privacy-preserving data analytics: data exploration and analysis, and federated learning. We also describe some real-world applications of privacy-preserving tools for data analytics in the biomedical domain.

The solutions we propose in this thesis, similarly to standard cleartext non-secure solutions, trade off accuracy with performance but not with privacy. Contrarily to differential privacy-based solutions, they do not, by default, introduce bias in the computed results. Whereas cryptography-based solutions (SMC or HE) are limited in the number of computing parties, our solutions efficiently scale to large numbers of data providers in strong threat models.

2.1 Centralized & Hardware-Based Solutions for Data Exploration and Statistical Analysis

Centralized systems for privacy-preserving data sharing [13; 62; 145; 194] and trusted-hardware-based solutions [181; 213] usually require one entity, i.e., a central entity or a hardware provider, to be trusted; this constitutes a single point of failure. Even though these systems can be more efficient than their decentralized counterparts, they often require a centralization or outsourcing of the data storage, which goes against regulations or is cumbersome to achieve [25] and can be inappropriate for sensitive data. Centralized solutions for medical data sharing [8; 92; 236], for example, have received limited adoption due to their inability to address these concerns [41; 195; 249; 248]. In our solutions, we avoid these issues by decentralizing data-storage, computation, and correctness verification, thus efficiently distributing trust. Furthermore, even in the presence of trusted hardware, side-channel attacks, based on memory and network access patterns, have proven to be effective in many scenarios [53; 143; 180; 250]. This shows the immaturity of deploying such systems - at their current stage - in order to address critical challenges such as secure data sharing. Instead, our solutions are based on well-established cryptographic techniques that rely on standard security models with mathematically proven properties. Their strong security guarantees, coupled with their ability to efficiently support thousands of data providers, make our solutions ready for immediate deployment in operational environments.

2.2 Federated Solutions for Data Exploration and Statistical Analysis

To alleviate the trust assumptions of centralized solutions, distributed (or federated) approaches were proposed.

Non-secure Approaches. In the database-research community, various architectures were proposed for efficient data-sharing and distributed-data management among different institutions. For example, PeerDB [176] is a peer-to-peer (P2P) distributed data-sharing system that offers capabilities for data management, content-based search and a flexible configuration of network topology. Yang et al. [253] propose a hybrid P2P system for distributed data-sharing that combines the efficiency of structured P2P networks and the flexibility of unstructured ones. Duan et al. [67] propose P4P (Peers for Privacy) for privacy-preserving data mining by employing a hybrid architecture that integrates the popular client-server paradigm and decentralizes the computation among a server and a number of peers. The framework assumes an adversary model with a number of constraints, such as a well-protected server and non-collusion between server and peers. These solutions, as opposed to ours, provide almost no security or privacy guarantees for dealing with sensitive data.

Secure Multiparty Computation. In order to execute queries and compute statistics on distributed datasets, multiple decentralized solutions [15; 21; 27; 78; 89; 111; 116; 142; 156; 170; 203; 220; 225; 252; 257; 259] rely on techniques that have a high expressive power, such as secret sharing and garbled circuits. These solutions are often flexible in the computations they offer but usually assume (a) a limited number of honest-but-curious computing parties and (b) no collusion or a two-party model. For example, Cho et al. [49] design a three-party (i.e., 3 computing nodes, servers or cloud-providers) secret-sharing-based solution that enables GWAS execution yet does not reveal information on the input data. A fundamental issue with data-secret sharing, however, is that data providers cannot store and manage their own data; instead, this is handled by the computing nodes. Storing sensitive data at a computing node might not be desirable, or even possible, especially if this computing node is physically in a different country or jurisdiction than the original data holder. Moreover, state-of-the-art garbled circuits libraries cannot appropriately address computations that involve more than two-to-four participants (e.g., [21; 142; 170; 225]). Our solutions are designed to be inherently parallelizable in order to guarantee efficient sharing of sensitive data among any number of data providers with strong security guarantees.

In Chapters 3 and 4, we explore how the systems' computations can be publicly and efficiently verified. This feature is usually not provided by the aforementioned solutions. Although these solutions might efficiently distribute trust, their strong honesty assumptions are risky when the data or the computed statistics are highly sensitive. Bater et al. [15] enable the evaluation of various SQL queries on datasets hosted by a set of distrustful data providers, but both the data providers and the computing entity are trusted to follow the protocol. Corrigan-Gibbs and Boneh [52] propose Prio, a system that ensures privacy, as long as one computing entity out of $|S|$ is honest; but it guarantees end results robustness only in the case where the involved parties are all honest-but-curious. Furthermore, Prio does not protect against DPs colluding among themselves or with the computing nodes. In our solutions, no entity has to be individually trusted in order to provide both privacy and robustness.

Differential Privacy. In Chapter 3, we propose an efficient protocol that provides differential privacy guarantees for queries executed on distributed databases. This solution, as most other existing solutions, relies on the addition of some noise that is derived from a probability distribution such that the final result respects differential privacy. Anandan et al. [9], Narayan et al. [167] and Mohammed et al. [161] propose to use secure two-party computation in order to jointly generate a probability distribution and obliviously derive a noise value from it. These approaches are efficient but limited to only two parties. Chen et al. [44] propose a solution with multiple parties that work together with a trusted proxy in order to add a distributively created noise. Dwork et al. [70] remove the need for a trusted party by using a verifiable secret-sharing of noise values that are then combined in order to generate the noise. Nevertheless, secret sharing imposes that at least two-thirds of the parties are honest, whereas our protocol is secure in an Anytrust Model [246].

Homomorphic Encryption. Systems relying on homomorphic encryption [44; 64; 103; 126; 148; 178; 186; 202] are often limited in the functionalities they offer (e.g., sum only). They present high-performance overhead in comparison with their less secure counterparts or still rely on honest-but-curious parties. In this work, we show how to overcome these limitations and propose systems that enable secure computations of multiple operations in a strong threat model.

2.3 Federated Solutions for Machine Learning

Multiple federated-learning solutions [90; 168; 218; 238] were proposed. In those, the data providers keep their data locally and share only aggregates or training model updates with a central server. For example, DataSHIELD [90] and Vantage6 [238] are generic platforms that translate local commands, e.g., in R, to federated computations, thus enabling distributed data analysis and machine learning (ML). However, multiple research contributions [157; 169; 243] show that these aggregates can still reveal significant information about the data providers' data. For example, Nasirigerdeh et al. propose sPLINK [168], a federated instantiation of the PLINK [193] software to perform a GWAS. With sPLINK, the data providers' partial covariance matrices (i.e., intermediate results) are revealed to the server that aggregates these matrices in order to perform the models training. Although the original data X is not actually transferred, some information about the original data can be inferred from the covariance matrix $X^T X$ computed by the aggregating server. In Chapter 6, we show how the covariance matrix can be collectively and obliviously computed by exchanging encrypted data such that the models can be trained without revealing any intermediate data.

Privacy-Preserving Training of Machine-Learning Models. Some works focus on *securely outsourcing* the training of linear ML models to the cloud, typically by using homomorphic encryption (HE) techniques [10; 30; 55; 102; 124; 126; 191]. For instance, Graepel et al. [102] outsource the training of a linear classifier by employing somewhat HE [79], whereas Aono et al. [10] approximate logistic regression and outsource its computation to the cloud by using additive HE [185]. Jiang et al. [121] present a framework for outsourcing logistic regression training to public clouds by combining HE with hardware-based security techniques (i.e., Software Guard Extensions). In our work, we consider a substantially different setting where the sensitive data are distributed among multiple (untrusted) data providers.

In line with the research direction of *privacy-preserving distributed learning*, most works operate on the two-computing-nodes model, where data owners encrypt or secret share their data among two non-colluding computing nodes that are responsible for the computations. For instance, Nikolaenko et al. [178] combine additive homomorphic encryption (AHE) and Yao's garbled circuits [255] to enable ridge regression on data that are horizontally partitioned among multiple data providers. Gascon et al. [89] extend the work of Nikolaenko et al. in the case of vertically partitioned datasets and improve its computation time by employing a novel conjugate gradient descent (GD) method. Whereas, Giacomelli et al. [95] further reduce computation and communication overheads by using only AHE. Akavia et al. [4] improve the performance of the protocols of Giacomelli et al. [95] by performing linear regression on packed encrypted data. Mohassel and Zhang [163] develop techniques to handle secure arithmetic operations on decimal numbers, and they employ stochastic GD that, along with multi-party-computation-friendly alternatives for non-linear activation functions, supports the training of logistic regression and neural network models. Schoppmann et al. [212] propose data structures that exploit data sparsity in order to develop secure computation protocols for nearest neighbors, naive Bayes, and logistic regression classification. Our work differs from these approaches, as it does not restrict to the two non-colluding computing-node model, and it focuses instead on |S|-party systems, with $|S| \geqslant 2$.

Other distributed and privacy-preserving machine-learning (ML) approaches employ a three-computing-nodes model and rely on secret-sharing techniques to train linear regressions [26], logistic regressions [49], and neural networks [162; 240]. However, such solutions are tailored to the three-party computing-node model and assume an honest majority among the computing parties. An honest majority is also required in the recent work of Rachuri and Suresh [199] who improve on the performance of Mohassel and Rindal [162] by extending its techniques to the four-party setting. Other works focus on the training of ML models among |S|-parties ($|S| \geqslant 4$), with stronger security assumptions, i.e., each party trusts itself. As already mentioned, Corrigan-Gibbs and Boneh [52] present Prio that relies on secret-sharing to enable the training of linear models, and Zheng et al. propose a general-purpose SMC-based collaborative platform [262] and Helen [263], a system that uses HE [185] and verifiable secret sharing [56] to execute ADMM [35] (alternating direction method of multipliers, a convex optimization approach for distributed data), which supports regularized linear models. In Chapter 4, we employ HE, along with encoding techniques, to enable the training of basic regression models and to provide auditability with the use of zero-knowledge proofs. In Chapters 5 and 6, we show how to further build on previous results to enable better scalability in terms of the number of model's features, size of the dataset, and number of data providers; and we offer richer functionalities by relying on the generic and widely-applicable SGD.

Another line of research considers the use of differential privacy for training ML models. Early works [3; 42] focus on a centralized setting where a trusted party holds the data, trains the ML model, and performs the noise addition. Differential privacy has also been envisioned in distributed settings, where to collectively train a model, multiple parties exchange or send differentially private model parameters to a central server [29; 64; 93; 114; 137; 220]. However, the training of an accurate collective model requires very high-privacy budgets, hence it is unclear what privacy protection is achieved in practice [112; 119; 243]. Some works consider hybrid approaches where differential privacy is combined with HE [125; 188], or multi-party computation techniques [120; 234]. We consider differential privacy to be an orthogonal approach; we show in our work how these techniques can be combined with our solutions to protect the final result, e.g., protect the resulting models and their predictions from inference attacks [83; 221].

Privacy-Preserving Prediction on ML Models. Another line of work is focused on privacy-preserving ML prediction, where a party (e.g., a cloud provider) holds a trained ML model on which another party (e.g., a client) wants to evaluate its private input. In this setting, Bost et al. [32] use additive HE techniques to evaluate naive Bayes and decision-tree classifiers, whereas Gilad-Bachrach et al. [96] employ fully homomorphic encryption (FHE) [31] to perform prediction on a small neural network. The computation overhead of these approaches has been further optimized by using multi-party computation (MPC) techniques [204; 206], or by combining HE and MPC [123; 144; 196]. Riazi et al. [205] evaluate deep neural networks by employing garbled circuits and oblivious transfer, in combination with binary neural networks. Boemer et al. [24] propose nGraph-HE2, a compiler that enables service providers to deploy their trained ML models in a privacy-preserving manner. Their method uses HE, or a hybrid scheme that combines HE with MPC, to compile ML models that are trained with well-known frameworks such as TensorFlow [2] and PyTorch [187]. The scope of our work is broader than these approaches, as we account not only for the private evaluation of machine-learning models but also for their privacy-preserving training in the distributed setting.

3 Federated Data Exploration

3.1 Introduction

In this chapter, we present UNLYNX, a new decentralized *operational* solution for efficiently protecting and querying a large amount of sensitive data that is distributed across a multitude of data providers. UNLYNX does not present a single point of failure and has the ability to scale up to large numbers of data providers. It achieves this while guaranteeing (i) data confidentiality at rest and during processing, (ii) unlinkability between data providers and their data, (iii) correctness of secure computations, and (iv) private release of end-results.

UNLYNX is a decentralized system where data providers are able to share their sensitive data without having to trust one single entity to protect their privacy and data confidentiality. In fact, trust is distributed among multiple entities that constitute a collective authority [230]. UNLYNX achieves two distinct types of decentralization. The first is the decentralization of the data, i.e., there is no central repository for all data. Each data provider can store its data on its own premises thus maintaining control over them. The second is the decentralization of the computations, i.e., there is no central authority responsible for all the computations. Instead, a group of collective-authority computing nodes is responsible for securely processing data from the different data providers.

In particular, UNLYNX enables the end-user to perform SQL-like queries over encrypted distributed data, to compute useful descriptive statistics (e.g., count/sum) on a selected subset of data records in a privacy-preserving way. This subset selection is based on a set of Boolean conditions. Although these operations represent only a subset of those supported by alternative approaches based on SMC or trusted third parties, it is also true that such a subset is enough to solve many data-sharing scenarios, e.g., cohort exploration. Moreover, contrary to alternative approaches, our solution is highly parallelizable by design and can scale to large numbers of data providers with millions of data records.

In UNLYNX, during the secure processing, data are homomorphically encrypted under a collective key and shuffled by a set of computing nodes, hence preventing any entity in the ecosystem from linking data back to their respective owners. By generating deterministically encrypted tags from probabilistically encrypted records, UnLynx is also able to filter encrypted records according to the set of Boolean conditions defined in the query. Finally, to prevent inferences based on the end result and to satisfy formal privacy notions (e.g., differential privacy), UnLynx provides a mechanism enabling the collective-authority computing nodes to obliviously perturb the end result with noise (unknown to any party) sampled from a known probability distribution. All computations performed by UnLynx can be verified through the use of cryptographic zero-knowledge proofs.

To evaluate the performance of UnLynx, we built a working prototype implemented as a modular system where the different security features are represented by independent modules that can be activated depending on the application domain and on the privacy/efficiency requirements. An experimental evaluation, in a realistic simulation environment, shows that our prototype scales almost linearly with respect to the amount of data to be shared and the size of the collective authority. A query - with 2 Boolean conditions and 1 grouping criteria, over 400,000 records distributed among 20 data providers, and processed by 3 independent computing nodes - can be executed in less than 24 minutes under the assumption of a strong adversary. By relaxing this assumption (e.g., by considering honest-but-curious computing nodes and deactivating some of the security modules) the execution time of the same query can be reduced to, at best, 2.5 minutes.

Contributions

We make the following contributions:

- A flexible, decentralized, strongest-link security system for privacy-conscious data sharing among a multitude of distributed data providers, built on top of well-established security and privacy techniques, and secure even in a strong adversary model.
- A novel use of collective authorities combined with the use of homomorphic encryption and zero-knowledge proofs.
- A set of new secure and distributed protocols enabling deterministic tagging, key switching and collective aggregating that, combined with a verifiable shuffle enable a set of collective-authority computing nodes to compute on distributed sensitive data and produce zero-knowledge proofs of their work.
- A novel distributed protocol that enables collective-authority servers to obliviously perturb an aggregate query end-result in order to satisfy formal notions of privacy (e.g., differential privacy) and to mitigate inference attacks from a malicious querier.
- A thorough evaluation of our modular system and of the different privacy/efficiency tradeoffs in a realistic simulation environment.

Acknowledgements. This work has been made possible with the help of Patricia Egger, Joao Sa Sousa, Dr. Jean Louis Raisaro, Dr. Zhicong Huang, Christian Mouchet, Prof. Bryan Ford, and Prof. Jean-Pierre Hubaux. We also thank Dr. Juan R. Troncoso-Pastoriza for his valuable feedback and Nicolas Gailly, Ismail Khoffi, and Linus Gasser, from the DeDiS lab for their help with software engineering.

3.2 System Overview

We introduce our system model and discuss UnLynx's functionality, security/privacy and performance requirements. Then we sketch our threat model.

3.2.1 System Model

Our system, as depicted in Figure 3.1, consists of $|S|$ data providers $DP_1, \ldots, DP_{|S|}$, $|CN|$ computing nodes $CN_1, \ldots, CN_{|CN|}$ and a querier Q[1]. Together, the computing nodes form a collective authority (CA) for privacy-preserving data sharing. The DPs combined constitute a distributed database that is used to answer queries. The querier and each of the DPs independently choose one computing node in the CA to communicate with. They can change this choice at any time. Data providers generate and/or store data that can pertain to either one or several individuals.

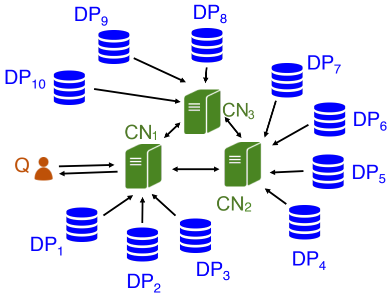


Figure 3.1 – Data providers (in blue), collective-authority computing nodes (in green) and querier (in red). In this example, $|CN| = 3$ and $|S| = 10$. The arrows show the information communication flow.

We assume a public immutable distributed ledger DL that is collectively managed by the computing nodes and contains a complete view of the system, the access rights of the queriers, a list of available DPs, the query history and the system's global variables, such as the privacy parameters. Any change in the topology or querier access rights triggers an update of the public ledger.

We now discuss the functionality that UnLynx must provide along with its security/privacy and performance requirements.

---

[1] There can be several queriers in the system, but as they do not interact with each other, without loss of generality, we consider the protocol for a single one.

Functionality. UNLYNX should permit SQL queries of the form '`SELECT SUM(∗)/COUNT(∗) FROM` $DP_1,\ldots,DP_l$ `WHERE ∗ AND/OR ∗ GROUP BY ∗`' and we consider that '∗' denotes an arbitrary number of attributes. We refer to the attributes involved in the `WHERE` clause and `GROUP BY` statement as filtering attributes. These queries can be executed on the distributed databases held by a set of l chosen DPs. Depending on its permission level, a querier can be limited to some types of queries. Finally, in some specific cases, UNLYNX can also provide the possibility of '`SELECT ∗`' queries. This would, for example, enable DPs to query and decrypt their own database. However, this type of query is not suited for secure data-sharing as it cannot be done on a distributed database held by multiple DPs, while ensuring data confidentiality and privacy. We further discuss this in Appendix A.2.

Security and privacy. UNLYNX should be able to filter query responses based on the Boolean conditions of the query, i.e. attributes in the `WHERE` clause, and to group responses according to the `GROUP BY` statement without revealing any information about any of the attributes or to which groups the responses belong to. The ***confidentiality*** of raw data must be protected at rest and during processing. Moreover, no entity should be able to trace a query response back to its provider, i.e. ***unlinkability*** between DPs and their data must be guaranteed. UNLYNX's primary goals are to enable data sharing, ensure the DPs' privacy and avoid any data leakage. Hence, UNLYNX does not intend to protect queriers' privacy. UNLYNX should permit any entity to check the ***correctness*** of the system's computations and it should ensure that any entity that computes incorrectly can be identified and excluded from future computations. UNLYNX must ensure ***(ϵ, δ)-differential privacy*** for any individual sharing his data. Finally, it should guarantee that only the querier is able to decrypt the result of its query.

Performance. We require UNLYNX to scale linearly with the number of DPs, the amount of data and the size of the CA. It should also provide a shorter response time by relaxing some of the security/privacy requirements.

3.2.2 Threat Model

We define UNLYNX's threat model by discussing the role of each entity in the system.

Collective authority computing nodes. We assume an Anytrust Model [246] for the CA computing nodes. In other words, to achieve the functionality and the security/privacy guarantees described in Section 3.2.1, UNLYNX does not require any particular computing node to be globally trusted or to be honest-but-curious. Instead, as long as there *exists* at least one computing node that is not malicious, these properties are guaranteed.

Data providers. We assume DPs to be honest-but-curious and discuss the impact of having malicious DPs in Section 3.4. UNLYNX does not protect against false information coming from the DPs, i.e., we do not ensure data integrity coming from the DPs. However, in Section 3.5, we propose a mechanism that enables computing nodes to verify that an input is within a range of values, hence mitigating the effect of DPs sending altered data. Each DP can independently choose one computing node in the CA to trust, i.e., each DP can choose a different computing node. Finally, a DP cannot collude with any other entity.

Queriers. We assume queriers to be malicious. They can try to infer sensitive information about DPs and can collude between themselves or with a subset of the CA computing nodes.

We assume that all network communication is authenticated and encrypted. This can be ensured by standard cryptographic techniques, e.g., using TLS protocol. The number of queries accepted per minute by UNLYNX is limited and if a computing node does not respond, it can be removed from the CA.

3.3 UnLynx Design

In order to achieve privacy-preserving sharing of sensitive data, we developed a modular system that enables the use of sub-protocols as building blocks according to security, privacy and performance requirements. In Section 3.3.1, we describe our decentralized data-sharing protocol that is depicted in Figure 3.2. This protocol combines sub-protocols that we describe in more detail in Section 3.3.2.

3.3.1 Decentralized Data-Sharing Protocol

The protocol starts with a querier who wants to retrieve some aggregate information about sensitive data stored by multiple DPs. The query is sent to a chosen computing node that broadcasts it to the other computing nodes in the CA. Then, the computing nodes broadcast the request to all DPs that respond with the requested sensitive data in encrypted form. These data are securely and privately processed by the computing nodes, before the query result is sent to the querier, who is then able to decrypt the final result. Table 1.1 contains the main notation and symbols used in this work.

Step 0. In the initialization phase, each computing node CN_i in the CA creates its own ElGamal key pair (k_i, K_i). The CA constructs its public key $K = \sum_{i=1}^{|CN|} K_i$ that is used by data providers to encrypt their sensitive data. A computing node of the CA generates the probability distribution that corresponds to the predefined differential privacy parameters of the system, ϵ and δ, and it uniformly samples some points from this distribution, based on the probability quantum parameter θ. We assume that these parameters are chosen before the initialization of our system. This choice highly depends on the application domain. The resulting samples are then used as obfuscation noise in Step 6. This process is explained in detail in Section 3.3.2.3.

Step 1. Querier Q sends its query to a computing node CN_i in the CA. An example of a query could be '`SELECT SUM(`$employed$`) FROM` $DP_1, \ldots, DP_d$ `WHERE` $age = E_K(46)$ `AND` $married = E_K(1)$ `GROUP BY` $gender$'. In order for the computing nodes to privately process the query, the attribute values in the `WHERE` clause are encrypted with the CA's public key K.

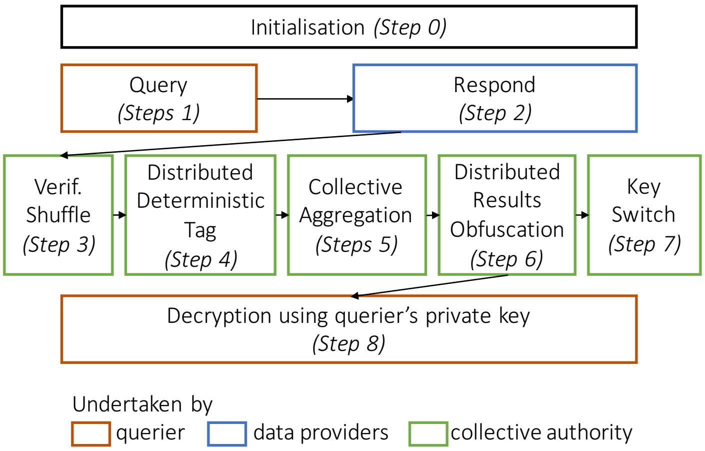

Initialisation (Step 0)
Query (Steps 1)
Respond (Step 2)
Verif. Shuffle (Step 3)
Distributed Deterministic Tag (Step 4)
Collective Aggregation (Steps 5)
Distributed Results Obfuscation (Step 6)
Key Switch (Step 7)
Decryption using querier's private key (Step 8)
Undertaken by
querier
data providers
collective authority

Figure 3.2 – Decentralized Data Sharing Protocol from beginning (DPs encrypt their data using the CA's public key) to end (querier decrypts final aggregate answers using its private key). The initialization step is executed once at the setup of the system.

Finally, CN_i broadcasts the query to all the other collective-authority computing nodes, and each of them sends the query to a different set of DPs such that each DP in the `FROM` clause receives the query. We assume there is no error during the broadcast, hence there are no query duplicates. Before forwarding the query to its DPs or to other computing nodes, each computing node verifies the access rights of the querier in the distributed ledger DL.

Step 2. The DPs select, for each of their data records, the encrypted or clear text values of the attributes specified in the query and send them back to their respective computing node. If the query contains a '`SELECT COUNT`' statement, the DPs append an attribute with value $E_K(1)$ to each of their responses. In order to prevent the computing nodes from knowing the count value, the DPs add dummy records (containing $E_K(0)$) to their responses. A DP can choose to not respond to a query, e.g., if it is too intrusive. Each DP digitally signs its set of responses, publicly logs it in the DL, and appends the signature to its message to ensure authenticity. If some DPs respond with filtering attributes (attributes involved in the `WHERE` clause and `GROUP BY` statement) in clear text, the computing nodes locally aggregate the responses for each different combination of filtering attributes, thus reducing the number of responses to be processed.

Step 3. The CA launches a verifiable shuffle sub-protocol in order to break the link between the DPs and their data. In particular, each computing node sequentially performs a verifiable shuffle on the data, as described in Section 1.3. Any clear text data are encrypted during the verifiable shuffle process. Eventually, each DP's data will be shuffled among themselves and with other DPs' data, once by each computing node.

Step 4. In order to execute the query, all the computing nodes run a distributed deterministic-tag sub-protocol on the filtering attributes. This protocol appends a deterministic tag to each filtering attribute of each DP's response. In other words, this protocol derives a deterministic value, i.e., a point on an elliptic curve, from a probabilistically-encrypted value. The derivation of this tag is explained in Section 3.3.2.1. The outputs of this protocol are used to filter the responses based on the query `WHERE` clause and to group the responses according to the `GROUP BY` statement. For example, if the query is '`SELECT SUM(`$employed$`) FROM` $DP_1,\ldots,DP_d$ `WHERE` $age{=}E_K(46)$ `AND` $married = E_K(1)$ `GROUP BY` $gender$', the `WHERE` clause is transformed to '`WHERE` $age{=}DT(E_K(46))$ `AND` $married{=}DT(E_K(1))$' where $DT(x)$ is the deterministic tag derived from x. The computing node will then check the query predicate for each response by verifying if '$DT(age_{resp_{ij}}) == DT(E_K(46))\ AND\ DT(married_{resp_{ij}}) == DT(E_K(1))$' holds. Here, $married_{resp_{ij}}$ and $age_{resp_{ij}}$ are the values of the married and age attributes of DP_i's response j. If the predicate is not verified, the response is discarded. Finally, the remaining responses are grouped based on the deterministic tag values derived from their `GROUP BY` statement attribute values, and the `SUM/COUNT` attribute values are aggregated in each group. This results in one aggregated response per group.

Step 5. The computing nodes perform a collective tree aggregation protocol, presented in Section 3.3.2.2. When this is done, one computing node has the total aggregate response for each group, encrypted under the CA's public key K.

Step 6. The computing node, which originally received the query from the querier, executes an oblivious results obfuscation sub-protocol and begins by verifying in the public distributed ledger DL if the same query has already been performed. We assume that no entity can know that two different queries yield the same results and we discuss how the verification can be executed with encrypted queries in Section 3.3.2.3. If the same query has already been executed, the computing node adds the same noise value that was used for the first query to the results of the new one. This avoids that the original result of a query can be inferred by repeating the same query and averaging-out the added noise values. If this is not the case, the computing node is responsible for starting a verifiable shuffle protocol (similar to Step 3) on the list of noise values generated in the initialization phase. The computing node chooses the first element in the shuffled list and adds this noise to the query results.

Step 7. The CA launches a key switch sub-protocol to obtain the query results encrypted under the querier's public key U, instead of under the CA's public key K.

Step 8. The total aggregated responses per group are sent to the querier who is able to decrypt them by using its private key u.

3.3.2 Sub-protocols

Here, we provide details about the distributed deterministic-tag, collective tree aggregation, distributed results obfuscation, key switch and dynamic collective-authority sub-protocols that we designed to be independent building blocks. They can be combined to achieve privacy-conscious data sharing with different levels of security, privacy and performance. For the distributed deterministic-tag, distributed results obfuscation and key switch, the collective-authority computing nodes are organized into a cycle as these sub-protocols are performed sequentially. For the collective tree aggregation, however, the CA can be organized into a tree to increase the protocol's efficiency. After each sub-protocol, we show how zero-knowledge proofs can be used to guarantee computation integrity.

3.3.2.1 Distributed Deterministic-Tag

The distributed deterministic-tag sub-protocol, or DDT, consists in tagging an ElGamal ciphertext of a message x, encrypted using the CA key K, with a deterministic value related to x without ever decrypting the ciphertext. This sub-protocol is executed in two successive rounds. We start with $E_K(x) = (C_1,\ C_2) = (rB,\ x + rK)$, the ciphertext tuple corresponding to an ElGamal encryption of message x that uses the CA's public key K.

In the first round, each computing node sequentially generates a fresh secret s_i and adds the value derived from its secret s_iB to C_2. This eliminates the possibility of having a deterministic tag of 0 as an output of the protocol when the input message is zero. After this first round, the encrypted message is $(C_1,\ C_2) = (rB,\ x + rK + \sum_{i=1}^{|CN|} s_iB)$. Let $(\tilde{C}_{1,0},\ \tilde{C}_{2,0}) = (C_1,\ C_2)$ be a ciphertext resulting from the first round.

In the second round, each computing node partially and sequentially modifies this ciphertext. More specifically, when computing node CN_i receives the modified ciphertext $(\tilde{C}_{1,i-1},\ \tilde{C}_{2,i-1})$ from computing node CN_{i-1}, it computes $(\tilde{C}_{1,i},\ \tilde{C}_{2,i})$ as

$$\tilde{C}_{1,i} = s_i\tilde{C}_{1,i-1} \tag{3.1}$$

and

$$\tilde{C}_{2,i} = s_i\left(\tilde{C}_{2,i-1} - \tilde{C}_{1,i-1}k_i\right) \tag{3.2}$$

Once all of these computations are done, we discard the first component $\tilde{C}_{1,|CN|}$ and obtain

$$\tilde{C}_{2,|CN|} = sx + \sum_{i=1}^{|CN|} s_i sB \tag{3.3}$$

where $s = \prod_{i=1}^{|CN|} s_i$ is a short-term collective secret corresponding to the product of each computing node's fresh secret. $\tilde{C}_{2,|CN|}$ is the deterministic tag collectively computed from the original ciphertext (C_1, C_2). In fact, each computing node CN_i uses the same s_i for all the ciphertexts for a given query. Thus, if two messages x_a and x_b are the same, then the corresponding tags will be the same. In our case, this sub-protocol is used to verify the query conditions, the `WHERE` clause and `GROUP BY` statement.

Zero-Knowledge Proofs for the distributed deterministic-tag. Each time a computing node adds a secret value or computes Equations (3.1) and (3.2), it must also compute a zero-knowledge proof to prove that the computations were done correctly. In the second round, when computing $(\tilde{C}_{1,i},\ \tilde{C}_{2,i})$, computing node CN_i is the prover and any entity can act as a verifier. Coming back to Equation (1.1) in Section 1.2, which we recall is $A_1y_1 + A_2y_2 = A$, it is easy to see that for Equation (3.2), $y_1 = s_i$, $y_2 = k_i s_i$ are the discrete logarithms of $s_i B$ and $k_i s_i B = s_i K_i$, respectively. The points $s_i K_i$, $s_i B$, $A = \tilde{C}_{2,i}$, $A_1 = \tilde{C}_{2,i-1}$ and $A_2 = -\tilde{C}_{1,i-1}$ on $\mathcal{G}$ are made public and are part of the proof. The publication of $s_i B$ also guarantees that computing node CN_i has used the same secret s_i for all data during a given query. Similar proofs can be obtained for the first round of the protocol and for Equation (3.1) and are sketched in Appendix A.1.

3.3.2.2 Collective Tree Aggregation

Given the ElGamal ciphertext tuples $(C_{1,i},\ C_{2,i})$ held by each computing node CN_i, the CA will produce one ciphertext $(C_{1,\text{aggr.}},\ C_{2,\text{aggr.}})$ as an aggregation of all ciphertexts $(C_{1,i},\ C_{2,i})$. This aggregation is possible due to the additive homomorphic property of the ElGamal cryptosystem.

In order to improve performance, the CA can be organized into a tree structure, in which each computing node will wait to receive the ciphertext tuples from its children and sum them before passing the result on to its own parent.

Zero-knowledge proofs for the collective tree aggregation. Here, a zero-knowledge proof consists in publishing the ciphertexts and the result of their aggregation. Due to the confidentiality property of the ElGamal cryptosystem, publishing the ciphertexts does not leak any information about the underlying plaintexts. In order to verify these proofs, a verifier can simply sum all of the ciphertexts and check that it corresponds to the published output.

3.3.2.3 Distributed Results Obfuscation

The distributed results obfuscation sub-protocol (DRO) enables the CA to collectively and homomorphically add noise, sampled from a probability distribution satisfying the differential privacy requirements, to the query results. This ensures $(\epsilon,\ \delta)$-differential privacy for DPs without revealing to any entity the amount of noise added. This sub-protocol is composed of two phases: the initialization phase, executed by the CA in the setup of the system; and the runtime phase, performed by the computing nodes in order to respond to each query.

In the initialization phase, to generate the probability-distribution curve, a computing node in the CA uses the globally applicable predefined and publicly available differential privacy parameter ϵ. The same computing node uses the predefined probability quantum parameter θ in order to quantize an approximate representation of the distribution curve. This enables the computing node to derive a list of noise values and to randomly chose a value that can then be added to the query result to ensure $(\epsilon,\ \delta)$-differential privacy. An example of the quantization of a Laplace distribution with $\theta = 0.05$ is depicted in Figure 3.3. We use the number of quantas (blue dots) that fit under the curve in order to approximate the distribution and to create the list of noise values. In our example, the noise value list contains 11 ‘0’s, 4 ‘1’s and ‘-1’s, 2 ‘2’s and ‘-2’s, and so on.

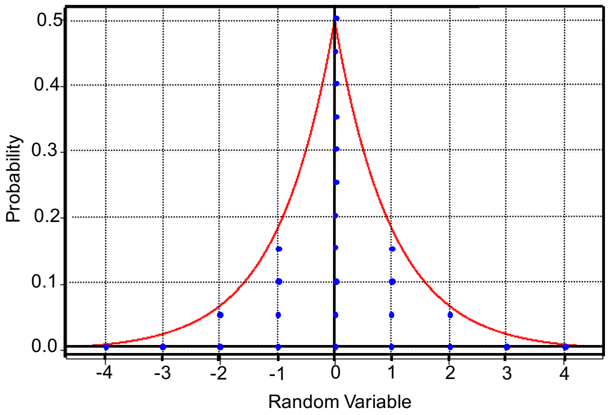


Figure 3.3 – Example of a quantization of Laplace distribution that is then used to derive noise values. The Laplace curve is in red and the quantas are in blue.

In the runtime phase, one computing node starts by verifying if the given query has already been answered by the system. We assume that no entity can know that two different queries yield the same results. To compare two queries, the computing node first retrieves, in the DL, all the queries that have been executed with the same attributes. For each of these, the computing node verifies if the values for the encrypted attributes match those in the new query. In order to do this, it subtracts both queries and tags the result and a 0 by executing a DDT sub-protocol. Finally, the computing node verifies if $\forall j,\ DT(E_K(Q_{new,j} - Q_{i,j}) = DT(E_K(0))$, where $Q_{i,j}$ is the jth attribute of query i and new is the new query. If all these equalities hold, the two queries are equal and the computing node adds the same noise to the new query results. This ensures that the noise cannot be averaged out. If the same query has not already been answered, the computing node starts the verifiable shuffle sub-protocol, described in Step 3, on the list of noise values. The re-randomization in the verifiable shuffle sub-protocol encrypts the clear text noise values. The zero-knowledge proof of the verifiable shuffle ensures the correctness of the computations. This results in a verifiable encryption and shuffling of the noise values. Then, the same computing node chooses the first element in the shuffled list as the noise value to be added. The verifiable shuffle sub-protocol ensures that the noise is chosen randomly from the proper distribution and that no entity knows its value. This noise is added to the query results and stored along with the query in the DL.

Zero-knowledge proofs for the distributed results obfuscation. In the initialization phase, the correctness of the computations can be verified by checking the value of the parameters and the generated noise values that are all publicly stored in the DL. For the runtime phase, the integrity of the computations is ensured by the zero-knowledge proofs of both the verifiable shuffle presented in Section 1.3 and the homomorphic addition described in Section 3.3.2.2. Given the quantization of the distribution, we prove in the following theorem that the proposed mechanism still provides (ϵ, δ)-differential privacy.

Theorem 3.3.1. *Let* $Laplace(0,\ b)$ *with* $b = \frac{\Delta f}{\epsilon}$ *be the Laplace distribution from which the noise is to be sampled,* θ *the unit quanta,* $[-T,\ T]$ *the range of integer noises (*T *is the integer bound), and* L *the length of the generated noise list in the initialization phase. Our mechanism* $\mathscr{M}$ *provides* (ϵ, δ)*-differential privacy where* $\delta = \frac{1}{L}$*, if we choose* $\theta = \frac{1}{2b}e^{-T/b}$*.*

Proof. Let w be the noisy output, μ_1 the original output for dataset D_1, and μ_2 the original output for dataset D_2. We have

$$\begin{aligned}
\frac{Pr[\mathscr{M}(D_1) = w]}{Pr[\mathscr{M}(D_2) = w]} &= \frac{\lceil \frac{1}{2b} e^{-|w-\mu_1|/b}/\theta \rceil / L}{\lceil \frac{1}{2b} e^{-|w-\mu_2|/b}/\theta \rceil / L} \\
&\leq \frac{\frac{1}{2b} e^{-|w-\mu_1|/b}/\theta + 1}{\lceil \frac{1}{2b} e^{-|w-\mu_2|/b}/\theta \rceil} \\
&\leq \frac{\frac{1}{2b} e^{-|w-\mu_1|/b}/\theta}{\frac{1}{2b} e^{-|w-\mu_2|/b}/\theta} + \frac{1}{\lceil \frac{1}{2b} e^{-|w-\mu_2|/b}/\theta \rceil} \\
&= e^{(|w-\mu_2|-|w-\mu_1|)/b} + \frac{1}{\lceil \frac{1}{2b} e^{-|w-\mu_2|/b}/\theta \rceil} \\
&\leq e^{|\mu_1-\mu_2|/b} + \frac{1}{\lceil \frac{1}{2b} e^{-|w-\mu_2|/b}/\theta \rceil} \\
&\leq e^{\epsilon} + \frac{1}{\lceil \frac{1}{2b} e^{-|w-\mu_2|/b}/\theta \rceil}.
\end{aligned} \tag{3.4}$$

Therefore, we have

$$Pr[\mathscr{M}(D_1) = w] \leq e^{\epsilon} Pr[\mathscr{M}(D_2) = w] + \frac{1}{L}. \tag{3.5}$$

Both the first equality in Equations 3.4 and inequality 3.5 come from the definition of a random variable in a Laplace distribution. The last inequality in Equations 3.4 is derived from the Laplace distribution defined in theorem 3.3.1. When w is at the boundary of $\mathscr{M}(D_1)$, w might be the output for D_1 but is not a possible output for D_2, hence $Pr[\mathscr{M}(D_2) = w] = 0$. For example, if $\mu_1 = \mu_2 - 1$, we have $\mathscr{M}(D_1) \in [\mu_1 - T,\ \mu_1 + T]$, and $\mathscr{M}(D_2) \in [\mu_2 - T,\ \mu_2 + T]$, and thus $Pr[\mathscr{M}(D_2) = \mu_1 - T] = 0$. If we choose $\theta >= \frac{1}{2b}e^{-T/b}$, then a boundary noise is sampled with probability $\frac{1}{L}$, hence the formula $Pr[\mathscr{M}(D_1) = w] \leq e^{\epsilon} Pr[\mathscr{M}(D_2) = w] + \frac{1}{L}$ still holds with $Pr[\mathscr{M}(D_2) = w] = 0$. Nevertheless, we should choose T large enough such that $\frac{1}{L}$ is sufficiently low to achieve strong differential privacy. □

3.3.2.4 Key Switch

The key switch sub-protocol enables the conversion of an ElGamal ciphertext of a message x encrypted under the CA's public key K to one of the same message x encrypted under any known public key, e.g., the querier's public key U, without ever decrypting. The sub-protocol is described below.

We start with $\mathrm{E}_K(x) = (C_1,\ C_2) = (rB, x + rK)$, a ciphertext tuple corresponding to the ElGamal encryption of message x using the CA's public key K. Let $(\tilde{C}_{1,0},\ \tilde{C}_{2,0}) = (0,\ C_2)$ be the initial modified ciphertext tuple. Each computing node partially and sequentially modifies this element. More specifically, when computing node CN_i receives $(\tilde{C}_{1,i-1}, \tilde{C}_{2,i-1})$ from computing node CN_{i-1}, it generates a fresh random nonce v_i and computes $(\tilde{C}_{1,i}, \tilde{C}_{2,i})$ as

$$\tilde{C}_{1,i} = \tilde{C}_{1,i-1} + v_i B \tag{3.6}$$

and

$$\begin{aligned} \tilde{C}_{2,i} &= \tilde{C}_{2,i-1} - (rB)k_i + v_i U \\ &= \tilde{C}_{2,i-1} - rK_i + v_i U. \end{aligned} \tag{3.7}$$

where $v = v_1 + \ldots + v_m$. Once all of these are computed, we obtain the new ciphertext that corresponds to x encrypted under the public key U, $(\tilde{C}_{1,m}, \tilde{C}_{2,m}) = (vB, x + vU)$ from the original ciphertext (C_1, C_2). At this point, the ciphertext $(\tilde{C}_{1,m}, \tilde{C}_{2,m})$ can be decrypted only by the holder of the private key u, where $U = uB$.

Zero-knowledge proofs for the key switch. To prove that the computations have been done correctly, each time a computing node computes Equations (3.6) and (3.7), it must also compute a zero-knowledge proof. Again, at each step i, computing node CN_i is the prover and any entity can be the verifier. Coming back to Equation (1.1) in Section 1.2, it is easy to see that for Equation (3.7), $y_1 = k_i$, $y_2 = v_i$ are the discrete logarithms of $k_i B = K_i$ and $v_i B$, respectively. All points K_i, $v_i B$, $A = \tilde{C}_{2,i} - \tilde{C}_{2,i-1}$, $A_1 = -rB$ and $A_2 = U$ are made public and do not leak any information about underlying secrets. A similar proof can be obtained for equation (3.6) and is sketched in Appendix A.1.

3.3.2.5 Dynamic Collective Authority

This sub-protocol enables us to add/remove a computing node to/from the collective authority. On the one hand, adding more computing nodes strengthens the privacy guarantees; on the other hand, detecting misbehavior, for example through the use of zero-knowledge proofs, should lead to the culprits' exclusion from the CA. When a computing node joins/leaves the CA, this computing node has to collaborate with all the DPs in order to change their data encryption from the CA's previous public key to the new one. We assume, here, that the DPs want to outsource computations and that the joining/leaving computing node participates in the protocol.

Without loss of generality, let $CN_{|CN|}$ be the computing node that needs to be added/removed. in order to have its data encrypted under the CA's new public key K_{new}, any DP storing data encrypted using the CA's previous public key K_{prev} must execute the following protocol. When adding a new computing node $CN_{|CN|}$ to the collective authority $CN_1, \ldots, CN_{|CN|-1}$, $K_{\text{prev}} = K_1 + \ldots + K_{|CN|-1}$ and $K_{\text{new}} = K_1 + \ldots + K_{|CN|}$. When removing computing node $CN_{|CN|}$ from $CN_1, \ldots, CN_{|CN|}$, $K_{\text{prev}} = K_1 + \ldots + K_{|CN|}$ and $K_{\text{new}} = K_1 + \ldots + K_{|CN|-1}$. Starting from a message x encrypted under K_{prev}, $(C_1, C_2) = (rB, x + rK_{\text{prev}})$, computing node $CN_{|CN|}$ multiplies C_1 by its private key $k_{|CN|}$

$$C_1 k_{|CN|} = (rB)k_{|CN|} = rK_{|CN|} \tag{3.8}$$

and adds/removes, the result to/from, C_2

$$\tilde{C}_2 = C_2 \pm rK_{|CN|} = x + rK_{\text{prev}} \pm rK_{|CN|} = x + rK_{\text{new}}. \tag{3.9}$$

Component $\tilde{C}_1$ remains the same, i.e., $\tilde{C}_1 = C_1$. The result is the new ciphertext tuple $(\tilde{C}_1, \tilde{C}_2) = (rB, x + rK_{\text{new}})$ corresponding to the same message x encrypted under the CA's new public key. Hence, it is possible to expand the CA and update the corresponding encryptions without needing to decrypt any of the ciphertexts. Finally, the DPs that trusted the computing node who left the CA can choose another computing node to trust or leave the system.

Zero-knowledge proofs for the dynamic collective authority. Using Equation (1.1) in Section 1.2, we see that for Equation (3.9), $y_1 = k_m$ is the discrete logarithm of $k_m B = K_m$. All points K_m, $A = \tilde{C}_2 - C_2$ and $A_1 = -C_1$ are made public and do not leak any information about underlying secrets. A similar proof can be obtained for Equation (3.8) and is sketched in Appendix A.1

3.4 Security Analysis

We analyze UnLynx's security and privacy by studying each step of our decentralized data-sharing protocol presented in Section 3.3.1 and by showing, for each step, how UnLynx guarantees the security and privacy requirements presented in Section 3.2.1.

Step 0. In the initialization step, each computing node i builds its own key pair (k_i, K_i) and, as long as one computing node keeps its secret key k_i hidden, the CA's secret key $k = \sum_{i=1}^{|CN|} k_i$ remains unknown. As a result, data confidentiality is ensured for DPs through the use of the CA's public key $K = \sum_{i=1}^{|CN|} K_i$ to encrypt their data. The correctness of the noise-value generation is ensured by the fact that all the parameters and values are stored in a public, immutable, distributed ledger DL, and the computations can be verified by any entity.

Step 1. UnLynx rules out unauthorized queries by checking the querier's permission in the DL. Moreover, because DPs publicly log the fact that they respond to a query, a computing node's malicious behavior, such as excluding a DP from a query or impersonating a DP, will be caught.

Step 2. Data authenticity is ensured by the DPs' digital signatures on the responses. Moreover, UnLynx enables an optional upper bound that can be used to hide the amount of data sent by DPs. More specifically, DPs either discard some records or add dummy responses that consist of responses with the `SELECT` attributes, all equal to $E_K(0)$; and the filtering attributes uniformly distributed over the range of possible values. This prevents an adversary from inferring any information (e.g., using traffic analysis) from the amount of data sent by a given DP.

As described in Section 3.2.2, we consider DPs to be honest-but-curious, which means that we assume they provide correct responses to a query and do not collude with any other entity. Otherwise, if malicious DPs collude with the querier or some of the computing nodes, they could infer some information about other (non-colluding) DPs. Nevertheless, UnLynx would still offer some protection when this is the case. In fact, when colluding with the querier, malicious DPs would only have access to an approximation of the query results of their target(s) because of the oblivious noise addition done in Step 6. When colluding with one or multiple computing nodes, malicious DPs would be able to observe the output of the Distributed Deterministic Tagging sub-protocol (Step 4). In the worst case, if there were only malicious DPs connected to a malicious computing node, they would be able to infer the mapping between an attribute and its deterministic tag. Yet, the data would be shuffled (Step 3) and mixed with dummy records (Step 2) beforehand. This would still ensure the unlinkability of the data and the secrecy of the honest DPs responses distribution.

Step 3. Unlinkability is guaranteed by the verifiable shuffle sub-protocol. Data confidentiality is maintained as the data are never decrypted during the protocol.

Step 4. The tag $(sx + \sum_{i=1}^{|CN|} s_i sB)$ is a collective encryption of the filtering attribute x because each s_i is known only to computing node CN_i. Hence, the confidentiality of the filtering attributes is guaranteed. Each step of this protocol is publicly verifiable due to the zero-knowledge proofs.

Step 5. A misbehaving computing node can be caught due to the zero-knowledge proofs for homomorphic additions.

Step 6. The input list of noise values is publicly known, and the shuffling and aggregating operations can be verified. Any entity can check that the noise added is one of the values in the list, without learning which one. Moreover, the queries are publicly logged and the same noise is used for identical queries. By providing (ϵ, δ)-differential privacy, UNLYNX is resilient against (colluding) queriers/computing nodes trying to infer information from query outputs and against other types of attacks such as set difference attacks presented by de Souza et al. [59]. UNLYNX maintains a global privacy budget, defined by Dwork [69], which is updated at each executed query by computing the sensitivity or the privacy loss of the query and by subtracting this value to the global budget. In fact, with (ϵ, δ)-differential privacy, the privacy loss is additive and the privacy budget imposes a limit on the cumulative values ϵ and δ after which new queries are not allowed. How to choose the privacy parameters according to the application domain and such that they provide a level of privacy sufficiently high while ensuring that the added noise is not detrimental for data utility and user experience is out of the scope for this work.

Step 7. Each step of the key switch sub-protocol can be verified and the result can be decrypted only by the querier. Each computing node can check that the encryption is indeed switched to the querier's public key and, if this is not the case, any computing node can block the process.

Step 8. If the querier does not receive the final result from the computing node to which it is connected, e.g., because the computing node is unresponsive, it can simply send the same request to another computing node.

In conclusion, data are encrypted during the whole protocol execution, therefore data confidentiality is guaranteed at each step. All computations are publicly verifiable due to the use of zero-knowledge proofs. Finally, UNLYNX provides (ϵ, δ)-differential privacy and unlinkability between DPs and their data through to the use of new distributed and secure sub-protocols, namely distributed deterministic-tag, distributed results obfuscation and verifiable shuffle sub-protocols.

3.5 Discussion & Extensions

By analyzing UNLYNX and its set of protocols, we can identify two potential improvements: (1) data integrity/input validation and (2) a way to exclude a non-cooperative computing node from the CA. We propose two new solutions that can be implemented in the next version of this system.

Input-Range Validation. In this chapter, we assume DPs are honest-but-curious and we do not ensure the correctness of the data they provide. Nevertheless, to limit the effect of DPs who introduce invalid data, we propose a set of input-range validation techniques. Adding input-data validation in UNLYNX is not intended for, and will not help in, situations where a DP injects much invalid data (many data records), but it can limit the damage to cumulative query results if only a few of a DP's records (inadvertently or maliciously) contain invalid data, e.g., a data-entry error by an organization employee. This would enable us to relax the assumption that DPs input only correct data. Camenisch et al. [37] present simple and efficient zero-knowledge proofs to prove that an encrypted value is in a specific range/set. If the encrypted value is an integer I, a zero-knowledge proof consists of proving that $I = \sum I_j \cdot b^j$ and that each b-ary digit of this integer is between $[0,\ b-1]$. For the first part of the proof we can adapt Equation (1.1), and as for the second part, we can use the set membership proofs provided by Camenisch et al. [37].

Enabling a Dynamic Collective Authority. In Section 3.3.2.5, we propose a sub-protocol that enables the system to remove/add a server from/to the CA when this computing node collaborates. However, this is not always the case and UNLYNX might want to exclude a misbehaving computing node that refuses to leave.

The first solution is to require the DPs to re-encrypt their data with the CA's new public key, assuming that DPs keep off-line backups of their data. When this is not possible, we propose that a threshold of t (out of $|CN|-1$) computing nodes reconstruct the secret key of the leaving computing node $CN_{|CN|}$ through the use of a $(t,\ |CN|-1)$-verifiable secret-sharing (VSS) scheme [50]. In such a scheme, a potentially dishonest dealer can share $CN_{|CN|}$'s secret key $k_{|CN|}$, among the $|CN|-1$ remaining computing nodes, in such a way that any t semi-honest computing nodes can reconstruct $k_{|CN|}$ but any subset of $t-1$ computing nodes learn nothing about $k_{|CN|}$. This secret sharing should be done for all computing nodes when they join the CA. In this way, when $CN_{|CN|}$ is removed from the CA, its private key can be reconstructed by the remaining CA computing nodes. This VSS weakens the threat model defined in Section 3.2.2

but enhances the dynamism of the CA by enabling it to discard a misbehaving or unresponsive computing node. In fact, by using a $(t, |CN|-1)$-verifiable secret-sharing scheme, the security of the scheme is guaranteed, as long as t out of $|CN|-1$ computing nodes are honest or honest-but-curious, instead of only 1 in the Anytrust model.

3.6 Performance Evaluation

We start with a theoretical analysis of UnLynx's computation and communication complexities. Then, we discuss our experimental setup and evaluate UnLynx's performance. We consider the performance when producing the results for *one* single query. We demonstrate that UnLynx yields acceptable performance and is scalable with the amount of data and the size of the CA.

3.6.1 UnLynx Complexity

We discuss the time and communication complexities for each of our sub-protocols. We denote by $|CN|$ the number of computing nodes in the CA, r the total number of records sent by all the DPs, and f and g the number of attributes in the `WHERE` clause and in the `GROUP BY` statement, respectively. The number of different combinations of `GROUP BY` attributes is denoted gd and we usually have $r >> gd$. Finally, t is the size of the noise-values list, and a is the number of attributes in the query `SELECT` statement. We discuss the complexity of each sub-protocol and UnLynx's overall complexity.

Verifiable Shuffle. In this sub-protocol, all the ciphertexts have to be shuffled and re-randomized by all the computing nodes, resulting in a computation and communication complexity of $\mathcal{O}(|CN| \cdot r \cdot (f+g+a))$.

Distributed Deterministic Tag. In this sub-protocol, only the attributes in the `WHERE` clause and in the `GROUP BY` statement are processed. Both the computation and communication complexities are therefore $\mathcal{O}(|CN| \cdot r \cdot (f+g))$.

Collective Aggregation. Before executing this sub-protocol, the responses are locally filtered and aggregated by each computing node, which means that the number of responses is reduced from r to $|CN| \cdot gd$. Moreover, the size of the responses is reduced from $g+f+a$ to $g+a$, as the `WHERE` clause attributes are no longer useful and can be discarded. The computing nodes can be organized in a binary tree structure, such that each computing node aggregates the results of its children and its own. Hence, the computation complexity is $\mathcal{O}(log_2(|CN|) \cdot gd \cdot (g+a))$ and the communication is $\mathcal{O}((|CN|-1) \cdot gd \cdot (g+a))$.

Distributed Results Obfuscation. Both the computation and communication complexities of this sub-protocol are $\mathcal{O}(|CN| \cdot t)$ and correspond to the complexity of the noise values shuffle. Other parts of the protocol incur a negligible workload.

Key Switch. The complexity depends mainly on the number of computing nodes and on the size of the responses. Both the computation and communication complexities are $\mathcal{O}(|CN| \cdot gd \cdot (g+a))$.

Overall Complexity. The overall complexity of our protocol can be reduced to the complexity of the verifiable shuffle sub-protocol, that is responsible for most of the computation and communication.

We recall that even if most of the sub-protocols need to be executed sequentially by the computing nodes, the computations performed locally by each computing node are highly parallelizable.

3.6.2 System Implementation

We implemented UnLynx in Go [100], and the experimental code is publicly available at https://github.com/LDS/UnLynx [237]. We relied on Go's native crypto library and the public advanced crypto DeDiS library [61]. The latter includes an implementation of a verifiable shuffle of sequences of ElGamal pairs [173] and zero-knowledge proofs for general statements about discrete logarithms [38]. We used ElGamal encryption on the Ed25519 elliptic curve [22] with 128 bit security. More specifically, our prototype implements all the sub-protocols described in Section 3.3. In all of these sub-protocols, we assume the existence of a distributed ledger for which the implementation is future work. This means that the proofs of correctness are stored in global variables and the query logging and equality checking are not implemented yet. The communication between different participants relies on TCP. Finally, in order to allow for an easy deployment of UnLynx in different environments, we implemented an application that automatically handles the creation of the CA on multiple computing nodes and provides queriers with an easy way to query the system.

3.6.3 System Evaluation

We used Mininet [160] to simulate a realistic virtual network between computing nodes. Each CA computing node ran on a separate machine and was connected to the others by a 1Gbps link with a communication delay of 10ms. For each of our computing nodes, we used machines with two Intel Xeon E5-2680 v3 CPUs with a 2.5GHz frequency, 256GB RAM that supports 24 threads on 12 cores. In our performance evaluation, we study the execution time of Steps 3 to 7 presented in Section 3.3. We do not include the time needed to initialize the system (Step 0) or for the data providers to encrypt their data since these operations are done once and offline. The time needed for the querier to build the query (Step 1), for the DPs to send their responses (Step 2), and for the querier to decrypt the results (Step 8) are also left out. For Step 1 and Step 8, the runtime is negligible in comparison to the whole process, whereas the time needed for Step 2 depends almost entirely on the communication links between the computing nodes and the DPs. In the following, we describe the default parameters used in our experiments and we observe the influence of each parameter on the overall system separately.

Parameter	Default Value
# of computing nodes ($\|CN\|$)	3
# of responses in tot.	15,000
# of filtering attributes	2
# of possible groups	10
# of aggregating attributes	10
# of noise values	1,000

Table 3.1 – Default parameter values used for the evaluation of UNLYNX.

We simulated distributed computations on 15,000 responses, evenly distributed among 3 computing nodes. A response was considered to contain 2 filtering attributes, e.g., one in the `WHERE` clause and one in the `GROUP BY` statement, and 10 aggregating attributes. We assume 10 different groups, i.e., `GROUP BY` attributes can form up to 10 different group combinations. For example, patients selected on a specific disease (`WHERE`) can be grouped based on gender and on 5 age categories, each selected patient therefore belongs to 1 out of 10 possible groups. We chose to use a list of 1,000 obfuscating noise values. The default parameters are summarized in Table 3.1. In the following graphs, each measurement is averaged over 10 independent runs.

We begin our evaluation by showing how our decentralized data-sharing protocol is collectively executed by three computing nodes (CN_1, CN_2, CN_3) with the default parameters described above. The results are shown in Figure 3.4.

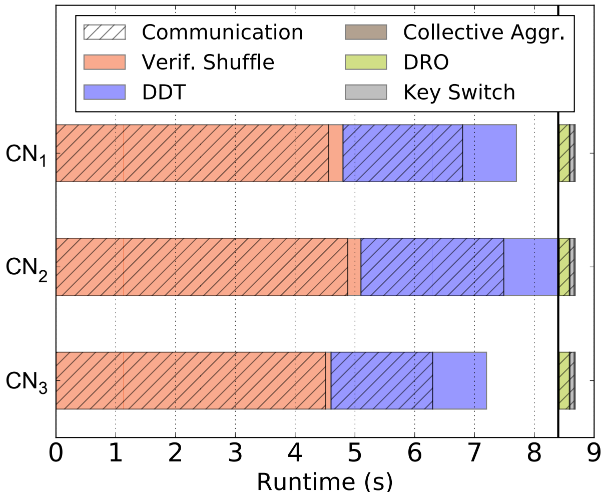


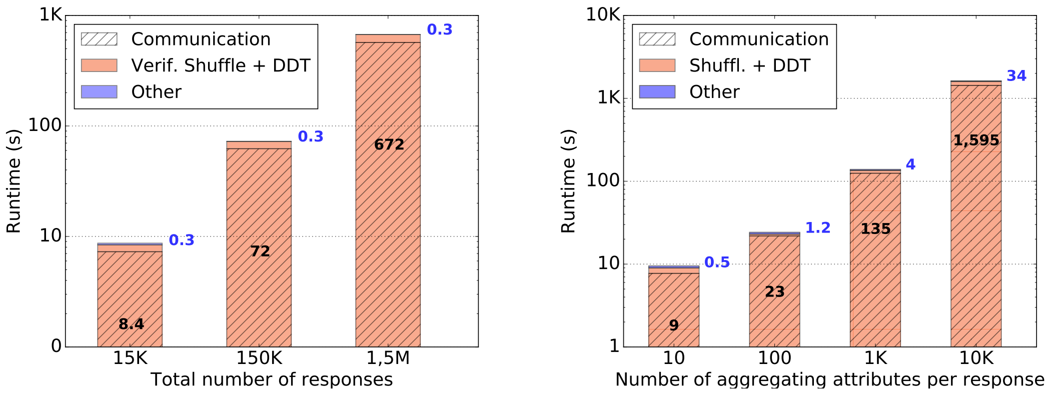
Figure 3.4 – Runtime for the different computing nodes (CN_1, CN_2, CN_3) in the CA.

Recall, each computing node has to run the verifiable shuffle and distributed deterministic-tag sub-protocols on the data received from their DPs. A computing node can run these two sub-protocols sequentially, without having to synchronize with the others. Nevertheless, they are still required to participate in the sub-protocols when requested. As a consequence, both sub-protocols can be executed in parallel by the CA, hence we can efficiently distribute the workload among the computing nodes. Once these first two steps are finished, the computing node responsible for initially processing the query begins a collective tree aggregation. Finally, the same computing node sequentially executes the distributed results obfuscation sub-protocol and then the key switch sub-protocol.

By further analyzing the graph, we see that the first two sub-protocols are the most time consuming, as they are required to process all the DPs' responses. In contrast, the last two sub-protocols are executed significantly faster, as they are only required to process aggregate responses. The DRO execution time is constant for a given number of computing nodes.

Using these observations, we often group the two first sub-protocols - "Ver. Shuffle + DDT" - and the three last sub-protocols - "Other" - in our experimental results. Moreover, we always separate the runtime of the sub-protocols and their respective proofs, as the proofs can be verified offline. At runtime, computing nodes are required to save all the information needed (e.g, the ciphertexts and the public values derived from the secret/ephemeral keys used) to create the proofs in order to be able to generate them when requested.

Finally, we observe that the communication is the most time consuming process and accounts for 75% of the overall execution time. We now study the scalability of UnLynx against different parameters.



(a) Runtime vs. total number of responses.

(b) Proof creation and verification runtime vs. total number of responses.

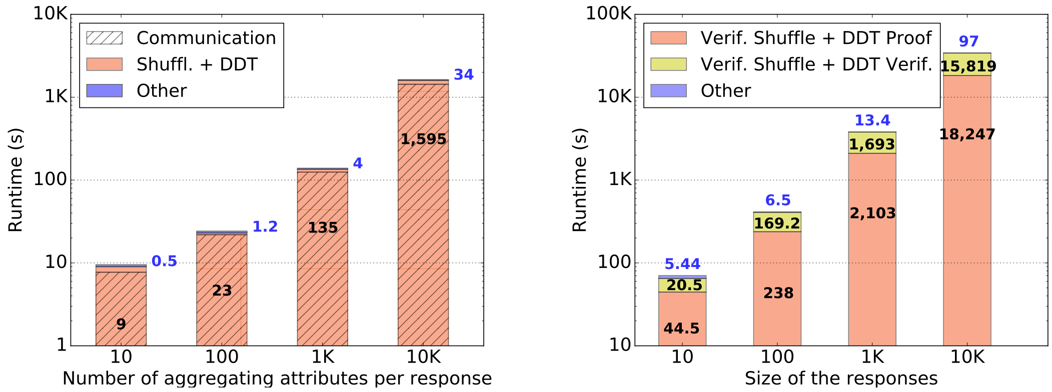
Figure 3.5 – Performance evaluation of UnLynx (using the default parameters presented in Table 3.1) with an increasing number of responses. (a) does not include proof creation or verification. In (b) *all* proofs from *all* computing nodes are computed and verified.

Varying the number of responses. To show that UnLynx scales almost linearly with the total number of responses, we begin by increasing the number of responses processed by each computing node in the collective authority.

Figure 3.5b shows the time it takes for *all* computing nodes to produce *all* zero-knowledge

proofs, as well as the time needed for an entity[2] to verify these proofs. We do not consider the time it takes for the computing nodes to publish their proofs and for a verifier to download the data necessary to verify them.
Results from Figure 3.5a show evidence of UnLynx's scalability. In fact, UnLynx is able to satisfy a request with 150K responses in less than 73 seconds. However, in Figure 3.5b, we observe that the zero-knowledge proofs incur a non-negligible computational overhead. For example, a query with 15K responses is answered in less than 9 seconds when it does not include proofs, whereas the complete execution takes almost 64 seconds. This represents an expansion factor of 7.

Varying the response size. We study the runtime of UnLynx against the number of attributes in each response. We consider that half of the attributes in a response are filtering attributes. Figure 3.6a shows that the runtime increases almost linearly with respect to the size of the responses, and the proofs bear a non-negligible overhead as shown in Figure 3.6b.



(a) Runtime vs. size of the responses (half of each response is filtering attributes and other half is aggregating attributes).

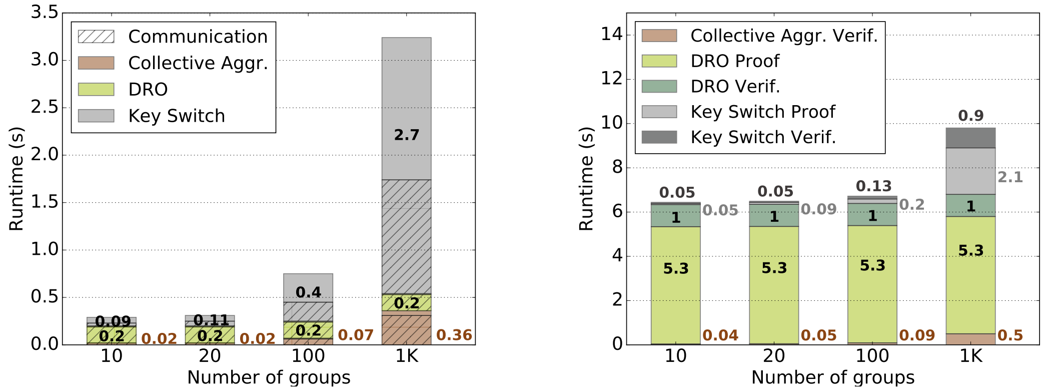
(b) Proof creation and verification runtime vs. responses' size.

Figure 3.6 – Performance evaluation of UnLynx (using the default parameters presented in Table 3.1) with an increasing size of responses. (a) does not include proof creation or verification. In (b) *all* proofs from *all* computing nodes are computed and verified.

Varying the number of groups. We observe UnLynx's runtime when increasing the number of possible groups or, in other words, the number of different combinations of `GROUP BY` attributes. This is plotted in Figures 3.7a and 3.7b. As the number of responses is constant, the execution time for the verifiable shuffle and DDT (Steps 3 and 4) are constant and, combined, they take less than 9 seconds on average. The proof creation and verification also remain constant at 29 and 15 seconds, respectively. The runtime, both with and without proofs, increases with the number of groups but remains within acceptable boundaries. For example, with 1,000 possible groups, the execution time without proofs takes approximately 9+3.3 = 12.3 seconds.

---

[2] We assume that this entity has the same computing power as one of the computing nodes.



(a) Runtime vs. number of groups.

(b) Proof creation and verification runtime vs. number of groups.

Figure 3.7 – Performance evaluation of UnLynx (using the default parameters presented in Table 3.1) with an increasing number of groups. (a) does not include proof creation or verification. In (b) *all* proofs from *all* computing nodes are computed and verified.

Varying the number of computing nodes. We assess the effect of the size of the CA on UnLynx's runtime. Figure 3.8 shows that the latency increases slightly with an increasing number of computing nodes. This is explained by the fact that the workload and data are distributed among a larger number of computing nodes, thus improving the parallelization of UnLynx. Nevertheless, adding a computing node increases the number of steps needed to complete each of the sub-protocols, which hinders the positive effect of improved parallelization.

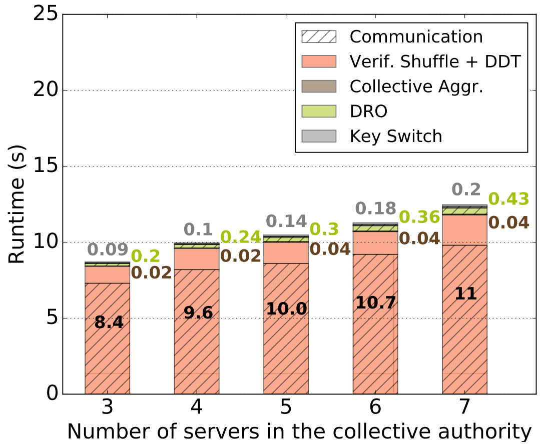


Figure 3.8 – Runtime for a varying number of computing nodes in the CA.

3.6.3.1 Storage Overhead

We now discuss the storage overhead induced by ElGamal encryption. We recall that our encryption relies on an elliptic curve with a security level of 128 bits and that each encrypted message is a pair of points on the curve. Each point is encoded with 32 bytes, hence each encrypted message is 64 bytes long. Therefore, the encryption of an integer (4 bytes) in clear text yields an expansion factor of $64/4 = 16$.

For example, assume each DP's database contains 10,000 lines and 120 columns. Each line contains data belonging to one individual, and the columns correspond to attributes. Here, the amount of data stored by each DP is 4.8MB if it is stored in clear, and 73.25MB if it is encrypted. Finally, storage overhead on each CA computing node is negligible, as data can be temporarily stored and discarded after locally aggregating the data.

3.6.3.2 Communication Overhead

As shown by our performance evaluation, most of the sub-protocol's execution time is dedicated to communication. We use the same example as in the previous section and assume that a query requests 120 columns, half of which are filtering attributes. In this case, each DP sends 73.25MB to the computing nodes. Considering six DPs, this results in 439.5MB of data to be processed by the CA. The communication overhead for each sub-protocol is given below.

Verifiable Shuffle. All the data have to be sent through all three computing nodes, resulting in a communication overhead of $439.5 \times 3 = 1315.5$MB.

Distributed Deterministic Tag. In this sub-protocol, only the attributes in the `WHERE` clause and in the `GROUP BY` statement need to be sent, which totals to $\frac{439.5}{2} \times 3 = 659.25$MB.

Collective Aggregation. Before executing this sub-protocol, the responses are locally filtered and aggregated by each computing node. With 10 possible groups, we obtain a communication overhead of only 0.14MB for when all the filtering attributes are in the `GROUP BY` statement (worst case scenario). We recall that `WHERE` clause attributes can be discarded after the responses have been filtered.

Distributed Results Obfuscation. The amount of traffic for this sub-protocol depends on the number of computing nodes and the size of the noise value list that, in this case, is 1,000 clear text integers. Therefore, for this particular example, we send around 0.004MB $\times 3 = 0.012$MB of data.

Key Switch. The communication overhead is 0.2MB.

Finally, in order to privately process 439.5MB of data, UnLynx needs to send $1315.5 + 659.25 + 0.14 + 0.012 + 0.2 = 1975.1$MB, which is 4.5 times the original amount of data.

In Figure 3.9, we observe the influence of the bandwidth capacity and communication delay between the computing nodes. By using the default parameters presented in Table 3.1 and with 1Gbps links, the maximum communication rate observed was 80Mbps. We show, for increasing latency, the complete runtime of UnLynx when the communication rate is not limited (80Mbps) and when it is reduced to 40Mbps and 20Mbps. As expected, the computation time is constant, around 1.3 seconds, and the communication time increases with both the bandwidth and the transmission delay. We observe that when the delay increases, reducing the bandwidth from 80Mbps to 40Mbps does not have a significant effect on the overall runtime. This occurs because when the delay increases the maximum communication rate also decreases.

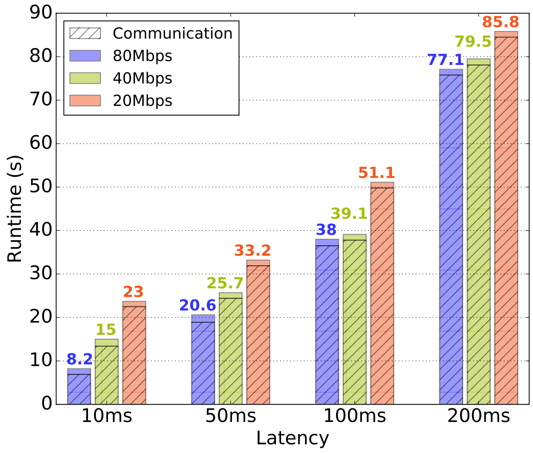


Figure 3.9 – Runtime for different values of bandwidth capacity and latency for the links between the computing nodes.

3.6.3.3 Dynamic Collective Authority

We observe the latency incurred by adding/removing a computing node to/from the CA. We do not include the time to transfer the data between DPs and computing nodes. We assume that the computing node leaving/joining the CA is willing to participate in the process. The results are shown in Figure 3.10 and depict an almost linear increase in runtime with the total number of ciphertexts collectively held by the DPs.

3.7 Example Application: Secure Survey

To illustrate that UnLynx is usable in practice, we present a realistic use case of a secure distributed survey. We also further assess our system's performance and study the possible tradeoffs that can be made in order to improve UnLynx's response time.

We show that our system could improve and simplify the process of carrying out a survey on sensitive personal data. In fact, such surveys are usually done on data that are anonymized,

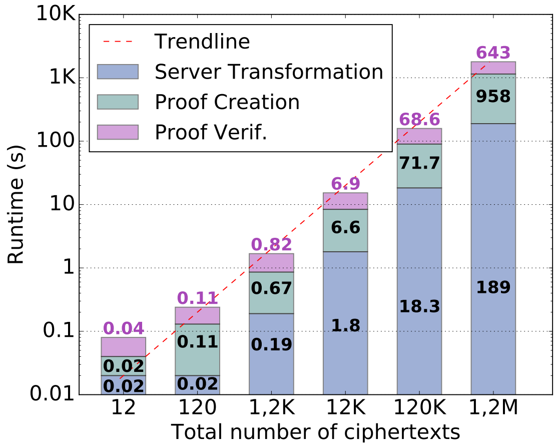


Figure 3.10 – Runtime for adding/removing a computing node to/from the CA.

hence reducing the precision of the results. Moreover, and as demonstrated by D. Bogdanov et al. [25], obtaining the permission to access such sensitive data is administratively heavy, requires the participation of multiple data-protection entities (local government, European Union, ...) and is extremely time and money consuming. UnLynx enables us to do a secure and privacy-preserving survey on data that are encrypted at rest and during computations, and that remain under the DPs’ control throughout the process. This can tremendously facilitate the access to data and the achievement of such surveys.

We consider a realistic example where a statistical institute wants to compute the average salary of the top 20 biotechnology companies in the United States. The query is ‘`SELECT AVG(`$salaries$`) FROM` $C_1,\ldots,C_{20}$ `WHERE` $age\ in\ [40:50]$ `AND` $ethnicity = Caucasian$ `GROUP BY` $gender$’. A `SELECT AVG(`$*$`)` query is executed by combining the results of the ‘`SELECT SUM(`$salaries$`), COUNT(`$*$`)`’ query on the same filtering attributes. In our example, the DPs are the companies, the querier is the statistical institute and the collective authority corresponds to three computing nodes handled by the statistical institute, a consortium of the companies and the US government (who wants to ensure that data are protected). We assume that each company has 20K employees.

In this scenario, which we refer to as our baseline, all data are encrypted and all the zero-knowledge proofs are created and verified. Then, we discuss two possible tradeoffs that can be used in UnLynx and compare their performance.

The first tradeoff is to consider all computing nodes as non-malicious, hence to not ask for proofs of correctness. This reduces the runtime by at least 90%, as shown in Figure 3.11. A verifier can also randomly ask for - and check - part of the proofs or request proofs from a single computing node. In both cases, this can be done offline and the proof-execution time decreases proportionally to the amount of omitted proofs. As computing nodes cannot anticipate which proofs will be verified, they must still compute correctly, if they do not want

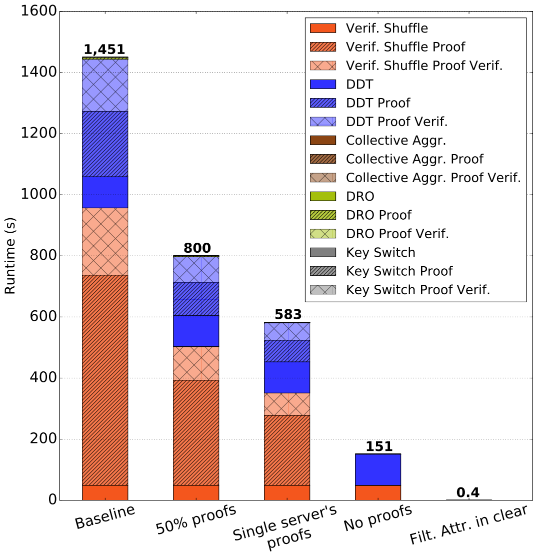


Figure 3.11 – Protocol runtime considering the tradeoffs between security/privacy and efficiency in the case of a secure census.

to be caught cheating.

The second tradeoff is for DPs to have all the filtering-attribute values stored in clear text. This enables computing nodes to locally aggregate DPs' responses for each combination of filtering attributes, thus reducing the number of responses to be processed from the number of records sent by the DPs to the number of different filtering attribute combinations. The latter usually being considerably smaller than the former.

In Figure 3.11, we can verify that each of these tradeoffs significantly enhances performance. Producing and verifying 50% of the proofs at each computing node reduces the execution time by 45%, whereas having all DPs send clear text filtering attributes reduces it by 99%.

Considering the results shown in Figure 3.11, it becomes obvious that, in order to control the computation and storage overhead, the categorization of attributes as either sensitive or non-sensitive is key in deciding what needs to be encrypted. We suggest the following guidelines for an efficient and privacy-preserving solution.
Non-sensitive attributes (e.g., age, gender or ethnicity). Stored in clear and protected by privacy-protection techniques yielding, for example, k-anonymity [229].
Sensitive attributes (e.g., salary). Store encrypted under the CA's public key K.
We provide an example database, Table 3.2, that respects these guidelines where $\mathrm{E}_K(x)$ refers to the ElGamal encryption of message x.

Finally, we argue that a response time of 24 minutes, for a secure distributed and privacy-preserving survey on 400,000 records, is acceptable. We recall that this response time does

ID	Gen.	Age	Ethnicity	...	Salary
P1	F	$40:50$	Caucasian	...	$\mathrm{E}_K(100,000)$
P2	M	$40:50$	Caucasian	...	$\mathrm{E}_K(10,500)$
P3	M	$30:40$	Asian	...	$\mathrm{E}_K(10,000)$
P4	F	$30:40$	Asian	...	$\mathrm{E}_K(100,500)$

Table 3.2 – Proposed database structure. In this specific example, gender, age and ethnicity are non-sensitive filtering attributes and are therefore left in clear. To reduce the risk of identity disclosure, the table values are generalized to satisfy k-anonymity with $k = 2$ for the quasi-identifiers age, gender and ethnicity. Salary is a sensitive attribute and is therefore encrypted under the CA's public key K

not include Steps 0, 1, 2 and 8, as explained in Section 3.6. In this specific case, if we assume that (1) DPs machines have the same settings as the computing nodes described in Section 3.6, (2) data are encrypted beforehand and (3) DPs respond all at the same time, then the time to transmit these data (Step 2) is around 0.4 seconds. This time depends exclusively on the communication link between the computing nodes and DPs and on the amount of data to be sent.

In conclusion, if we assume that the filtering attributes, namely age, gender and ethnicity are stored in clear, protected by anonymization techniques and that the proofs of correctness are not executed, then UNLYNX's response time is reduced to 0.4 seconds.

3.8 Conclusion

UNLYNX is a modular decentralized system for privacy-preserving data sharing among multiple data providers. Specifically, we enable a querier to obtain aggregate statistics for different grouping criteria on a set of different databases. We achieve this through protocols that enable a number of independent computing nodes to compute on distributed data sets and that provide proofs of correctness of their work. We further build upon advances in several areas, such as zero-knowledge proofs and verifiable shuffling, and we bring them all together into UNLYNX, providing security and privacy guarantees against malicious behavior. Additionally, by introducing a new protocol for distributed obfuscation of results, UNLYNX ensures (ϵ, δ)-differential privacy for individuals sharing their data. The performance evaluation of our prototype shows that it is efficient and almost linearly scalable with the amount of data to be processed. We provide a realistic use case of a secure distributed survey.

4 Federated Statistical Analysis

4.1 Introduction

Improving upon and using some techniques introduced in Chapter 3 (i.e., UNLYNX), we propose DRYNX, an operational, decentralized and secure system that enables queriers to compute statistical functions and to train and evaluate machine-learning models on data hosted at different sources, i.e., on distributed datasets. DRYNX ensures data confidentiality, data providers' (DPs) privacy and protects individuals' data from potential inferences stemming from the release of end results, i.e., it ensures differential privacy. It also provides computation correctness. Finally, it ensures that strong outliers, either maliciously or erroneously input by DPs, cannot influence the results beyond a certain limit, and we denote this by *results robustness*. DRYNX enables full and lightweight auditability of query execution by relying on a new efficient distributed solution for storing and probabilistically verifying proofs. These guarantees are ensured in a strong adversarial model where no entity has to be individually trusted and a fraction of the system's entities can be malicious. DRYNX relies on interactive protocols, homomorphic encryption, zero-knowledge proofs of correctness and distributed differential privacy. It is scalable, dynamic and modular: Any entity can leave or join the system at any time and DRYNX offers security features or properties that can be enforced depending on the application, e.g., differential privacy.

4.1.1 Use Cases

Throughout this chapter, we illustrate DRYNX's utility in the medical sector, as it is a paradigmatic example where privacy is paramount and data sharing is needed. Recently, multiple initiatives have emerged to realize the promise of personalized medicine and to address the challenges posed by the increasing digitalization of medical data [97; 216; 227]. In this context, the ability to share highly sensitive medical data while protecting patients' privacy is becoming of primary importance. We illustrate the possible use of DRYNX in two specific settings that cover most medical data sharing scenarios: (1) Hospital Data Sharing (*HDS*), where multiple hospitals enable statistical computations and the training of machine-learning models across their datasets of patients (e.g., [108; 227]), and (2) Personal Data Sharing (*PDS*), where a medical institute runs studies, e.g., on heart issues, by directly computing on data collected from people's wearables (e.g., [11; 184]).

In this chapter, we make the following contributions:

- We propose DRYNX, an efficient, modular and parallel system that enables privacy-preserving statistical queries and the training and evaluation of machine-learning regression models on distributed datasets.
- We present a system that provides data confidentiality and individuals' privacy, even in the presence of a strong adversary. It ensures the correctness of the computations, protects data providers' privacy and guarantees robustness of query results.
- We propose techniques that enable full and lightweight auditability of query execution. DRYNX relies on a new efficient distributed solution for storing and verifying proofs of query validity, computation correctness, and input data ranges. We exemplify and evaluate the implementation of this solution by using a blockchain.
- We propose and implement an efficient, modular and multi-functionality query-execution pipeline by
  - introducing *Collective Tree Obfuscation*, a new distributed protocol that enables a collective and verifiable obfuscation of encrypted data;
  - presenting multiple data-encoding techniques that enable distributed computations of advanced statistics on homomorphically encrypted data. We propose new encodings, and improvements and adaptations of previously introduced private-aggregation encodings to our framework and security model;
  - adapting an existing zero-knowledge scheme for input-range validation to our security model;
  - proposing a new construction of the *key switch* protocol (see Section 3.3.2.4) introduced in Chapter 3, improving both its performance and capabilities.

To the best of our knowledge, DRYNX is the only operational system that provides the aforementioned security and privacy guarantees. DRYNX implementation is fully available at www.github.com/ldsec/drynx.

Acknowledgements. This work is the result of a collaboration with Dr. Juan R. Troncoso-Pastoriza, Joao Sa Sousa, and Prof. Jean-Pierre Hubaux. We also thank Dr. Henry Corrigan-Gibbs for his valuable feedback and the members of the DeDiS lab for their help with software engineering.

4.2 System Overview

In this section, we describe the system and threat models, before presenting Drynx's functionality and security requirements.

4.2.1 System Model

The system model is represented in Figure 4.1. For simplicity, we describe here the logical roles in Drynx, and in Section 4.6 we discuss the fact that a physical node can simultaneously play multiple roles. A querier Q can execute a statistical query and the training and evaluation of a machine-learning model on distributed datasets held by DPs. The CNs collectively handle the computations in the system; i.e., from Q's perspective, they emulate a central server and provide answers to her queries. The verifying nodes' (VNs) role is to provide auditability; they collectively verify the query execution and immutably store the corresponding proofs. They enable an auditor, e.g., Q or an external entity, to easily verify (audit) the correctness of the query execution.

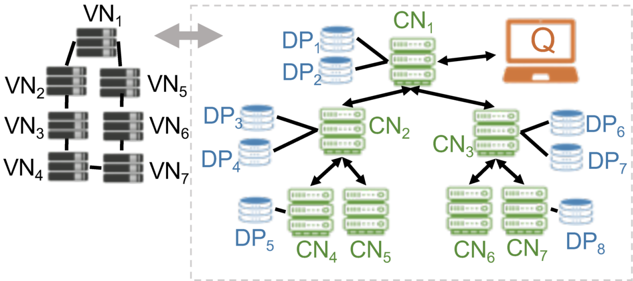


Figure 4.1 – A querier Q, Data Providers DP_i, Computing Nodes CN_i and Verifying Nodes VN_i.

In Drynx's typical workflow, the query is defined by the querier Q and is then broadcast to the CNs and DPs. The DPs answer with their encrypted responses that are then collectively aggregated and processed by the CNs, before the result is sent to Q. As in Chapter 3, we assume that the used data formats are sufficiently homogeneous among different DPs and that the DPs are able to interpret the queries, e.g., there is a common ontology of attributes and the query-language is agreed-on during system setup.

An exemplifying instantiation of this system model in the *HDS* scenario (Section 4.1.1) would feature the CNs as universities that want to enable researchers (Q) to compute on data held by multiple hospitals (DPs). VNs can be independent or governmental institutions ensuring that data protection regulations are respected.

We assume that the system's topology and public information, e.g., public keys, are known by all entities. Authentication and authorization are out of scope of this chapter and we briefly discuss them in Section 4.6.

4.2.2 Threat Model

We assume a stronger threat model than in Chapter 3 as the DPs can be malicious, when we assume them to be honest-but-curious in Chapter 3. More specifically:

- *Queriers.* They are considered malicious as they can try to infer information about the DPs from the queries end results or by colluding with other entities in the system.
- *Computing Nodes.* We consider an Anytrust model [246], which means that all DRYNX's security and privacy guarantees (Section 4.2.4) are ensured, as long as at least one of the CNs is honest-but-curious (or plain honest).
- *Data Providers.* The DPs are considered malicious as they can try to produce an incorrect answer to a query in order to bias the final results. They can also collude with other nodes to infer information about other DPs or about a query end-results.
- *Verifying Nodes.* We assume that a threshold number of the VNs is honest. This threshold, e.g., $f_h = 2f + 1$ out of $f_t = 3f + 1$, where f_t is the number of VNs, is defined depending on the consensus algorithm [40; 256] that is used to ensure a correct and immutable storage of the proofs' verification results.

4.2.3 Functional Requirements

DRYNX enables the computation on distributed datasets of any operation in the family of *encodable operations*. An *encodable operation* can be separated in two parts: the DPs' local computations and the collective aggregation. In the collective part, the computations are executed on encrypted data and are thus limited by the homomorphism in the used cryptographic scheme, e.g., additions and/or multiplications. DPs' computations are executed locally and are therefore not limited.

Definition 1. *An* encodable operation f *computed among* $|S|$ *DPs is defined by:*

$$f(\bar{r}) \equiv \pi(\{\rho(\bar{r}_i)\}_{i=1}^{|S|}),$$

in which the encoding ρ *is defined by*

$$\rho(\bar{r}_i) \equiv (\mathbf{V_i}, c_i),$$

where $\mathbf{V_i} = [v_{i,1}, ..., v_{i,d}]$ *is a vector of* d *values computed on a set of* $c_i = |\bar{r}_i|$ *records, where* $|.|$ *stands for cardinality.* $\bar{r}$ *is the set of all distributed datasets' records,* $\bar{r}_i$ *is the set of records that belong to* DP_i*, and* π *is a polynomial combination of the outputs of the encodings* ρ*. The encodings are defined as locally computed functions on the subsets* ($\bar{r}_i$) *of each* DP_i*. It is also possible to express an encodable operation as a recursive function:*

$$f_k(\bar{r}) \equiv \pi(\{\rho(\bar{r}_i, f_{k-1}(\bar{r}))\}_{i=1}^{|S|}).$$

In DRYNX, for any specific operation f, each DP_i creates an encoding ρ computed on its set of records $\bar{r}_i$. Then, π is executed in two parts: the CNs first aggregate all DPs' encodings outputs ($\sum_{i=1}^{|S|}\{\rho(\bar{r}_i)\}$) and, if needed, the querier post-processes π on the aggregated result (e.g, if π involves information-preserving operations not executable by the CNs under homomorphic encryption).

We give here an instantiation of Definition 1 that enables the computation of the `average`, and in Section 4.5 we show how an *encoding* can be instantiated to enable the computation of: `sum`, `count`, `frequency count`, `average`, `variance`, `standard deviation`, `cosine similarity`, `min/max`, `AND/OR` and `set intersection/union`, and the training and evaluation of `linear` and `logistic regression` models.

For example, if Q wants to compute the `average` (f) heart rate over multiple patients across hospitals (HDS (Section 4.1.1)), each hospital (DP_i) answers with the encoding of its (encrypted) local sum of each patient's heart rate (h): $\rho(\bar{r}_i) \equiv ([\sum_{j=1}^{c_i} h_{i,j}], c_i)$. These encodings are then (homomorphically) added across all hospitals, and Q can (decrypt and) compute the global average by using $\pi = \sum_{i=1}^{|S|} v_{i,1} / \sum_{i=1}^{|S|} c_i$. We remark here that whereas ρ and π are application dependent, the workflow is common to all the possible operations.

Finally, in DRYNX, an auditor can efficiently audit a query execution. Moreover, the proofs required for auditability are produced such that their creation does not affect the query runtime.

4.2.4 Security Requirements

DRYNX must ensure:

- *Data confidentiality.* The data input by the DPs have to remain confidential at any time. Only Q is able to see the query answer.
- *DPs' privacy.* No entity is able to infer information about one single DP or about any individual storing his data in a DP's database.
- *Query Execution Correctness.* We consider the query execution to be correct when both *results robustness* and *computation correctness* requirements are met:
    - *Results robustness.* The query results are protected against strong outliers, either maliciously or erroneously input by the DPs.
    - *Computation correctness.* Any computation undertaken by the CNs is correctly executed.

4.3 Drynx Design

To overcome the limitations in existing works and meet the requirements presented in the previous section, we propose a novel system model in which we enable query auditability by introducing VNs. Additionally, DRYNX provides multiple functionalities in a stronger threat model by relying on DPs that encode locally computed results proven to be within a certain range before aggregating their results under additive homomorphic encryption. It limits the trust in DPs by controlling that their results are in these pre-defined ranges. We propose a system that remains generic and practical while operating in a threat model stronger than existing works. We discuss now the design of this system.

In DRYNX*'s Security Design* (Section 4.3.1), we show how we build DRYNX to meet all its security requirements:

- In Section 4.3.1.1, we introduce a simple query-execution pipeline enabling DRYNX's functionalities and protecting data confidentiality.
- In Section 4.3.1.2, we build upon the previously introduced query-execution pipeline and explain how to ensure DPs' privacy by introducing the new concept of a *neutral encoding*. This enables a DP to privately choose whether to answer a query. We also explain how DRYNX handles bit-wise operations and maintains DPs' privacy. Finally, we introduce distributed differential privacy that is used to ensure that no entity infers information about a single DP or individual from the query end results.
- In Section 4.3.1.3, we show how we provide auditability in an efficient way by relying on a set of VNs. We describe how DRYNX ensures results robustness by leveraging on range proofs and how all DRYNX's computations can be verified by relying on proofs of correctness.

In DRYNX*'s Optimized Design* (Section 4.3.2), we discuss how to optimize DRYNX's performance:

- In Section 4.3.2.1, we present DRYNX's full query-execution pipeline. We show how multiple parts of the query execution and verification can be run concurrently thus optimize DRYNX's runtime.
- In Section 4.3.2.2, we introduce a tradeoff between security and performance by enabling a probabilistic verification of the query execution.

4.3.1 Drynx Security Design

We present DRYNX core security architecture.

4.3.1.1 Data Confidentiality

First, we introduce a confidential distributed data-sharing system (Figure 4.2) that can run the same operations as DRYNX, but only meets one of the security requirements: *data confidentiality*.

We describe the query execution protocol, and sketch the proof of confidentiality for this system. Afterwards, we describe how to enhance this construction to meet DRYNX's other security requirements without breaking data confidentiality.

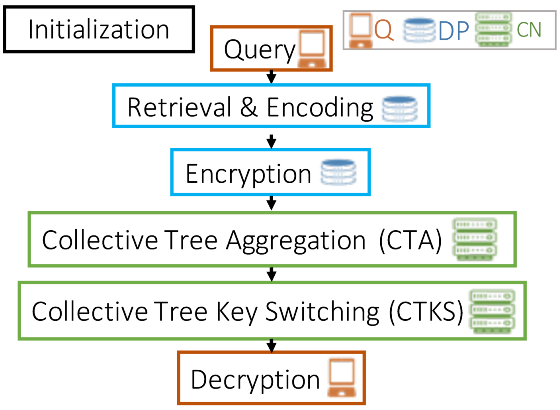

Initialization
Query
Q
DP
CN
Retrieval & Encoding
Encryption
Collective Tree Aggregation (CTA)
Collective Tree Key Switching (CTKS)
Decryption

Figure 4.2 – Confidential System Query Execution.

1. *Initialization.* Each CN_i, DP_i and Q_i generates its own private-public key-pair (k_i, K_i). The CNs' public keys are then summed up in order to create K, the CNs' public collective key that is used to encrypt all the processed data.

2. *Query.* Q formulates the query that is broadcast in clear through the CNs to the DPs. Although the querier could directly communicate with the DPs, our choice simplifies the communication scheme and the synchronisation inside the system, as the CNs have to know the query and receive the DPs inputs to perform the computations in the remaining steps. The query defines the operation, the attributes on which the operation is computed, the participating DPs and (optionally) the filtering conditions. DRYNX works independently of the query language. We illustrate its use with a SQL-like query to compute the `average` heart rate among patients for which data are held by $|S|$ DPs:

   `SELECT average` $heart\_rate$ `ON` DP_1, ..., $DP_{|S|}$
   `WHERE` $patient\_state$ = $'hypertensive'$

3. *Retrieval & Encoding.* The DPs compute their local answer by following ρ which is defined in the operation *encoding* (Definition 1). For this purpose, they first locally retrieve the corresponding data.

4. *Encryption.* The DPs encrypt their encoded answer under K and send the corresponding ciphertexts back to the CNs.

5. *Collective Tree Aggregation (CTA).* The CNs collectively aggregate, under additive homomorphic encryption, all DPs' responses by executing a CTA protocol relying on the *Collective Tree Aggregation* protocol defined in Chapter 3 (Section 3.3.2.2). The CNs are

organized into a tree structure such that each CN waits to receive the aggregation results from its children and sums them up before passing the result on to its own parent.

6. *Collective Tree Key Switching ($CTKS$).* The CNs collectively convert the aggregated result, encrypted under K, to the same result encrypted under Q's public key K', without ever decrypting. This protocol (Protocol 4.1) is a new construction of the *Key Switching* proposed in Chapter 3. Conceptually, each CN partially decrypts m (i.e., the term $-(C_1)k_i$ in the computation in step 2) and re-encrypts it with Q's public key K' (i.e., the term $+\alpha_i K'$ in step 2).

Protocol 4.1 Collective Tree Key Switching (CTKS)

Input. $\mathrm{E}_K(m) = (C_1,\ C_2) = (rB, mB + rK),\ K'$
Output. $\mathrm{E}_{K'}(m) = (C'_1, C'_2) = (r'B, mB + r'K')$

Protocol.

1. The root CN_1 sends C_1 down the tree to all CNs.
2. Each CN_i generates a secret uniformly-random nonce α_i and computes $w_{i,1} = \alpha_i B$ and $w_{i,2} = -(C_1)k_i + \alpha_i K'$
3. The CNs collectively aggregate (i.e., using CTA) all the $w_{i,1}$ and $w_{i,2}$.
4. CN_1 finally computes $(C'_1, C'_2) = (\sum w_{i,1}, C_2 + \sum w_{i,2}) = (r'B, mB + r'K')$ where $r' = \sum \alpha_i$.

In Protocol 4.1, we improve the efficiency of the corresponding protocol introduced in Chapter 3 (Section 3.3.2.4) by changing the way the ciphertexts are transformed and by organizing the CNs in a tree structure, thus reducing its execution time. In this structure, multiple CNs can perform their local operations (3 scalar multiplications and 1 addition) in parallel, and the CTA requires $\#CN - 1$ aggregations and communications between the nodes. We show the computational complexity of all DRYNX protocols in Table 4.2.

7. *Decryption.* Q decrypts and decodes the query results.

Security Arguments. We show that, as long as one CN is honest, an adversary who controls the remaining CNs, DPs and Q cannot break data confidentiality. Without loss of generality, we assume that at least one DP is honest, as only in this case there is data to protect from the adversary. We sketch the proof by relying on the real/ideal simulation paradigm [139] and show that an adversary cannot distinguish a "real" world experiment, in which the adversary is given "real" data (sent by honest DPs), and an "ideal" world experiment, in which the adversary is given data (e.g., random) generated by a simulator. It can be shown that the DPs send encrypted data that are never decrypted before being aggregated and re-encrypted ($CTKS$) under Q's public key. Therefore, due to the cryptosystem's semantic security, the adversary cannot distinguish between a simulation and a real experiment. It can be seen that data confidentiality is thus ensured during end-to-end query execution:

In *Retrieval & Encoding*, the DPs operate only on their local data and no external data is seen by any malicious party. In *Encryption*, the DPs encrypt their responses with K and these responses are aggregated, still under encryption, in CTA. The (summed) ciphertexts cannot be decrypted unless all CNs collude, which is not possible as they follow an Anytrust model. Finally, in $CTKS$ (Protocol 4.1), a ciphertext is switched from K to Q's public key such that Q can decrypt:

- in *$CTKS$ Steps: 1-3.* The ciphertext is encrypted under K and thus cannot be decrypted without the collusion of all CNs.
- in *$CTKS$ Step: 4.* The ciphertext is always $(\tilde{C}_1, \tilde{C}_2) = (\tilde{r}B, mB + \tilde{r}K')$ where $\tilde{r} = \sum_{i=0}^{t} \alpha_i$ and $0 \leq t \leq \#CN$ and can only be decrypted if the t CNs collude with Q, who is the intended recipient of the message.

4.3.1.2 DPs' Privacy

As mentioned before, we now build on top of the query-execution pipeline (see Figure 4.2) proposed in the previous section. In this section, we explain how to ensure DPs' privacy. DRYNX complete query-execution is shown later in Figure 4.4. DRYNX protects DPs' and individuals' privacy by ensuring that (a) each DP can privately decide whether to answer a query, (b) only the result of the operation, as defined by the operation *encoding*, is disclosed to Q, and (c) no entity can infer information about a single DP or individual.

Neutral Response. If a DP determines that a query can jeopardize its privacy, it can choose to not respond, or answer with a neutral response, thus hiding its refusal to participate in the query without distorting the query results. For this purpose we define *neutral response*:

Definition 2. *A DP_i sends a* neutral response *by defining its response encoding (Definition 1) by $\rho(\bar{r}_i) \equiv (\mathbf{O}, 0)$, where $\mathbf{O}$ is the neutral vector such that $\mathbf{W} + \mathbf{O} = \mathbf{W}$ with $\mathbf{W}$ being any encoding vector; $c_i = 0$ as DP_i computes on 0 records.*

In Section 4.5, we describe how a *neutral response* can be generated for each listed *encoding*.

Security Arguments. A DP not answering a query would suggest (leak) to other entities that this query is too sensitive for it. DPs' responses are always encrypted and, due to the indistinguishability property of the underlying cryptosystem, a *neutral response* is indistinguishable from a non-neutral one, thus effectively hiding the DP's refusal.

Privacy-Preserving Bit-wise Operations. In DRYNX, DPs' responses are summed through the available additive homomorphism; if these responses are binary, the result of the sum can leak to Q more than the operation result. For example, when an `OR` operation is executed over a set of DPs, Q should only know if the answer is $true$ (1) or $false$ (0). Nevertheless, if the DPs' responses are naively summed, Q gets the number of DPs that answered '1' and '0'. To overcome this issue, we propose the *Collective Tree Obfuscation (CTO)* protocol, detailed in Protocol 4.2. For bit-wise operations, CTO is run between steps CTA and $CTKS$ of the query execution. In CTO, the CNs collectively obfuscate a ciphertext by multiplying it with a random secret.

CTO enables privacy-preserving bit-wise operations in DRYNX as a '1' is obfuscated to a random value whereas '0' is preserved. To know the result of the operation, Q only checks if the final value is '0' or not.

Protocol 4.2 Collective Tree Obfuscation (CTO)

Input. $\mathrm{E}_K(m) = (C_1,\ C_2) = (rB, mB + rK)$

Output. $\mathrm{E}_K(sm) = (srB, smB + srK)$

Protocol.

1. Root CN_1 sends $(C_1,\ C_2)$ down the tree to all CNs.
2. Each CN_i generates a secret uniformly random nonce s_i and computes $(\hat{C}_{i,1}, \hat{C}_{i,2}) = s_i \cdot (C_1,\ C_2)$
3. The CNs collectively aggregate (i.e., using CTA) all the $(\hat{C}_{i,1}, \hat{C}_{i,2})$.
4. CN_1 obtains $\mathrm{E}_K(sm) = s \cdot (C_1,\ C_2)$ where $s = \sum s_i$.

Security Arguments. Protocol 4.2 does not hinder the confidentiality of m and indeed obliviously and statistically obfuscates m. The confidentiality relies on the cryptosystem's semantic security, as m remains encrypted during the whole protocol execution. A multiplicative blinding of m in $\mathbb{Z}_p$ is defined by $s \cdot m$, where s is a secret scalar value in $\mathbb{Z}_p$. The output of the CTO protocol is the encryption of $(\sum s_i) \cdot m$. We can rewrite $(\sum s_i) \cdot m$ by separating the contributions of the honest CNs h (at least one CN due to our Anytrust model assumption) and malicious CNs e: $(\sum_{i \in h} s_i + \sum_{i \in e} s_i) \cdot m = (\sum_{i \in h} s_i) \cdot m + (\sum_{i \in e} s_i) \cdot m$. Even if an adversary knows $(\sum_{i \in e} s_i) \cdot m$, the other term $(\sum_{i \in h} s_i) \cdot m$ ensures a multiplicative blinding of m in $\mathbb{Z}_p$.

Distributed Differential Privacy. DRYNX relies on the Collective Differential Privacy (CDP) protocol, introduced in Chapter 3, to ensure differential privacy, and prevent information inference about some DPs and/or individuals from the query results. We briefly recall here the

---

Protocol 4.3 Collective Differential Privacy (CDP)

Input. ϵ (defined in Section 1.5), Δf: query sensitivity, and θ: quanta

Output. $\mathrm{E}_K(\hat{n}_1, ..., \hat{n}_{\tilde{l}})$

Initialization

1. The distribution $LD = Laplace(0,\ \Delta f / \epsilon)$ is publicly agreed on.
2. LD is publicly sampled, using the quanta θ, to a list of $\tilde{l}$ noise values $\tilde{n}_1, ..., \tilde{n}_{\tilde{l}}$.

Protocol.

1. Each CN privately and sequentially shuffles $\tilde{n}_1, ..., \tilde{n}_{\tilde{l}}$, producing $\mathrm{E}_K(\hat{n}_1, ..., \hat{n}_{\tilde{l}})$.
2. First elements of $\mathrm{E}_K(\hat{n}_1, ..., \hat{n}_{\tilde{l}})$ are used as oblivious noise values and added to the query result.

---

main steps of the protocol and show in Figure 4.4 where this protocol is integrated in DRYNX's complete workflow. The choice of parameters depends on the application's privacy policy and is out of the scope of this paper.

Security Arguments. As detailed in Chapter 3, we observe that the list of noise values is verifiably generated from the differential privacy parameters and that all the CNs privately shuffle the values.

4.3.1.3 Query Execution Correctness

We first describe how DRYNX provides auditability by enabling an efficient verification of the query execution correctness. The latter is achieved by guaranteeing results robustness and computation correctness. The first is ensured by limiting the DPs' values to be in a specific range (by means of range proofs) and the second by using ZKPs for all the CNs computations.

Auditability. To provide an efficient solution for the query verification, DRYNX relies on a set of VNs that verify the query correctness in parallel to its execution and without affecting its runtime. After each operation, Q, the CNs and DPs create proofs of correct computations or value range that they sign with their private key (to provide authentication). Their signed proofs are sent to all the VNs. This enables an efficient query execution as the proof creation and verification are executed independently from it.

In order to implement this solution, we can rely on the distributed architecture of the VNs and can provide integrity and immutability by using a blockchain, i.e., the *proof blockchain*. This enables the public and immutable storage of both the query and its verification results. Moreover, it enables an efficient and lightweight verification of the query correctness. An auditor, e.g., Q, has only to request the block corresponding to the query, to verify the VNs signatures and to check the query verification results. We detail this in Protocol 4.4 and show an example of the *proof blockchain* in Figure 4.3.

Protocol 4.4 Query Verification

Query

Q:

1. *Q* signs and broadcasts the query to the VNs.

VNs:

1. Each VN verifies *Q*'s signature.
2. Each VN deterministically derives the list of expected proofs for the query. It initializes a *query-proofs map* that stores the result of the verification for each proof: `true`, `false`, `not received` (before a predefined timeout).

Query Execution.

DP or CN:

1. A DP or CN executes an operation, then creates, signs and sends the corresponding proof to the VNs.

VNs:

1. Each VN verifies the prover's signature.
2. Each VN verifies the proof and stores the result in its *query-proofs map*.
3. Each VN stores the proof in its local (key, value)-database. The key is uniquely and deterministically derived from the query, the prover's ID and the proof type.

End of Query Execution (or timeout).

VNs:

1. One of the VNs (e.g., chosen in a round-robin fashion) gathers all VNs' *query-proofs maps*.
2. The same VN creates a block containing the *Query Unique ID*, the *Query* and all the *query-proofs maps*.
3. The block is sent around such that each VN checks that its *query-proofs map* and the query are correctly saved. If this is the case, the VN signs the block.
4. The VNs run a consensus algorithm such that a block signed by a threshold f_h of VNs is consistently added to the blockchain. Each VN keeps a local copy of the blockchain.

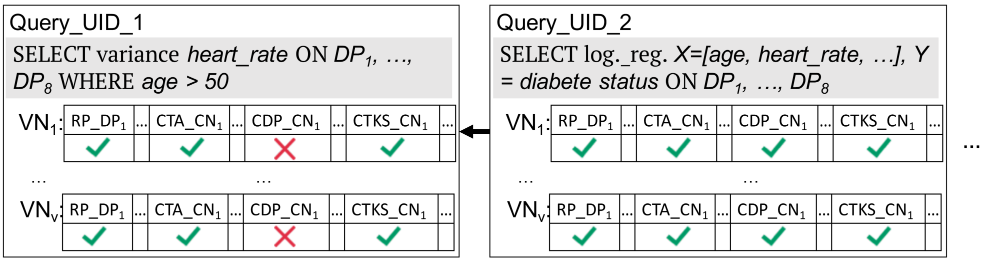

Query_UID_1
SELECT variance heart_rate ON DP1, …, DP8 WHERE age > 50
VN1: RP_DP1 … CTA_CN1 … CDP_CN1 … CTKS_CN1 …
✓ ✓ ✗ ✓
…
VNv: RP_DP1 … CTA_CN1 … CDP_CN1 … CTKS_CN1 …
✓ ✓ ✗ ✓
Query_UID_2
SELECT log._reg. X=[age, heart_rate, …], Y = diabete status ON DP1, …, DP8
VN1: RP_DP1 … CTA_CN1 … CDP_CN1 … CTKS_CN1 …
✓ ✓ ✓ ✓
…
VNv: RP_DP1 … CTA_CN1 … CDP_CN1 … CTKS_CN1 …
✓ ✓ ✓ ✓
…

Figure 4.3 – *Proof blockchain.* Each block contains Query ID and content, and each VN's *query-proofs map*. RP stands for range proof.

Security Arguments. If an entity trusts a threshold f_h of the VNs, it can verify the query correct execution by checking the corresponding block in the *proof blockchain*. The verifier can check that f_h nodes agree on the correctness of the proofs. A block is created for every query, even if the proofs are wrong, thus enabling any entity to determine which parties were involved in incorrectly computed queries. Otherwise, as all the proofs are universally verifiable and stored by all VNs, an auditor, not trusting f_h of the VNs, can request the proofs from a subset of them and check the proofs by itself.

Results Robustness. If the querier defines a query with range boundaries on the DPs' values, the DPs are requested to create proofs of range by following the algorithm detailed in Algorithm 4.1. For example, the DPs' inputs for a statistical query on human height can be limited to the centimeters range $[30, 250]$. This algorithm is built by adapting the $[0, u^l)$-range proof scheme proposed by Camenisch et al. [37] to the Anytrust model. In this algorithm, the prover, i.e., DP, writes its secret value m in base-u and commits to the u-ary digits by using the CN_is' signatures on these digits ($A_{i,b}$ in Algorithm 4.1). The l created commitments complete the proof. To adapt this algorithm to the Anytrust model, the DP must compute multiple proof elements, i.e., c, $V_{i,j}$, $a_{i,j}$, by combining all CNs' signatures, i.e., Z_i, $A_{i,b}$. This ensures that the DP uses at least one CN's signature for which it does not know the underlying secret. The same transformation in [37] can be applied to generalize the proof to any range $[b_l, b_u)$.

Security Arguments. Both the correctness and the zero-knowledge property of the range proof are proven by Camenisch et al. [37].

Algorithm 4.1 Input Range Validation in Anytrust Model

A DP proves that its secret $m \in [0, u^l)$, where u and l are two integers. $C_2 = mB + r\Omega$ corresponds to the right part of $E_\Omega(m) = (C_1, C_2)$. B is the Elliptic-Curve base point, $e()$ is a pairing function (bilinear map [37]) on an Elliptic Curve and H is a hash function.

```
Initialization:
Each $CN_i$ picks a random $x_i \in \mathbb{Z}_p$ and computes $Z_i \leftarrow Bx_i$, $A_{i,b} \leftarrow B(x_i + b)^{-1}$ $\forall b \in \mathbb{Z}_u$.
All $Z_i$ and $A_{i,b}$ are made public.
Proof Creation:
DP computes value $c = H(B, C_2, \sum_i Z_i)$ and
for each $j \in \mathbb{Z}_l$ such that $m = \sum_j m_j u^j$ do
    Pick three uniformly-random values $s_j, t_j, v_j \in \mathbb{Z}_p$
    for each computing node $CN_i$ do
        $V_{i,j} = A_{i,m_j} v_j$
        $a_{i,j} \leftarrow -s_j \cdot e(V_{i,j}, B) + t_j \cdot e(B, B)$
    end for
    $z_{v_j} \leftarrow t_j - v_j c \ (mod\ p)$ and $z_{m_j} \leftarrow s_j - m_j c \ (mod\ p)$
end for
DP picks $n \in \mathbb{Z}_p$ and computes $z_r = n - rc \ (mod\ p)$ and $D \leftarrow \sum_j Bu^j s_j + \Omega n$
DP publishes $proof = \{C_2, c, z_r, z_{v_j}, z_{m_j}, D, a_{i,j}, V_{i,j}\}$ $\forall j \in \mathbb{Z}_l$ and $\forall i \in \{1, \dots, \#CN\}$.
Proof Verification:
Any entity can check that:
    $D = C_2 c + \Omega z_r + \sum_j Bu^j z_{m_j}$ and $a_{i,j} = e(V_{i,j}, Z_i)c - z_{m_j} \cdot e(V_{i,j}, B) + z_{v_j} \cdot e(B, B)$, $\forall j \in \mathbb{Z}_l$ and $\forall i \in \{1, \dots, |CN|\}$.
```

These proofs are universally verifiable and sound in the Anytrust model. The latter comes from the fact that the elements depending on the CNs' secrets x_i are computed as a combination of all their public signatures. As at least one CN_i is honest-but-curious, one of the x_i is unknown (not revealed) to the DP (prover).

Computation Correctness. In order to ensure the correctness of the query execution, each computation executed by a CN has to be proven correct.

- *Collective Tree Aggregation.* The CNs provide to-be-aggregated input ciphertexts and the resulting ciphertexts that constitute the ZKP.
- *Collective Tree Obfuscation.* The CNs produce an obfuscation proof by relying on Expression (1.1) in Section 1.2. Each CN_i multiplies C by s_i to obtain the obfuscated ciphertext (C'_1, C'_2) with (a) $C'_1 = s_i C_1$ and (b) $C'_2 = s_i C_2$. For both equations, $y_1 = s_i$ is the discrete logarithm; we have the public values $A = C'_1$, $A_1 = C_1$ for (a) and $A = C'_2$, $A_1 = C_2$ for (b), which constitute the proof.
- *Collective Differential Privacy.* In this protocol, each CN sequentially executes a Neff shuffle and produces the corresponding ZKP of correctness described in Section 1.3. This proof basically contains the input and output lists, the public key encrypting the ciphertexts, and commitment values.
- *Collective Tree Key Switching.* The CNs create the ZKP by applying Equation (1.1) in Section 1.2, in which we have $y_1 = k_i$, $y_2 = \alpha_i$, the discrete logarithms of $k_i B = K_i$ and $\alpha_i B$, respectively. All points K_i, $\alpha_i B$, $A = w_{i,2}$, $A_1 = -rB$ and $A_2 = K'$ are made public and do not leak any information about the underlying secrets.

Security Arguments. We rely on proofs that are universally verifiable and zero-knowledge. They do not affect data confidentiality beyond what can be inferred from the proven facts themselves.

4.3.2 Drynx Optimized Design

We present DRYNX's final query execution pipeline, before describing how the query verification's performance can be optimized.

4.3.2.1 Full Query Execution Pipeline

We show DRYNX's full pipeline in Figure 4.4. Query execution and verification are executed concurrently and multiple steps of the query execution can be executed in parallel. The CNs aggregate each DP's response in CTA, as soon as they receive it. The noise generated from the CDP has to be added after all the results have been aggregated. However, if the differential privacy parameters are predefined, this protocol can be executed independently from the other steps or even pre-computed.

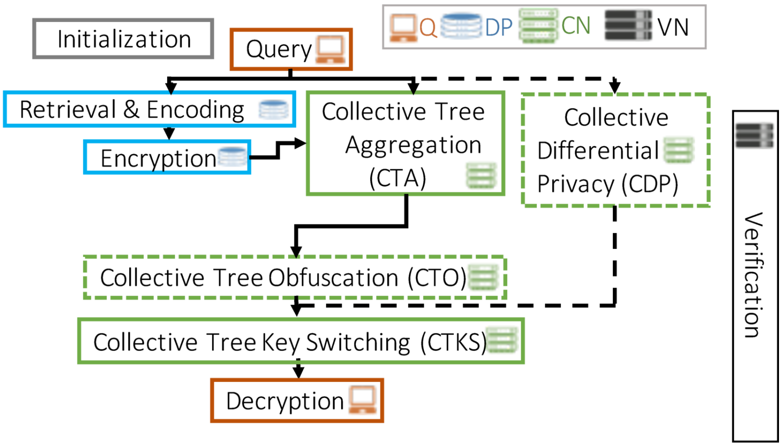

Q
DP
CN
VN
Initialization
Query
Retrieval & Encoding
Encryption
Collective Tree Aggregation (CTA)
Collective Differential Privacy (CDP)
Collective Tree Obfuscation (CTO)
Collective Tree Key Switching (CTKS)
Decryption
Verification

Figure 4.4 – DRYNX's complete optimized query-execution. Arrows represent causal links. Steps without direct links can be executed independently and dashed steps are optional.

4.3.2.2 Probabilistic Query Verification

To improve the performance of the query verification, we enable a probabilistic verification of the proofs by the VNs. We show that this strategy still enables a verifier to detect a misbehaving entity with a high probability, yet considerably improves performance (see Section 4.7). A *proof* for a specific operation (e.g., $CTKS$ for a set of ciphertexts S) can have multiple *sub-proofs* (e.g., $CTKS$ for one ciphertext $C \in S$). One *proof* is considered incorrect if one or more of the *sub-proofs* is incorrect. We introduce the two thresholds T and T_{sub} that define the probability of verifying a single *proof* and a *sub-proof*, respectively. We modify the VNs' operations in step 2 of the *Query Execution* described in Protocol 4.4, by adding this probabilistic verification based on T and T_{sub}. Each VN stores all the *proof* it receives. It then generates a random value $r \in [0,1]$; if $r < T$, it starts the probabilistic verification of the *sub-proofs*. For each *sub-proof*, the same method is applied, using T_{sub}.

Security Arguments. The probabilistic verification of each proof is redundantly done by each VN. A *proof* is verified with a probability $p_{ver} = 1-(1-T)^{N_{VN}}$, where N_{VN} is the number of VNs, and a *sub-proof* with a probability $p_{ver_{sub}} = 1-((1-T)+T(1-T_{sub}))^{N_{VN}}$. The probability that a *proof* or a *sub-proof* is verified by at least f_h nodes is

$$P_{f_h} = \sum_{i=f_h}^{N_{VN}} \binom{N_{VN}}{i} p^i (1-p)^{N_{VN}-i},$$

where p is either p_{ver} (for a *proof*) or $p_{ver_{sub}}$ (for a *sub-proof*). For example, if $N_{VN} = 7$, $T = 1$ and $T_{sub} = 0.3$, all the *proofs* are at least partially verified and each *sub-proof* is verified by $f_h = 5$ VNs with $P_{f_h} = 98.48\%$. Each *sub-proof* is thus verified by at least f_h of the VNs with a high probability. Due to the honesty assumption, a *sub-proof* is at least verified by one honest VN with a high probability. Moreover, the thresholds T and T_{sub} can be set to arbitrarily reduce the probability that one *sub-proof* is not verified by at least one honest node. Therefore, if all the VNs that participated in the verification agree on the result, the auditor knows the *proof* is correct, otherwise it can either choose to only trust some of the VNs or fetch all proofs and verify them itself, as all the proofs are universally verifiable. For example, an auditor can choose to verify only the proofs that were not checked by any of the VNs she trusts.

4.4 Security Analysis

We employed only existing, peer-reviewed cryptographic schemes and discussed the composability of the security of the different blocks in previous sections. In other words, we first discussed the security of DRYNX's simple query-execution pipeline (Figure 4.2) in Section 4.3.1.1 and then explained how we built on top of and integrated new features (Sections 4.3.1.2, 4.3.1.3 and 4.3.2.2) in this pipeline without hindering its security. We corroborate these arguments with a brief summary of the security analysis.

- *Data confidentiality.* In Section 4.3.1.1, we sketched the proof for confidentiality in our simplified system and discussed in Section 4.3.1 how further design choices do not hinder confidentiality. In summary, data confidentiality is ensured as the data are always encrypted and no operation, e.g., ZKP creation, affects it.

- *DPs' privacy.* DPs can privately decide whether to answer a query, and differential privacy is ensured for the DPs and individuals, which protects them from potential inferences stemming from the release of end results. The latter is ensured in DRYNX by blindly adding noise, sampled from a specific distribution, to the query end results. As described in Section 4.3.1.2, this noise can be verified to be from a specific distribution (e.g., Laplacian) and no entity knows which noise value is added.

- *Results robustness.* This is ensured as all DPs' values can be verified to be within a certain range and all CNs' computations must be proven correct, as depicted in Section 4.3.1.3. By enforcing the generation of range proofs by DPs, we protect against strong outliers, maliciously or erroneously input, which can significantly distort the query results. DPs can still input incorrect values, but their influence on the final result is limited. We give an intuition on how robust a computation is against such behavior in Section 4.7.2.

- *Computation correctness.* The proofs of correct computations (Section 4.3.1.3) ensure that the DPs' answers are correctly aggregated (CTA) and that the remaining steps (CTO, $CTKS$, CDP) are correctly executed.

4.5 Encodings

We present a set of statistical computations that can be executed in DRYNX. We then explain how to instantiate *encodings* (Definition 1) for the training of both linear and logistic regression machine-learning models. We adapt the logistic regression solution, proposed by Aono et al. [10], to our framework, thus enabling Q to train this model in a verifiable and privacy-preserving way, even in the presence of a strong adversary. Some of the encodings are adapted from the Corrigan-Gibbs and Boneh [52] system and improved upon.

Numerical Statistics. Table 4.1 lists a set of simple statistics that can be performed with DRYNX. The `sum`, `mean`, `variance`, `std. deviation`, `cosine similarity (cosim)` and $\mathtt{R}^2$ operations are executed by requiring the DPs to send the results of their local and partial statistic computations. As an example, for `variance`, each DP_i locally computes the sum of the values (records) h_j that match the query, $(\sum_{j=1}^{c_i} h_j)$ where c_i is DP_i's dataset cardinality, the square of those same values $(\sum_{j=1}^{c_i} h_j^2)$ and generates $\rho(\bar{r_i}) = ([\sum_{j=1}^{c_i} h_j, \sum_{j=1}^{c_i} h_j^2], c_i)$. These values are independently aggregated among all DPs and the overall variance is computed by Q, after decryption, using the corresponding π (defined in Table 4.1). For the `frequency count`, DPs are expected to send the vector $\mathbf{V_i}$ filled with the number of occurrences (fc) for specific values. The `cosine similarity` is computed between two vectors ϕ and $\bar{\boldsymbol{\phi}}$, where each DP_i holds a subset of the coefficients of each vector.

Operat. (f)	π (on $\lvert S\rvert$ DPs)	ρ $(\mathbf{V_i}{=}[v_{i,1},...,v_{i,d}],c_i)$
sum	$\sum_{i=1}^{\lvert S\rvert} v_{i,1}$	$([\sum_{j=1}^{c_i} h_j],\ c_i)$
mean	$\frac{\sum_{i=1}^{\lvert S\rvert} v_{i,1}}{\sum_{i=1}^{\lvert S\rvert} c_i}$	$([\sum_{j=1}^{c_i} h_j],\ c_i)$
variance std. dev.	$\sigma^2 = \frac{\sum_{i=1}^{\lvert S\rvert} v_{i,2}}{\sum_{i=1}^{\lvert S\rvert} c_i} - (\frac{\sum_{i=1}^{\lvert S\rvert} v_{i,1}}{\sum_{i=1}^{\lvert S\rvert} c_i})^2$ $\sigma{=}\sqrt{\sigma^2}$	$([\sum_{j=1}^{c_i} h_j,\ \sum_{j=1}^{c_i} h_j^2],$ $c_i)$
AND/OR	$\sum_{i=1}^{\lvert S\rvert} v_{i,1} \overset{?}{=} 0$	$([R_j],c_i)$ or $([b_j],\ c_i)$
min/max	$l/rm_{\neq 0}(\sum_{i=1}^{\lvert S\rvert} v_{i,1},...,$ $\sum_{i=1}^{\lvert S\rvert} v_{i,d})$	$([R_{j,1},\ ...,\ R_{j,d}],\ c_i)$ or $([b_{j,1},\ ...,\ b_{j,d}],\ c_i)$
frequ. count	$\sum_{i=1}^{\lvert S\rvert} v_{i,1},...,\sum_{i=1}^{\lvert S\rvert} v_{i,d}$	$([fc_{j,1},\ ...,\ fc_{j,d}],\ c_i)$
set int/un	$\sum_{i=1}^{\lvert S\rvert} v_{i,1},...,\sum_{i=1}^{\lvert S\rvert} v_{i,d}$	$([R_{j,1},\ ...,\ R_{j,d}],\ c_i)$ or $([b_{j,1},\ ...,\ b_{j,d}],\ c_i)$
cosim	$s(\phi,\overline{\phi}){=}\frac{\sum_{i=1}^{\lvert S\rvert} v_{i,1}}{\sqrt{\sum_{i=1}^{\lvert S\rvert} v_{i,2}}\sqrt{\sum_{i=1}^{\lvert S\rvert} v_{i,3}}}$	$([\sum_{j=1}^{c_i} \phi_j\overline{\phi}_j,\sum_{j=1}^{c_i} \phi_j^2,$ $\sum_{j=1}^{c_i} \overline{\phi}_j^2],\ c_i)$
R^2	$1 - \frac{\sum_{i=1}^{\lvert S\rvert} v_{i,3}}{\sigma^2}$	$([\sum_{j=1}^{c_i} y_j,\ \sum_{j=1}^{c_i} y_j^2$ $\sum_{j=1}^{c_i}(y_j-\hat{y}_j)^2],\ c_i)$

Table 4.1 – Example set of *encoding* instantiations. All DPs *encoding*s (ρ) are then aggregated such that Q computes π at the end.

Bit-Wise Statistics. As depicted in Table 4.1, bit-wise operations can be executed in two ways: Each DP_i either (1) sends a random encrypted integer R or (2) sends an encrypted bit b. For (1), in the `OR` (resp. `AND`) case, each DP_i is requested to send an encrypted integer $E_K(R_i)$, where $R_i = 0$ if the input is 0 (resp. 1), and a random positive integer otherwise. The `OR` (resp. `AND`) expression is $true$ (resp. $false$) if the sum $\sum R_i > 0$. Q obtains the final result by testing if the output is 0 or not. The result of this operation can be erroneous if $\sum R_i \equiv 0 \ mod(\#G)$, or in other words, if the order $\#G$ of the Elliptic Curve subgroup divides the sum of all DPs' random values. This happens only with a probability smaller than $1/(\#G-1)$ (proof in Appendix B.1). This probability is close to 0 as $\#G$ is much bigger than the decryptable plaintext values, and can be further reduced by repeating the query. Alternatively, in (2) each DP_i has to send $b_{i,j} = 0$ or $b_{i,j} = 1$ encrypted value. This eliminates the error probability but requires more computations and proofs of correctness, as the DPs have to prove that their values are in $\{0,1\}$, and a CTO protocol (Section 4.3.1.2) has to be executed to preserve privacy. The `min` (resp. `max`) is computed by applying the `or` operation element-wise among vectors $\mathbf{V_i}$. Each DP_i computes its local min (resp. `max`) m_{DP_i} in a specified range, e.g., [0:100], which is represented by $\mathbf{V_i} = [b_{i,0},...,b_{i,100}]$. Each $b_{i,j} > m_{DP_i}$ (resp. $b_{i,j} < m_{DP_i}$) is encoded with a '1' (or random) and a '0' otherwise. The `min` (resp. `max`) across all DPs corresponds to the leftmost (resp. rightmost) position with a '1' in the vector resulting from the `OR` operation. Similarly, the `set intersection` (resp. `union`) is computed by using the `AND` (resp. `OR`) operation element-wise on the vectors $\mathbf{V_i}$.

Regression Models.

Linear Regressions. We assume a dataset distributed over the DPs with D features $x_1,...,x_D$ and a label value y such that $y \approx c_0 + c_1 \times x_1 + c_2 \times x_2 + ... + c_D \times x_D$. DRYNX computes the least-squares linear fit over all the DPs by building a system of $D+1$ equations that Q can use in order to compute the linear regression coefficients $c_0, c_1, c_2, ..., c_D$:

$$\begin{pmatrix} n & \sum x_{\mu,1} & \dots & \sum x_{\mu,D} \\ \sum x_{\mu,1} & \sum x_{\mu,1}^2 & \dots & \sum x_{\mu,1}x_{\mu,D} \\ \dots & \dots & \dots & \dots \\ \sum x_{\mu,D} & \sum x_{\mu,1}x_{\mu,D} & \dots & \sum x_{\mu,D}^2 \end{pmatrix} \begin{pmatrix} c_0 \\ c_1 \\ \dots \\ c_D \end{pmatrix} \approx \begin{pmatrix} \sum y_\mu \\ \sum y_\mu x_{\mu,1} \\ \dots \\ \sum y_\mu x_{\mu,D} \end{pmatrix} \tag{4.1}$$

where all the sums are between $\mu = 1$ and $\mu = \sum_{i=1}^{|S|} c_i$. Each DP_i sends $\sum_{j=1}^{c_i} x_{j,\eta}$, $\sum_{j=1}^{c_i} x_{j,\eta}x_{j,\zeta}$, $\sum_{j=1}^{c_i} y_j$, $\sum_{j=1}^{c_i} y_j x_{j,\eta}$, $\forall \eta,\zeta \in \{1,2,...,D\}$, $\eta \neq \zeta$.

Logistic Regressions. We consider again a dataset of n records (distributed among the DPs) with a dimension D where each record $x^{(\mu)} = (1, x_1^{(\mu)}, \cdots, x_D^{(\mu)}) \in R^D$ consists of D features and an offset term of 1, and is associated with a label $y^{(\mu)} \in \{0,1\}$. The original logistic regression cost function is

$$J(\theta) = \tfrac{1}{n}\textstyle\sum_{\mu=1}^{n}\Big[-y^{(\mu)}\log(h_\theta(x^{(\mu)})) - (1-y^{(\mu)})\log(1-h_\theta(x^{(\mu)}))\Big] + lr_\theta,$$

where $h_\theta(x) = 1/(1+\exp(\sum_{\eta=0}^{D}\theta_\eta x_\eta))$ and $lr_\theta = \frac{\lambda}{2n}\sum_{\eta=1}^{D}\theta_\eta^2$, λ is the L2-regularization parameter. $J(\theta)$ can be approximated by a linear function

$$J_a(\theta) = \Big[\tfrac{1}{n}\textstyle\sum_{\tau=1}^{k}\sum_{r_1,...,r_\tau=0}^{D} a_\tau(\theta_{r_\tau}\cdots\theta_{r_\tau})A_{\tau,r_1,...,r_\tau} - a_0\Big] + LR_\theta,$$

by using the fact that $\log(\frac{1}{1+\exp(x)}) \approx \sum_{\tau=0}^{k} a_\tau x^\tau$, where $a_0, a_1, ..., a_k$ can be chosen as the $k+1$ first coefficients of the Taylor expansion of $\log(\frac{1}{1+\exp(x)})$, or as the coefficients of the quadratic approximation that minimizes the area between the original function and its approximation. The $A_{\tau,r_1,\cdots,r_\tau}$ coefficients are defined by

$$A_{\tau,r_1,\cdots,r_\tau} = \textstyle\sum_{\mu=1}^{n} a_{\tau,r_1,\cdots,r_\tau}^{(\mu)} = \sum_{\mu=1}^{n}(y^{(\mu)} - y^{(\mu)}(-1)^\tau - 1)(x_{r_1}^{(\mu)}\cdots x_{r_\tau}^{(\mu)}),$$

where the $a_{\tau,r_1,\cdots,r_\tau}^{(\mu)}$ are computed and encrypted by the DPs before being collectively aggregated by the CNs.

Neutral Response. A neutral response for `and` and `set intersection` is $O = [1,...,1]$, and $O = [0,...,0]$ for the other operations mentioned above.

Optimized and Iterative Encoding DRYNX can also be used in order to execute iterative processes, e.g., a k-means algorithm. In this case, each iteration can simply be mapped to a query sent to the system. An iterative process can also be used in order to optimize existing *encodings*, such as the `min` and `max`. In their basic versions, these *encodings* rely on a d-bit vector in which each bit represents a value in a predefined range of size $d = |b_u - b_l|$. This means that each DP sends d ciphertexts. This process can be optimized by using a binary-search iterative process as depicted in Protocol 4.5. In the *Range Reduction* step, each query only requires one ciphertext per DP and reduces by half the range of possible answers. This step is repeated until this range is reduced to a predefined size EL (i.e., entropy limit). It must be noted that the execution of other iterative processes would work in a similar way: For example, for a k-means algorithm [107], Q performs one iteration by executing one query that includes the centroids in clear; the DPs then assign their points to the closest centroid before aggregating their points by cluster; then, the same operation is repeated among all DPs by using DRYNX typical query workflow and Q computes the new centroids. As in Protocol 4.5 and as described below, this algorithm leaks the cleartext intermediate results. We do not address the problem of hiding the revealed intermediate results, e.g, by using differential privacy, in this chapter.

Protocol 4.5 Iterative Process (`max` example)

Input. Query = `max` in $ra = [b_l, b_u]$ and EL
Output. Max value
Range Reduction:

```
while $|ra| > EL$ do
    $Q$ sends SELECT OR ($\exists v \in [\lfloor \frac{(b_l+b_u)}{2} \rfloor, b_u]$) ON $DP_1,...,DP_{|S|}$
    if query returns $true$ then
        $ra = [\lfloor \frac{(b_l+b_u)}{2} \rfloor, b_u]$
    else
        $ra = [b_l, \lceil \frac{(b_l+b_u)}{2} \rceil]$
    end if
end while
```

Final Step:

```
$Q$ sends SELECT MAX $[b_l, b_u]$ ON $DP_1,...,DP_{|S|}$
```

Security Arguments. For all *encoding* and in each query, Q learns the elements of $\mathbf{V}$ (aggregated over all DPs) and the (approximate) number of samples considered c, as defined by *encoding*.

For the iterative process, in the *Range Reduction*, the DPs' answers remain confidential, but the range is sent in clear in each query thus revealed to other entities. Q controls the size of the range of possible values that is leaked by defining an entropy limit EL. In the *final step,* the `max` query is privately executed on the remaining range. This provides a tradeoff between performance and privacy (that we analyze in Section 4.7). The number of ciphertexts is lowered to $n = g + \lceil \frac{d}{2^g} \rceil$, $g = \lfloor log_2(\frac{d}{EL}) \rfloor$ as each query reduces by half the range of possible answers, which reduces the amount of computations and proofs by a factor $\frac{d}{n}$. For example, if Q wants to know the DPs' minimum value in $[0, 1000)$ with $EL = 100$, the workload is reduced by a factor of 7.8 and the query leaks a range of 100 possible minimum values.

4.6 Discussion and Extensions

We illustrate multiple extensions for DRYNX by relying on our use cases, HDS and PDS (Section 4.1.1).

Modularity. DRYNX is highly modular and some of its security features can be enabled or disabled, depending on the application. For example, if results robustness is not required, input-range validation can be omitted without hindering DRYNX's execution and the remaining security guarantees are preserved. The same applies for DPs' privacy features, e.g., differential privacy.

For example, in HDS, each hospital (or DP) locally executes the query on multiple patient records and the range proofs can be omitted if the range of possible values is too broad or if the hospital is trusted to input correct values. Otherwise, the range boundaries have to be set accordingly. In this case, the querier has to use her knowledge on the attributes involved (e.g., age is between 0 and 150) and the information she has on the DPs' data (e.g., DPs have a maximum of X data samples) to define the ranges. In PDS, the ranges for the input values can be used to enforce tighter bounds (e.g., heart rate can only take values in [40,100] beats-per-minute) as each DP has one data record.

DRYNX also enables the collective protection of data at rest by having DPs locally encrypt their data with the CNs' collective key K. This limits the flexibility of the system as DPs are then required to pre-compute all necessary inputs (e.g., the square root of the values to enable the computation of the `variance`) and the range proofs before entering the encrypted data in their databases. It also requires a fixed set of CNs, as only they can operate with that pre-encrypted data.

As mentioned before, DRYNX's primary goal is to guarantee DPs' privacy and still enable the queriers to obtain the results of computations performed over multiple databases. For this, DRYNX enables optional security and privacy features, such as differential privacy. These features can be enabled or disabled depending on the application requirements, hence enabling multiple trade-offs between security and privacy, performance and accuracy (see below).

Collusion Resistance. Each participant can play multiple roles without hindering DRYNX's security. For example, in HDS, a hospital can be a DP and also play the role of a CN, to ensure its data confidentiality without having to trust any other hospital. It can also be a VN thus take part in the verification process.

Availability. DRYNX's privacy and security guarantees hold even in the case where multiple CNs or DPs become unavailable. Any entity can leave or join the system without hindering

DRYNX's operation, as long as they are not involved in a query under execution. In the event of a CN becoming unresponsive during the query execution, the CTA and $CTKS$ steps cannot be finalized, as they both require the participation of all CNs. Therefore, in this case, the process is stopped and Q can request the same query by choosing another set of CNs, e.g., by excluding the faulty CN(s). An unresponsive DP only reduces the number of responses included in the statistic being computed and does not disrupt DRYNX's process. Standard mechanisms, e.g., limiting the rate at which queries are accepted, can be implemented in DRYNX to prevent DDoS attacks.

Accuracy. There are several aspects that can influence output precision in DRYNX. (a) We first remark that the DPs' inputs to the system have to be approximated by fixed-point representation if they are floating values, as explained in Section 1.1.
(b) DRYNX's encodings and query executions do not intrinsically hinder the accuracy of the computed results, as all operations are exact, as long as the target function is exactly *encodable*. In fact, it is worth noting that the encoding for the logistic regression training is built from an approximation of the original cost function.
Additionally, (c) the DPs can privately decide whether to answer a query; this choice can influence the final result. However, the number of samples considered in the computation, i.e., c_i in Definition 1, is always sent to Q, who can then observe if this number changed since her last query. It also enables her to take an informed decision on the statistical significance of the results, to accept them or not.
(d) DRYNX can guarantee differential privacy by adding noise to the final result. In this case, DRYNX returns approximate results, and the accuracy loss depends on the chosen privacy parameters and the executed operation. The choice of these parameters and the perturbation introduced in the results is thus orthogonal to this work.
Finally, (e) malicious DPs can try to distort the query result by inputting erroneous values. DRYNX limits malicious DPs' influence on the final result by enabling the querier to restrict the range of possible inputs. This bounds the perturbation that some DPs can generate on the results. If the inputs were not bounded, one malicious DP could completely distort the final result by inputting extreme values. It is difficult to provide hard numbers for the accuracy of DRYNX in the presence of malicious DPs, as it depends on many parameters such as the executed operation, the chosen input ranges, the number of DPs and data records. Nonetheless, in Section 4.7 we show how the use of ranges limits the influence of malicious DPs in two examples.

Authentication/Authorization. Authentication and authorization fall out of the scope of this paper, but for the sake of completeness we briefly mention here that DRYNX can integrate off-the-shelf solutions based on federated or distributed architectures [127; 179; 183].

4.7 Performance Evaluation

We discuss our experimental setup and evaluate DRYNX's performance. We show that it scales almost (in some cases better than) linearly with the number of CNs, VNs and DPs, and we compare DRYNX against existing solutions. We also discuss multiple security, privacy and performance tradeoffs.

4.7.1 System Implementation

We implemented DRYNX in Go [98], and our full code is publicly available [88]. We relied on Go's native crypto-library and on DeDiS public advanced crypto-libraries [61]. For the implementation of the proofs' storage and verification, we use a skipchain [177], which is made of blockchain-like blocks that, to enable clients to efficiently navigate arbitrarily on the chain, also contain back-and-forward pointers to older and future blocks. We rely on a (private) permissioned blockchain [57], as in our examples HDS and PDS (Section 4.1.1), the participants, i.e., researchers, patients or hospitals, have to be known and authorized. However, DRYNX works independently of the blockchain type, and a permission-less blockchain can also be used in a less restrictive scenario. DRYNX relies on additive homomorphic encryption and works independently of the used Elliptic Curve; we tested it on the Ed25519 [22] and bn256 Elliptic Curves [14]. Both curves provide 128-bit security, and we used bn256 by default as it enables pairing operations (required for range proofs). As for UNLYNX presented in Chapter 3, DRYNX's prototype is built as a modular library of protocols that can be combined in multiple ways. The communication between different participants relies on TCP with authenticated channels (through TLS).

4.7.2 System Evaluation

We relied on a similar evaluation environment as in Chapter 3. We used Mininet [160] to simulate a realistic virtual network between the nodes; we restricted the bandwidth of all connections between nodes to 100Mbps and imposed a latency of 20ms on all communication links. We evenly distributed the CNs, DPs, VNs and Q on a set of 13 machines that have two Intel Xeon E5-2680 v3 CPUs with a 2.5GHz frequency that supports 24 threads on 12 cores and 256GB RAM.

We begin our evaluation by studying how the different steps in DRYNX's pipeline can be executed in parallel. We then show that DRYNX's runtime only slightly increases when the number of records per DP grows (and the number of DPs remains constant).

In our **default setup**, we consider 6 ***CNs*** and 7 ***VNs.*** We set the proof verification thresholds $T = 1.0$ and $T_{sub} = 0.3$ and show, in Section 4.7.2.1, the effect of these thresholds on DRYNX's execution time. The joint use of these thresholds ensures that all the proofs are at least partially verified and that each *sub-proof* is verified by f_h VNs with a probability of 98.5%. We show

DRYNX'runtime without the CDP protocol as CDP can be pre-computed or run in parallel with other steps. We notice that the CDP's runtime depends on the number of CNs and on the size $\tilde{l}$ of the list of noise values. This creates a tradeoff between privacy and performance as a greater $\tilde{l}$ provides a higher privacy level, as it reduces $\delta = 1/\tilde{l}$ but also increases the time to generate and shuffle the list of noise values. With a Laplacian distribution and $\tilde{l} = 100$, CDP's runtime is 2.9 seconds with an overhead of 8.1 seconds for the proof verification.

4.7.2.1 Drynx Evaluation

Parallel Execution. Figure 4.5 shows the runtime for training a *logistic regression* model. We use a randomly-generated dataset of 12 floating-point features and 600,000 records split among 12 DPs. We remark that the operations are verified in parallel to the query execution; this parallelization enables Q to obtain the query result as soon as it is computed (denoted by query execution dashed line). At the end of the verification process, an auditor can check the query by verifying the signature and the *query-proofs map* of the corresponding block in the *proofs blockchain*, which in this case takes 0.4 seconds. The blocks' sizes are small as they only contain the query and the corresponding *query-proofs map*; in this example one block is 56kB.

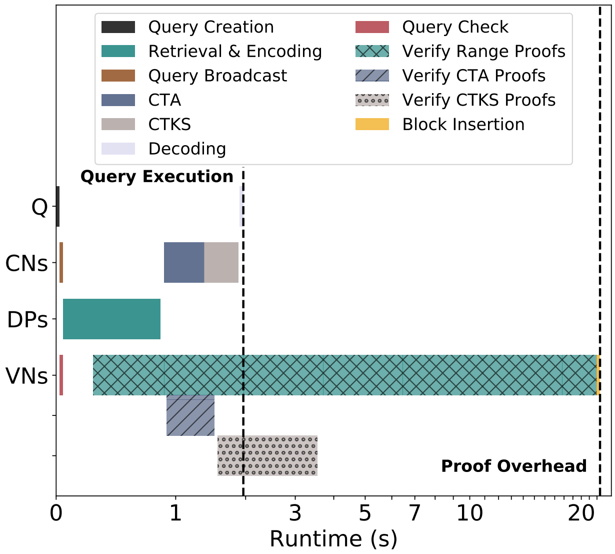
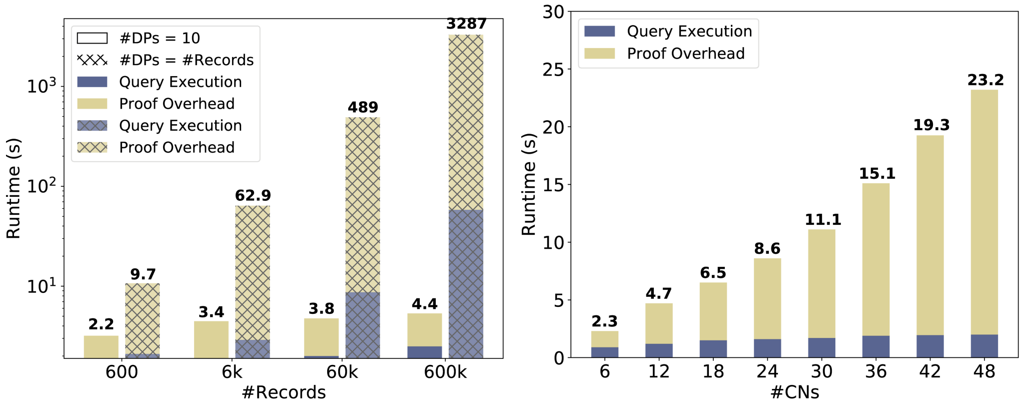


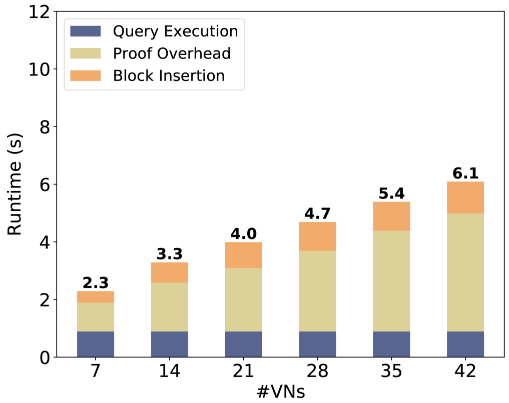
Figure 4.5 – `Logistic regression training` with 12 features, 600,000 records and a maximum of 100 iterations.

Scaling. We show how DRYNX's execution time evolves with an increasing number of data records (Figure 4.6a), CNs and DPs (Figure 4.6b) and VNs (Figure 4.6c). Inspired by HDS and PDS, we simulate the computation of the heart-rate *variance* (values between $[0, 256)$) over a set of distributed patients. In Figure 4.6a, we observe that DRYNX scales better with (a) the number of records per DP (and fixed number of DPs) than (b) with the number of DPs; case (a) represents HDS, where a DP is an hospital with a database of multiple patients, whereas case (b) represents PDS, where each patient is a DP ($\#DPs = \#records$). This is because (a) enables the DPs to locally pre-aggregate their data, thus reducing the amount of proofs and computations. For Figures 4.6b and 4.6c and for the remaining part of the evaluation, we set the number of DPs to 10 per CN. In *HDS*, this could correspond to a use case in which some DPs are hospitals and the others are independent doctors sharing their data. We observe that DRYNX's runtime increases with the number of DPs, CNs, and VNs. However, an increasing number of CNs and VNs also means a higher security level, as the trust is distributed among more entities.



(a) `Variance`: increasing number of records.

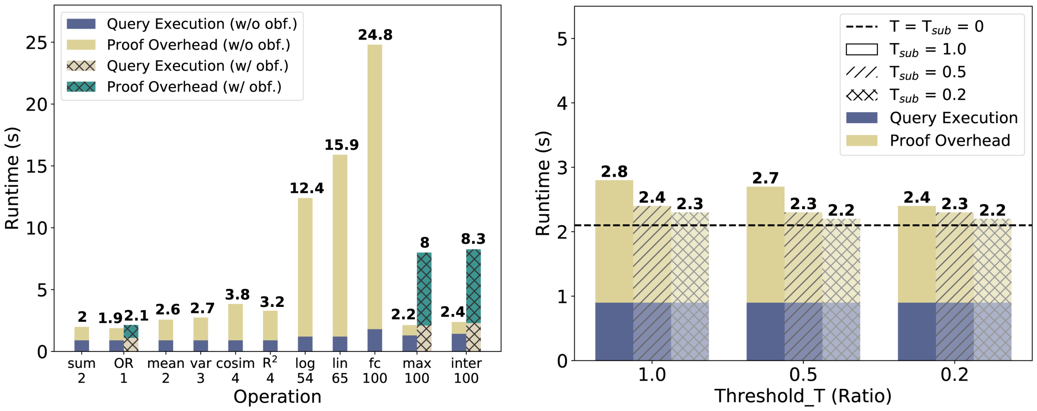
(b) `Variance`: increasing number of CNs and DPs with 10 DPs per CN.



(c) `Variance`: increasing number of VNs.

Figure 4.6 – DRYNX's scaling.

Operations. Figure 4.7a shows DRYNX's runtime for all the operations with a large integer range of $[0, 2^{20}]$ for each of the DPs' inputs (the size of the DPs' inputs is shown below each operation). We observe that for all operations, the query execution time is always below 1.5 seconds; and the overhead incurred by the proofs verification increases with the size of the DPs' inputs. This is expected, as the larger the DPs' inputs become, the more ciphertexts there are for the system to process, and the more proofs there are to verify. We also observe that bit-wise operations take more time when the DPs opt to send a bit value that is then obfuscated (using the CTO protocol).



(a) Runtime for different operations with DPs' inputs sizes. Range of accepted input values: $(0, 2^{20})$.

(b) `Variance`: proofs verification thresholds.

Figure 4.7 – DRYNX's operations and verification thresholds.

Verification Thresholds. In Figure 4.7b, we show how the different thresholds on the proofs verification affect DRYNX's performance with a *variance* query. It can be seen that sending the proofs (communication time is denoted by a dashed line) is the most time consuming part, and that reducing the thresholds reduces the verification time. For example, by having $T = 1$ and $T_{sub} = 0.2$, we effectively reduce the verification workload by a factor close to 0.8, and a *sub-proof* is still verified by $f_h = 5$ of the VNs with a high probability (83.48%).

Malicious DPs. By enforcing DPs' values to be within a specific range, DRYNX limits the influence of malicious DPs on the computed statistic. We illustrate this in a simple and realistic example (using PDH from Section 4.1.1) by computing the `average` heart rate over a dataset of 8922 hypertensive patients [146]. The real heart-rate values are limited to be between 40 bpm (beats per minute) and 100 bpm and, as presented by Lorgis et al. [146], the average value obtained among honest DPs is a_h =70 bpm with a 95% confidence interval of ±6 bpm. Each patient (DP_i) must send $(\mathbf{V_i}, c_i) = ([heart\_rate], count)$ (Definition 1), in which $heart\_rate$ has to be in $[40, 100]$ and $count$ in $[0, 1]$. In order to maximize the result's distortion, a malicious DP can send an extreme value, which is within the range bounds. We assume that all malicious DPs collude and send the same value $heart\_rate = e$, and that the computed average is given by $a_m = (h \cdot a_h + e \cdot d)/(h + c)$, where h and d are the numbers of honest and dishonest DPs, and c is the sum of c_i sent by malicious DPs. The relative error is $|1 - (a_m/a_h)|$. We remark that a malicious DP can maximize this error with a valid input by sending $([100], 0)$. In Figure 4.8, we observe that with 1% of malicious DPs for the range [40,100], the highest relative error is 1.44%. This error corresponds to 1 bpm, still in the 95% confidence interval. We observe similar results when the cosine similarity is computed in the same settings. For this example, we also present the worst-case scenario in which the cosine similarity computed on the honest DPs is 1 and the malicious DPs input extreme values from the range of accepted values to reduce the similarity. As shown in Figure 4.8, these numbers highly depend on the chosen bounds. Even if many other factors influence this error (e.g., the computed operation and the distribution of the values), it shows that Drynx can limit the power of malicious DPs.

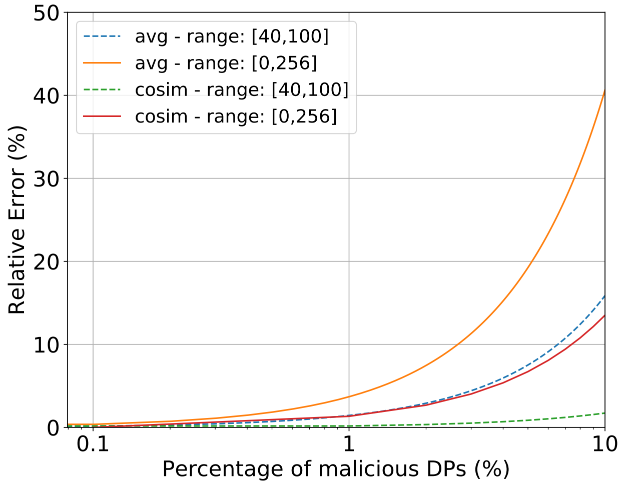
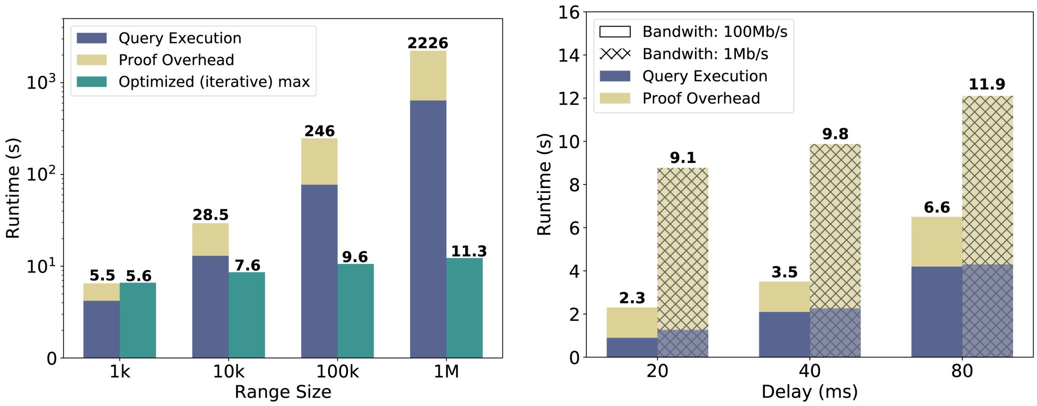


Figure 4.8 – `Average` and `cosim`: influence of malicious DPs.

Iterative Queries. Figure 4.9a depicts how Drynx's runtime can be reduced by using multiple queries to execute a `min/max` operation in a binary-search style. This represents a tradeoff between privacy and performance, as each iterative query is sent in clear, leaking the interval where the min/max value is. We assume that Q sets the entropy limit $EL = 100$, in other words, another entity in the system can learn that the `min/max` is in an interval of at least 100 values. The precise value is kept private. We observe that the execution time is not improved when the range is small, but is greatly reduced when the range grows, reaching an execution time reduction of almost 96% at a range size of 100,000.



(a) `max` (iterative): increasing range size.

(b) `Variance`: runtime w.r.t. network bandwidth and delay.

Figure 4.9 – Drynx Evaluation. In Figure (a), the optimized *max* aggregates the runtime for all necessary query executions and the respective proof overheads.

Communication. Figure 4.9b depicts Drynx's runtime evolution with respect to both the communication delay and bandwidth capacity with a heart rate *variance* query. We remark that when the latter is reduced by a factor 100, the runtime increases by a factor 2 or 3. This shows that our system is more sensitive to communication delay than bandwidth capacity.

	DPs answer	*DPs* answer with RV	*CTA*	CTO	*CTKS*	*CDP*
QE C.C.	-	-	(CN-1) · *A*	(CN-1)·*A* + (2·CN)·*SM*	(CN-1)·*A* (3·CN)·*SM*	CN · VS
QE comm.	DP · 4p (1.28kB)	DP · 4p (1.28kB)	CN · 4p (0.768kB)	2 · CN · 4p (1.536kB)	CN · 6p (1.344kB)	CN · 4p · n (76kB)
QV comm. and storage	DP ·VN ·(s+4p) (15.68kB)	DP · VN · (s + (7 + 2l + 2l · CN)p + l · CN · pap + h) (2.43MB)	CN · V N · (s + 4p) (9.048kB)	CN ·VN ·(s+8p+h) (16.128kB)	CN ·VN ·(s+10p+h) (18.816kB)	(CN +1)·VN ·(s+ 13np + h) (2.04MB)

Table 4.2 – Communication and Storage costs. DP, VN, CN = number of entities; RV = Range Validation; QE=query execution; C.C.=computation complexity; QV=query verification; *A*=ciphertext addition; *SM*= scalar multiplication; *VS*= verifiable shuffle; *p*= Elliptic Curve point; s, h and pap are the size of the Schnorr-signature, the hash and the pairing point, respectively. l comes from the range boundaries and n is the list size in CDP.

Bandwidth. In Table 4.2, we present the computation and bandwidth complexities for 1 ciphertext (i.e., 2 points (2p) on the Elliptic Curve, 2p = 64 bytes) per DP. We use DP, VN, and CN as the numbers of corresponding entities in the system. s is the size of the Schnorr signature [80] ($s = 96$ bytes), h is the hash size ($h = 32$ bytes), l comes from the range $[0, u^l)$ for the range proofs ($u^l = 16^2, l = 2$), pap is a pairing point's size ($pap = 384$ bytes) and n is the number of values that are used in the CDP ($n = 100$). We do not include the computational complexity for the local computations executed by the DPs and CNs. We refer to Neff's work [173] for the complexity of the verifiable shuffle (VS). We observe that when the number of CNs and VNs increases, the computational, bandwidth and storage costs increase for all the steps. As having more CNs or VNs improves the security and the distribution of the workload in the system, it creates a tradeoff between security, efficiency, and scalability.

4.7.2.2 Comparison with Existing Works

We supplement the related work's overview in Chapter 2, by presenting here a qualitative and quantitative comparison with multiple systems that are DRYNX's closest related works. We compare DRYNX against SMCQL [15], Prio [52], Boura et al. [34], Aono et al. [10], Kim et al. [126] and Gazelle [123]. In Table 4.3, we show that DRYNX provides several functionalities in a strong threat model and achieves results that can rival with other secure and dedicated approaches, notably in the training of logistic regression models as depicted in Figure 4.3. DRYNX performs as well or better than its closest related work, Prio, and provides better security guarantees.

We observe that solutions based exclusively on secret sharing and garbled circuits, namely SMCQL [15], Prio [52] and Boura et al. [34], offer multiple or advanced functionalities but fail to provide proofs of correct executions. Systems solely based on homomorphic encryption (HE), namely Aono et al. [10], Helen [263] and Kim et al. [126], are limited in the functionalities

		SMCQL	Boura et al.	Prio	Aono et al.	Kim et al.	Gazelle	Helen	Drynx
System Properties	Distribution of Trust	X	✓	✓	X	X	2-party	✓	✓
	Distrib. Comput. / Stor.	✓	✓	✓	X	X	2-party	✓	✓
	Modular architecture	X	✓	✓	X	X	X	X	✓
Security Guarantees (assuming Drynx's threat model)	Data Confidentiality	X	✓	✓	✓	✓	✓	✓	✓
	DPs' Privacy	X	X	X	X	X	X	~	✓
	Comput. Correctness	X	X	X	X	X	X	✓	✓
	Results' Robustness	X	X	X	X	X	X	✓	✓
Functionalities	Statistics	✓	X	✓	X	X	X	X	✓
Accuracy|AUC	Lin. Reg. Training	X	✓	✓	~	~	X	✓	✓
	Log. Reg. Training	-	✓	~	✓	✓	X	X	✓
	SPECTF [229]				75.4|0.76	-			74.8|0.73
	Pima [195]				75.4|0.87	-			77.5|0.83
	LBW [149]				-	69.3|0.66			70.2|0.73
	PCS [201]				-	69.1|0.75			75.1|0.81
	Neural Networks	X	X	X	X	X	✓	X	X

Table 4.3 – Solutions Comparison. For logistic regression, we split the datasets [147; 192; 198; 226] among 10 DPs before standardization and use a scale factor = 10^2 for fixed-point representation and a learning rate of 0.1. We split the datasets with 80% for training and 20% for testing.

they offer. Furthermore, Aono et al. [10] and Kim et al. [126] rely on data centralization. Gazelle [123] combines HE and garbled circuits and enables complex evaluations of neural networks, but does not protect DPs' privacy or provide computation correctness. Contrarily, Drynx enables multiple operations while distributing trust, computations, and data storage, and it provides strict security guarantees in a stronger adversarial model.

We quantitatively compare Drynx to Prio [52], which is, to the best of our knowledge, the closest prior work. Prio [52] relies on secret-shared non-interactive proofs that are created by the DPs to prove the correctness of their inputs to the system and that are collectively verified by the CNs. Even though both systems have similar functionalities, Prio provides input-range verification and computation correctness only when all the CNs are honest-but-curious. We adapted the Gorrigan-Gibbs prototype implementation [197] of Prio to a similar deployment environment as Drynx so that both use the same communication settings, thus enabling a fair comparison. In Figure 4.10, we compare Prio's runtime in an illustrative example by using the `min` operation on the range $[0, 1000)$ with increasing number of CNs and DPs, against multiple settings of Drynx. This figure shows that Drynx significantly outperforms Prio when computing `min` without using obfuscation (CTO) and thus accepts a small probability of error ($1/(\#\mathcal{G})$) while avoiding the need for range proofs. If we use obfuscation, Drynx scales similarly as Prio, but it must be noted that Drynx performs its operations in a stronger threat model. When used in Prio's threat model (delimited by a black line), Drynx is about two times faster. This is because each range proof can be sent and verified by a single VN as all VNs are considered honest-but-curious under Prio's threat model.

We also remark that we brought multiple improvements on the work presented in Chapter 3

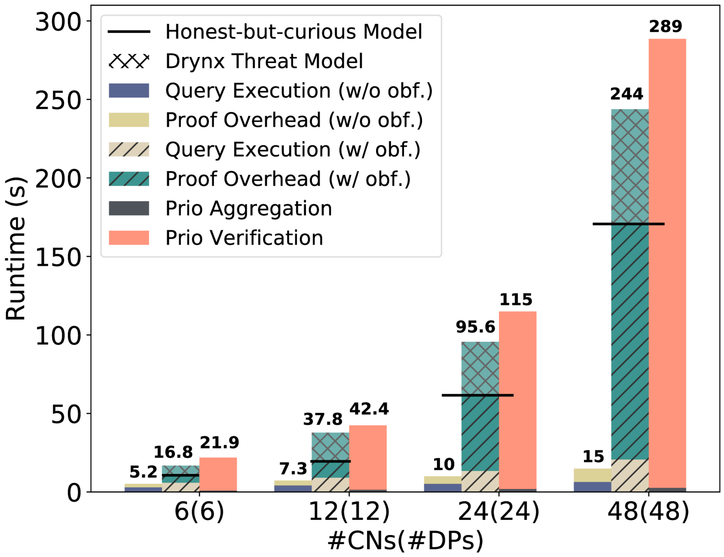


Figure 4.10 – Comparison with Prio for a *min* query. The left bars present DRYNX's execution time and right bars show Prio's runtime.

(UNLYNX). DRYNX's query execution time for the `sum` is faster than UNLYNX, as we improved the $CTKS$ protocol by enabling its execution in a tree fashion, thus reducing its execution complexity from $O(\#CN)$ to $O(log(\#CN))$. Unlike UNLYNX, DRYNX enables the verification of DPs' value ranges, which, for the computation of a `sum`, adds an overhead of only 0.6 seconds (out of a total time of 2 seconds, as depicted in Figure 4.7a). However, DRYNX enables a faster scalable verification of proofs by an auditor. After the proofs are verified and the results stored in the *proof blockchain*, an auditor can simply request and verify the corresponding block, which in this case takes approximately 0.4s. In UNLYNX, an auditor has to request the proofs from each entity and verify them by itself, which takes 1.4s.

4.8 Conclusion

In this chapter, we have proposed DRYNX, a novel system that enables a querier to compute statistics and train machine-learning models on distributed datasets in a strong adversarial model where no entity is individually trusted. DRYNX provides query-execution auditability and ensures the end-to-end confidentiality of the data. It protects the privacy of the data providers and relies on an immutable and distributed ledger to provide efficient correctness verification and proofs storage. DRYNX is highly modular, offering configurable tradeoffs between security, privacy, and efficiency. Finally, DRYNX enables privacy-preserving computations of widely-used statistics on sensitive and distributed data, thus offering features that are absolutely needed in crucial areas such as user-behavior analysis or medical research.

5 Federated Learning

5.1 Introduction

In this chapter, we address the problem of privacy-preserving learning and prediction among multiple parties, i.e., data providers (DPs), that want to collaborate without sharing their data with each other. To address this issue, we design a solution that uses the MapReduce abstraction [60] that is often used to define distributed ML tasks [51; 239]. MapReduce defines parallel and distributed algorithms in a simple and well-known abstraction: PREPARE (data preparation), MAP (distributed computations executed independently by multiple nodes or machines), COMBINE (combination of the MAP results, e.g., aggregation) and REDUCE (computation on the combined results). We build on and extend this abstraction to determine and delimit which information, e.g., MAP outputs, have to be protected to design a decentralized **privacy-preserving** system for ML training and prediction. The model is locally trained by the DPs (MAP) and the results are iteratively combined (COMBINE) to update the global model (REDUCE). As before, we exploit the partitioned (distributed) data to enable DPs to keep control of their respective data, and we distribute the computation to provide an efficient solution for the training of ML models on confidential data. After the training, the model is kept secret from all entities and is obliviously and collectively used to provide predictions on confidential data that are known only to the entity requesting the prediction. We remark that differential-privacy-based federated-learning solutions [3; 42; 64; 114; 120; 125; 137; 188; 220; 234] follow the same model, i.e., they can be defined according to the MapReduce abstraction. However, most solutions introduce a trade-off between accuracy and privacy [119], and do not provide data and model confidentiality simultaneously. On the contrary, our solution uses a different paradigm in which, similarly to non-secure solutions, the accuracy is traded off with the performance (e.g., number of iterations), but not with privacy.

We propose SPINDLE (Scalable Privacy-preservINg Distributed LEarning), a system that enables the privacy-preserving, distributed (cooperative) execution of the widespread stochastic mini-batch gradient-descent (SGD) on data that are stored and controlled by multiple DPs. SPINDLE builds on a state-of-the-art multiparty, lattice-based, quantum-resistant cryptographic scheme to ensure data and model confidentiality, in the passive-adversary model in which all-but-one DPs can be dishonest. SPINDLE is meant to be a generic and widely-applicable system that supports the SGD-based training of many different ML models. This includes, but is not limited to, support vector machines, graphical models, generalized linear-models and neural networks [66; 101; 131; 233; 261]. For concreteness and comparison with existing works, we instantiate SPINDLE for the training of and prediction on generalized linear models (GLMs) [174], (e.g., linear, logistic and multinomial logistic regressions). GLMs are easily interpretable, capture complex non-linear relations (e.g., logistic regression), and are widely-used in many domains such as finance, engineering, environmental studies and healthcare [141]. In a realistic scenario where a dataset of 11,500 samples and 90 features is distributed among 10 DPs, SPINDLE efficiently trains a logistic regression model in less than 54 seconds, achieving an accuracy of 83.9% (See Table 5.2 in Section 5.7), equivalent to a non-secure centralized solution. The distribution of the workload enables SPINDLE to efficiently cope with a large number of DPs (parties), as its execution time is practically independent of it. SPINDLE handles a large number of features, by optimizing the use of the cryptosystem's packing capabilities, and by exploiting *single-instruction multiple-data (SIMD)* operations. It is able to perform demanding training tasks, with high number of iterations and thus high-depth computations, by relying on the multiparty cryptoscheme's ability to collectively refresh a ciphertext with no significant overhead. As shown by our evaluation, these properties enable SPINDLE to support training on large and complex data such as imaging or medical datasets. Moreover, SPINDLE scalability over multiple dimensions (features, DPs, data) represents a notable improvement with respect to DRYNX that we introduced in Chapter 4 and to state-of-the-art secure solutions [263].

In this chapter, we make the following contributions:

- We analyze the problem of privacy-preserving distributed training and of the evaluation of ML models by extending the widespread MapReduce abstraction with privacy constraints. Following this abstraction, we instantiate SPINDLE, the first operational and efficient distributed system that enables the privacy-preserving execution of a complete machine-learning workflow through the use of a cooperative gradient descent on a dataset distributed among many data providers.
- We propose multiple optimizations that enable the efficient use of a quantum-resistant multiparty (N-party) cryptographic scheme by relying on parallel computations, SIMD operations, efficient collective operations and optimized polynomial approximations of the models' activation functions, e.g., sigmoid and softmax.
- We propose a method for the parameterization of SPINDLE by capturing the relations among the security and the learning parameters in a graphical model.
- We evaluate SPINDLE against centralized and decentralized secure solutions and demonstrate its scalability and accuracy.

To the best of our knowledge, SPINDLE is the first operational system that provides the aforementioned features and security guarantees.

Acknowledgements. This work has been made possible by the help of Dr. Juan R. Troncoso-Pastoriza, Dr. Apostolos Pyrgelis, Sinem Sav, Joao Sa Sousa, Jean-Philippe Bossuat, and Prof. Jean-Pierre Hubaux.

5.2 Secure Federated Training and Evaluation

We first introduce the problem of privacy-preserving distributed training and evaluation of machine-learning (ML) models. Then, we present a high-level overview and architecture of a solution that satisfies the security requirements of the presented problem. In Section 5.3, we present SPINDLE, a system that enables the privacy preserving and distributed execution of a stochastic gradient-descent. We instantiate our solution for the training and evaluation of the widely-used Generalized Linear Models [174]. In the rest of this chapter, matrices are denoted by upper-case-bold characters and vectors by lowercase-bold characters; the i-th row of a matrix $\boldsymbol{X}$ is depicted as $\boldsymbol{X}[i,\cdot]$, and its i-th column as $\boldsymbol{X}[\cdot,i]$. Similarly, the i-th element of a vector $\boldsymbol{y}$ is denoted by $\boldsymbol{y}[i]$. We provide a list of recurrent symbols in Table 1.3.

5.2.1 Problem Statement

We consider a setting where a dataset $(\boldsymbol{X}_{n\times c}, \boldsymbol{y}_n)$, with $\boldsymbol{X}_{n\times c}$ a matrix of n records and c features, and $\boldsymbol{y}_n$ a vector of n labels, is distributed among a set of data providers, i.e., $S = \{DP_1,\ldots,DP_{|S|}\}$. The dataset is horizontally partitioned, i.e., each data provider DP_i holds a partition of n_i samples $(\boldsymbol{X}^{(i)}, \boldsymbol{y}^{(i)})$, with $\sum_{i=1}^{|S|} n_i = n$. A querier, which can also be a data provider (DP), requests the training of a ML model on the distributed dataset $(\boldsymbol{X}_{n\times c}, \boldsymbol{y}_n)$ or the evaluation of an already trained model on its input $(\boldsymbol{X'},\cdot)$.

We assume that the DPs are willing to contribute their respective data to train and to evaluate ML models on the distributed dataset. To this end, DPs are all interconnected and organized in a topology that enables efficient execution of the computations, e.g., in a tree structure as depicted in Figure 5.1. Even though the DPs wish to collaborate for the execution of ML workflows, they do not trust each other. As a result, they seek to protect the confidentiality of their data (used for training and evaluation) and of the collectively learned model. More formally, we require that the following privacy properties hold in a passive-adversary model in which all-but-one DPs can collude, i.e., the DPs follow the protocol, but up to $|S|-1$ DPs might share among them the information they observe during the execution, to extract information about the other DPs' inputs.

(a) Data Confidentiality: The training data of each data provider DP_i, i.e., $(\boldsymbol{X}^{(i)}, \boldsymbol{y}^{(i)})$ and the querier's evaluation data $(\boldsymbol{X'},\cdot)$ should remain only known to their respective owners. To this end, data confidentiality is satisfied as long as the involved parties (DPs and querier) do not obtain any information about other parties' inputs other than what can be deduced from the output of the process of training or evaluating a model.

(b) Model Confidentiality: During the training process, no data provider DP_i should gain more information about the model that is being trained than what it can learn from its own input data $(\boldsymbol{X}^{(i)}, \boldsymbol{y}^{(i)})$. During prediction, the querier should not learn anything more about the model than what it can infer from its input data $(\boldsymbol{X}', \cdot)$ and the corresponding predictions $\boldsymbol{y}'$.

We remark here that, in contrary to the work presented in Chapter 4, input correctness and computation correctness are not part of the problem requirements, i.e., we assume that DPs input correct data and do not perform wrong computations. We discuss possible countermeasures against malicious DPs in Section C.4.1.

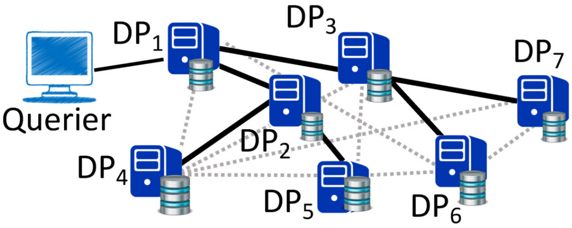


Figure 5.1 – SPINDLE's Model. Thick arrows represent a possible (efficient) query-execution flow.

5.2.2 Solution Overview

To address the problem of privacy-preserving distributed learning, we leverage the MapReduce abstraction, which is often used to capture the parallel and repetitive nature of distributed learning tasks [51; 239]. We complement this abstraction with a protection mechanism $P(\cdot)$; $P(x)$ denotes that value x has to be protected to satisfy data and model confidentiality (Section 5.2.1). We present the extended MapReduce abstraction in Protocol 5.1. In PREPARE, the data providers ($DP_i \in S$) pre-process their data $(\boldsymbol{X}^{(i)}, \boldsymbol{y}^{(i)})$, they agree on the learning parameters and on one data provider that plays the role of DP_R and is then responsible for the execution of REDUCE.

Protocol 5.1 Extended MapReduce Abstraction.

```
TRAINING: $S$ receives query from Querier and outputs $P(\boldsymbol{W}_G^{(\cdot,g)})$
Each $DP_i$ has $(\boldsymbol{X}^{(i)}, \boldsymbol{y}^{(i)})$
DPs appoint $DP_R$ and agree on learning params.                              – PREPARE
Each $DP_i \in S$ initializes its local model $\boldsymbol{W}^{(i,0)}$

for $j = 1, \dots, g$ do
    Each $DP_i \in S$ computes:                                               – MAP
    $P(\boldsymbol{W}^{(i,j)}) \leftarrow \mathrm{Map}((\boldsymbol{X}^{(i)}, \boldsymbol{y}^{(i)}), P(\boldsymbol{W}_G^{(\cdot,j-1)}), P(\boldsymbol{W}^{(i,j-1)}))$

    Each $DP_i$ sends $P(\boldsymbol{W}^{(i,j)})$ to $DP_R$                           – COMBINE
    $DP_R$: $P(\boldsymbol{W}^{(\cdot,j)}) \leftarrow C(P(\boldsymbol{W}^{(i,j)})), \forall\, DP_i \in S$
    $DP_R$: $P(\boldsymbol{W_G}^{(\cdot,j)}) \leftarrow \mathrm{Red}(P(\boldsymbol{W_G}^{(\cdot,j-1)}), P(\boldsymbol{W}^{(\cdot,j)}))$       – REDUCE
end for

PREDICTION: $DP_R$ receives $P(\boldsymbol{X}')$ from Querier and uses $P(\boldsymbol{W_G}^{(\cdot,g)})$ to compute $P(\boldsymbol{y}')$ that is
    sent back to the Querier
```

In Figure 5.1, DP_1 plays DP_R's role. As explained later, DP_R only manipulates protected data and is subject to the same security constraints as any other DP. We discuss the choice of DP_R and its availability in Appendix C.4. Each DP_i then iteratively (g iterations) trains its local model ($P(\boldsymbol{W}^{(i,j)})$ at iteration j) on its data in MAP. For this purpose, DP_i relies on its local data, the current global model $P(\boldsymbol{W}_G^{(\cdot,j)})$ and its previous local model $P(\boldsymbol{W}_G^{(i,j-1)})$. They combine their local models in COMBINE (through an application-dependent function $C(\cdot)$) to obtain the combination of DPs' local models at iteration j, i.e., $P(\boldsymbol{W}^{(\cdot,j)}$. Then, they update the global model $P(\boldsymbol{W}_G^{(\cdot,j)})$ in REDUCE. To capture the complete ML workflow, we extend the MapReduce architecture with a PREDICTION phase in which predictions $P(\boldsymbol{y}')$ are computed from the querier's protected evaluation data $P(\boldsymbol{X}')$ by using the (protected) global model $P(\boldsymbol{W_G}^{(\cdot,g)})$ obtained during the training.

5.3 SPINDLE Design

Following the extended MapReduce abstraction described in Section 5.2.2, we design a system, named SPINDLE, that enables the privacy-preserving execution of the widely applicable cooperative gradient descent [241; 242] – which is used to minimize many cost functions in machine-learning [131; 233; 261]. We instantiate this system for the training of and prediction on Generalized Linear Models [174]. We introduce these concepts in Section 5.3.1. To implement the protection mechanism $P(\cdot)$, SPINDLE builds on a multiparty fully homomorphic encryption scheme introduced in Section 1.4. Then, in Section 5.3.2, we describe how SPINDLE instantiates the phases of the extended MapReduce abstraction and how we address the collective data-processing on the distributed dataset through secure and interactive protocols. We demonstrate how training is performed, notably by executing the gradient descent operations under homomorphic encryption, and how predictions are executed on encrypted models. We present the detailed cryptographic operations in Section 5.4 and analyze SPINDLE's security in Section 5.6.

5.3.1 Background

Cooperative Gradient-Descent. We rely on a distributed version of the popular mini-batch stochastic gradient-descent (SGD) [131; 233; 261]. In the standard version of SGD, the goal is to minimize $\min_{\boldsymbol{w}}[F(\boldsymbol{w}) := (1/n)\sum_{\phi=1}^{n} f(\boldsymbol{w};\ \boldsymbol{X}[\phi,\cdot])]$, where $f(\cdot)$ is the loss function defined by the learning model, $\boldsymbol{w} \in \mathbb{R}^c$ are the model parameters, and $\boldsymbol{X}[\phi,\cdot]$ is the ϕ^{th} data sample (row) of $\boldsymbol{X}$. The model is then updated by m iterations $\boldsymbol{w}^{(l)} = \boldsymbol{w}^{(l-1)} - \alpha[\zeta(\boldsymbol{w}^{(l-1)};\ \boldsymbol{B}^{(l)})]$, for $l = 1,\ldots,m$, with α the learning rate, $\boldsymbol{B}^{(l)}$ a randomly sampled sub-matrix of $\boldsymbol{X}$ of size $b \times c$, and $\zeta(\boldsymbol{w};\boldsymbol{B}) = \boldsymbol{B}^T(\sigma(\boldsymbol{B}\boldsymbol{w}) - I(\bar{\boldsymbol{y}}))$, where $\bar{\boldsymbol{y}}$ is the vector of labels corresponding to the batch $\boldsymbol{B}$. The activation function σ and $I(\cdot)$ are both model-dependent, e.g., for a logistic regression σ is the sigmoid and $I(\cdot)$ is the identity.

We rely on the **cooperative SGD** (CSGD) proposed by Wang and Joshi [241; 242], due to its properties; in particular: (i) modularity, as it can be synchronous or asynchronous, and can be combined with classic gradient-descent convergence optimizations such as Nesterov accelerated SGD [175]; (ii) applicability, as it accommodates any ML model that can be trained with SGD and enables the distribution of any SGD based solution; (iii) it guarantees a bound on the error-convergence depending on the distributed parameters; e.g., the number of iterations and the update function for the global weights [33; 241; 242; 260]; and (iv) it has been shown to work well even in the case of non-independent-and-identically-distributed (non-i.i.d.) data partitions [152; 241; 242]. The data providers (DPs), each of which owns a part of the dataset, locally perform multiple iterations of the SGD before aggregating their model weights into the global model weights. The global weights are included in subsequent local DP computations to avoid that they learn, or descend, in the wrong direction. For simplicity, we present SPINDLE with the synchronous CSGD version, where the DPs perform local model updates simultaneously. For each DP_i, the local update rule at global iteration j and local iteration l is:

$$\boldsymbol{w}^{(i,j,l)} = \boldsymbol{w}^{(i,j,l\text{-}1)} - \alpha\zeta(\boldsymbol{w}^{(i,j,l\text{-}1)};\boldsymbol{B}^{(l)}) - \alpha\rho(\boldsymbol{w}^{(i,j,l\text{-}1)} - \boldsymbol{w}_G^{(\cdot,j\text{-}1)}), \tag{5.1}$$

where $\boldsymbol{w}_G^{(\cdot,j-1)}$ are the global weights from the last global update iteration $j-1$, α is the learning rate and ρ, the elastic rate, is the parameter that controls how much the data providers can diverge from the global model. The set of DPs S performs z local iterations between each update of the global model that is updated at global iteration j with a moving average by:

$$\boldsymbol{w}_G^{(\cdot,j)} = (1 - |S|\alpha\rho)\boldsymbol{w}_G^{(\cdot,j-1)} + \alpha\rho\sum_{i=0}^{|S|}\boldsymbol{w}^{(i,j,z)}. \tag{5.2}$$

Generalized Linear Models (GLMs). GLMs [174] are a generalization of linear models where the linear predictor, i.e., the combination $\boldsymbol{X}\boldsymbol{w}$ of the feature matrix $\boldsymbol{X}$ and weights vector $\boldsymbol{w}$, is related to a vector of class labels $\boldsymbol{y}$ by an activation function σ such that $E(\boldsymbol{y}) = \sigma^{-1}(\boldsymbol{X}\boldsymbol{w})$, where $E(\boldsymbol{y})$ is the mean of $\boldsymbol{y}$. In this work, we consider the widely-used linear (i.e., $\sigma(\boldsymbol{X}\boldsymbol{w}) = \boldsymbol{X}\boldsymbol{w}$), logistic (i.e., $\sigma(\boldsymbol{X}\boldsymbol{w}) = 1/(1+e^{-\boldsymbol{X}\boldsymbol{w}})$) and multinomial (i.e., $\sigma(\boldsymbol{X}\boldsymbol{w}_\lambda) = e^{\boldsymbol{X}\boldsymbol{w}_\lambda}/(\sum_{j\in cl} e^{\boldsymbol{X}\boldsymbol{w}_j})$, for $\lambda \in cl$) regression models. We remark that for multinomial regression, the weights are represented as a matrix $\boldsymbol{W}_{c\times|cl|}$, where c is the number of features, cl is the set of class labels and $|cl|$ its cardinality. In the rest of the chapter, unless otherwise stated, we define the operations on a single vector of weights $\boldsymbol{w}$ and we note that in the case of multinomial regression, they are replicated on the $|cl|$ vectors of weights, i.e., each column of $\boldsymbol{W}_{c\times|cl|}$.

Multiparty Homomorphic Encryption. For the protection mechanism of SPINDLE, we rely on the multiparty fully-homomorphic encryption scheme introduced in Section 1.4. With this scheme, each data provider can independently compute on ciphertexts encrypted under the collective public key pk but, as the secret is distributed among all DPs, they have to all collaborate in order to decrypt data. In SPINDLE, the data providers rely on this mechanism to train a collectively encrypted model before switching its encryption, without decrypting the data, to the querier's public key such that only the querier can decrypt the result. This multiparty scheme is based on the Cheon-Kim-Kim-Song cryptosystem (CKKS) [47] that enables approximate arithmetic, and whose security is based on the ring learning with errors (RLWE) problem [150]. A plaintext/ciphertext encodes a vector of up to $N/2$ values, which enables SPINDLE to efficiently perform parallelized operations on encrypted vectors of values. As introduced in Section 1.4 and explained in the following sections, SPINDLE also relies on the multiparty construction of the cryptoscheme to replace heavy cryptographic operations, e.g., refresh or bootstrap of a ciphertext, by lightweight interactive protocols.

5.3.2 SPINDLE Protocols

We first describe SPINDLE's operations for training a Generalized Linear Model following Protocol 5.1. In this case, the model $\boldsymbol{W}$ is a vector of weights that we denote by $\boldsymbol{w}$, and MAP corresponds to multiple local iterations of the gradient descent. Recall that in the case of multinomial regression, all operations are repeated for each label class $\lambda \in cl$.

5.3.2.1 TRAINING

PREPARE. The data providers (DPs) collectively agree on the training parameters: the maximum number of global g and local z iterations, and the learning parameters $lp = \{\alpha, \rho, b\}$, where α is the learning rate, ρ the elastic rate, and b the batch size. The DPs also collectively initialize the cryptographic keys for the distributed CKKS scheme by executing DKeyGen(·) (see Section 1.4). Then, the DPs initialize their local weights and pre-compute operations that involve only their input data ($\alpha\boldsymbol{X}^{(i)} I(\boldsymbol{y}^{(i)})$ and $\alpha\boldsymbol{X}^{(i)T}$). We discuss in Appendix C.4 how the DPs can collaborate to standardize or normalize the distributed dataset (if needed) and check that their respective inputs are consistent, e.g., they have data distribution homogeneity.

Protocol 5.2 MAP.

```
Each $DP_i$ outputs $\langle \boldsymbol{w}^{(i,j)} \rangle \leftarrow$ Map $((\boldsymbol{X}^{(i)}, \boldsymbol{y}^{(i)}), \langle \boldsymbol{w}_G^{(\cdot,j-1)} \rangle, \langle \boldsymbol{w}^{(i,j-1)} \rangle)$
1: $\langle \boldsymbol{w}^{(i,j,0)} \rangle = \langle \boldsymbol{w}^{(i,j-1)} \rangle$
2: for $l = 1, \dots, z$ do
3:     Select batch $(\boldsymbol{B}, \tilde{\boldsymbol{y}})$ of $b$ rows in $(\boldsymbol{X}^{(i)}, \boldsymbol{y}^{(i)})$
4:     $\langle \boldsymbol{u}[k] \rangle = \boldsymbol{B}[k,\cdot] \bullet \langle \boldsymbol{w}^{(i,j,l-1)} \rangle$, for $k = 1, \dots, b$
5:     $\langle \boldsymbol{v}[e] \rangle = \alpha \boldsymbol{B}[\cdot,e]^T \bullet \sigma(\langle \boldsymbol{u} \rangle)$, for $e = 1, \dots, c$
6:     $\boldsymbol{\mu}[e] = \sum_{k=1}^{b} \alpha \boldsymbol{B}[\cdot,e]^T I(\tilde{\boldsymbol{y}}[k])$, for $e = 1, \dots, c$
7:     $\langle \boldsymbol{w}^{(i,j,l)} \rangle = \langle \boldsymbol{w}^{(i,j,l-1)} \rangle + \boldsymbol{\mu}\text{-}\langle \boldsymbol{v} \rangle - \alpha\rho(\langle \boldsymbol{w}^{(i,j,l-1)} \rangle - \langle \boldsymbol{w}_G^{(\cdot,j-1)} \rangle)$
8: end for
9: $\langle \boldsymbol{w}^{(i,j)} \rangle = \mathrm{RR}(\langle \boldsymbol{w}^{(i,j,z)} \rangle)$
```

MAP. As depicted in Protocol 5.2, the DPs execute z iterations of the cooperative gradient-descent local update (Section 5.3.1). The local weights of DP_i (i.e., $\langle \boldsymbol{w}^{(i,j,l-1)} \rangle$) are updated at a global iteration j and a local iteration l by computing the gradient (Protocol 5.2, lines 4, 5, and 6) that is then combined with the current global weights $\langle \boldsymbol{w}_G^{(\cdot,j-1)} \rangle$ (Protocol 5.2, line 7) following Equation 5.1. Remember that $\bullet$ is the dot product (see Section 1.4. These computations are performed on batches of b samples and c features. We detailed these operations between matrices and encrypted vectors in Section 5.4.2. To ensure that the update of DP_i's local weights, i.e., the link between the ciphertexts $\langle \boldsymbol{w}^{(i,j-1)} \rangle = \langle \boldsymbol{w}^{(i,j,0)} \rangle$ and $\langle \boldsymbol{w}^{(i,j,z)} \rangle$, does not leak information about the DP's local data, $\langle \boldsymbol{w}^{(i,j,z)} \rangle$ is re-randomized $\mathrm{RR}(\cdot)$ at the end of MAP (Protocol 5.2, line 8), i.e., DP_i adds to it a fresh encryption of 0. Note that in Protocol 5.2, line 5 the activation function $\sigma(\cdot)$ is computed on the encrypted vector $\langle \boldsymbol{u} \rangle$ (or a matrix $\langle \boldsymbol{U} \rangle$ in the case of multinomial). The exponential activation functions for logistic (i.e., sigmoid) and multinomial (i.e., softmax) regressions have to be approximated to polynomial functions to be evaluated on encrypted data by using the homomorphic properties of CKKS. We rely on a least-square polynomial approximation (LSPA) for the sigmoid, as it provides an optimal average mean-square error for uniform inputs in a specific interval, which is a reasonable assumption when the input distribution is not known. For softmax, we rely on Chebyshev approximation (CA) to minimize the maximum approximation error and thus avoid that the function diverges on specific inputs. The approximation intervals can be empirically determined by using synthetic datasets with distribution similar to the real ones, by computing the minimum and maximum input values over all DPs and features, or by relying on estimations based on the data distribution [109].

Algorithm 5.1 takes as input an encrypted vector/matrix $\langle \boldsymbol{u} \rangle$ or $\langle \boldsymbol{U} \rangle$ and the type of the regression t (i.e., linear, logistic or multinomial). If t is linear, the protocol simply returns $\langle \boldsymbol{u} \rangle$. Otherwise, if t is logistic, it computes the activated vector $\langle \sigma(\boldsymbol{u}) \rangle$ by using the sigmoid's LSPA (apSigmoid($\cdot$)). If t is multinomial, it computes the activated matrix $\langle \sigma(\boldsymbol{U}) \rangle$ using the softmax approximation that is computed by the multiplication of two CAs, one for the nominator e^x (apSoftN($\cdot$)) and one for the denominator $\frac{1}{\sum e^{x_j}}$ (apSoftD($\cdot$)), each computed on different

Algorithm 5.1 Activation Function $\sigma(\cdot)$.

```
Func. $\sigma(\langle \boldsymbol{u} \rangle$ or $\langle \boldsymbol{U} \rangle, t)$ returns the activated $\langle \sigma(\boldsymbol{u}) \rangle$ or $\langle \sigma(\boldsymbol{U}) \rangle$
if $t$ is Linear then
    $\langle \sigma(\boldsymbol{u}) \rangle = \langle \boldsymbol{u} \rangle$
else if $t$ is Logistic then
    $\langle \sigma(\boldsymbol{u}) \rangle = \mathrm{apSigmoid}(\boldsymbol{u})$
else if $t$ is Multinomial, input is a matrix $\langle \boldsymbol{U}_{c \times |cl|} \rangle$ then
    $\langle \boldsymbol{m} \rangle = \mathrm{apMax}(\langle \boldsymbol{U} \rangle)$
    for $\lambda \in cl$ do
        $\langle \boldsymbol{U'}[\lambda, \cdot] \rangle = \langle \boldsymbol{U}[\lambda, \cdot] \rangle - \langle \boldsymbol{m} \rangle$
        $\langle \sigma(\boldsymbol{U}[\lambda, \cdot]) \rangle = \mathrm{M}(\mathrm{apSoftN}(\langle \boldsymbol{U'}[\lambda, \cdot] \rangle), \mathrm{apSoftD}(\langle \boldsymbol{U'}[\lambda, \cdot] \rangle))$
    end for
end if
```

intervals. The polynomial approximation computation is detailed in Algorithm C.1 (Appendix C.1). To avoid an explosion of the exponential values in the softmax, a vector $\langle \boldsymbol{m} \rangle$ that contains the approximated max (apMax($\cdot$)) value of each column of $\langle \boldsymbol{U} \rangle$ is subtracted from all input values, i.e., from each $\langle \boldsymbol{U}[\lambda, :] \rangle$ with $\lambda = 0, \ldots, |cl|$. Similar to softmax, the approximation of the max function requires two CAs, and is detailed in Appendix C.1.

COMBINE. The MAP outputs of each DP_i, i.e., $\langle \boldsymbol{w}^{(i,j)} \rangle$, are homomorphically combined ascending a tree structure, such that each DP_i aggregates its encrypted updated local weights with those of its children and sends the result to its parent. In this case, the combination function $C(\cdot)$ is the homomorphic addition operation. At the end of this phase, the DP at the root of the tree DP_R obtains the encrypted combined weights $\langle \boldsymbol{w}^{(\cdot,j)} \rangle$.

REDUCE. DP_R updates the encrypted global weights $\langle \boldsymbol{w}_G^{(\cdot,j)} \rangle$, as shown in Protocol 5.3. More precisely, it computes Equation 5.2 by using the encrypted sum of the DPs' updated local weights $\langle \boldsymbol{w}^{(\cdot,j)} \rangle$ (obtained from COMBINE), the previous global weights $\langle \boldsymbol{w}_G^{(\cdot,j-1)} \rangle$, the predefined elastic rate ρ and the learning rate α. After g iterations of the MAP, COMBINE, and REDUCE, DP_R obtains the encrypted global model $\langle \boldsymbol{w}_G^{(\cdot,g)} \rangle$ and broadcasts it to the rest of the DPs.

Protocol 5.3 REDUCE.

```
$DP_R$ computes $\langle \boldsymbol{w}_G^{(\cdot,j)} \rangle \leftarrow \mathrm{Red}(\langle \boldsymbol{w}_G^{(\cdot,j-1)} \rangle, \langle \boldsymbol{w}^{(\cdot,j)} \rangle, \rho, \alpha)$
$\langle \boldsymbol{w}_G^{(\cdot,j)} \rangle = (1 - \alpha\rho|S|)\langle \boldsymbol{w}_G^{(\cdot,j-1)} \rangle + \alpha\rho\langle \boldsymbol{w}^{(\cdot,j)} \rangle$
```

5.3.2.2 PREDICTION

The querier's input data $(\boldsymbol{X'}, \cdot)$ is encrypted with the collective public key pk. Then, $\langle \boldsymbol{X'} \rangle_{pk}$ is multiplied (dot product $\bullet$) with the weights of the trained model $\langle \boldsymbol{w}_G^{(\cdot,g)} \rangle$ and processed through the activation function $\sigma(\cdot)$ to obtain the encrypted prediction values $\langle \boldsymbol{y'} \rangle$ (one prediction per row of $\boldsymbol{X'}$). The prediction results encrypted under pk are then collectively switched by the DPs to the querier public key pk' using DKeySwitch$(\cdot)$, so that only the querier can decrypt $\langle \boldsymbol{y'}_{pk'} \rangle$.

Protocol 5.4 PREDICTION.

```
$DP_R$ gets $\langle \boldsymbol{X'}_{n' \times c} \rangle$ from Querier and computes $\langle \boldsymbol{y'}_{n'} \rangle$ using $\langle \boldsymbol{w}_G^{(\cdot,g)} \rangle$
1: $\langle \boldsymbol{y'}[p] \rangle = \sigma(\langle \boldsymbol{X'}[p,\cdot] \rangle \bullet \langle \boldsymbol{w}_G^{(\cdot,g)} \rangle)$, for $p = 0, \ldots, n'$
2: $\langle \boldsymbol{y'} \rangle_{pk'} = \mathrm{DKeySwitch}(\langle \boldsymbol{y'} \rangle, pk', \{sk_i\})$
```

5.4 System Operations

We describe how SPINDLE relies on the properties of the distributed version of CKKS to efficiently address the problem of privacy-preserving distributed learning. We first describe how we optimize the protocols of Section 5.3.2 by choosing when to execute cryptographic operations such as rescaling and (distributed) bootstrapping. Then, we discuss how to efficiently perform the MAP protocol that involves a sequence of vector-matrix-multiplications and the evaluation of the activation function, in the encrypted domain.

5.4.1 Cryptographic Operations

As explained in Section 1.4, ciphertext multiplications incur the execution of other cryptographic operations hence increase SPINDLE's computation overhead. This overhead can rapidly increase when the same ciphertext is involved in sequential operations, i.e., when the operations' multiplicative depth is high. As we will describe in Section 5.7, SPINDLE relies on the Lattigo [159] lattice-based cryptographic library, where a ciphertext addition or multiplication requires a few ms, whereas Rescale$(\cdot)$, Relin$(\cdot)$, and DBootstrap$(\cdot)$, are 1-order, 2-orders, and 1.5-orders of magnitude slower than the addition, respectively. These operations can be computationally heavy, hence their execution in the protocols should be optimized. Note that we avoid the use of the centralized traditional bootstrapping, as it would require a much more conservative parameterization for the same security level, resulting in higher computational overheads (see Section 5.7).

Lazy Rescaling. To maintain the precision of the encrypted values and for efficiency we rescale a ciphertext $\{\langle \boldsymbol{v} \rangle, \tau, \Delta\}$ only when Δ is close to the current ciphertex-level modulus Q_τ (see Section 1.4). Hence, we perform a ReScale$(\cdot)$ only when this condition is met after a series of consecutive operations.

Relinearization. Letting the ciphertext size increase after every multiplication would add to the subsequent operations an overhead that is higher than the relinearization. Hence, to maintain the ciphertext size and degree constant, a Relin(·) operation is performed after each ciphertext-ciphertext multiplication. We here note that a Relin(·) operation can be deferred if doing so incurs in lower computational complexity (e.g., if additions performed after the ciphertext-ciphertext multiplications reduce the number of ciphertexts to relinearize).

Bootstrapping. In the protocols of Section 5.3.2, we observe that the data providers' local weights and the model global weights ($\langle \boldsymbol{w} \rangle$ and $\langle \boldsymbol{w}_G \rangle$, resp.) are the only persistent ciphertexts over multiple computations and iterations. They are therefore the only ciphertexts that need to be bootstrapped, and we consider three approaches for this. With **Local bootstrap (LB)**, each data provider (DP) bootstraps (calling a DBootstrap(·) protocol) its local weights, every time they reach level τ_b during the MAP local iterations and before the COMBINE. As a result, the global weights are always combined with fresh encryptions of the local weights and only need to be bootstrapped after multiple REDUCE. Indeed, REDUCE involves a multiplication by a constant hence a Rescale(·). With **Global bootstrap (GB)**, we use the interdependency between the local and global weights, and we bootstrap only the global weights and assign them directly to the local weights. The bootstrapping is performed on the global weights during REDUCE. Thus, we modify TRAINING so that MAP operates on the (bootstrapped) global weights, i.e., $\langle \boldsymbol{w}^{(i,j-1)} \rangle = \langle \boldsymbol{w}_G{}^{(\cdot,j-1)} \rangle$, for a DP_i at global iteration j. By following this approach, the number of bootstrap operations is reduced, with respect to the local approach, because it is performed by only one DP and depends only on the number of global iterations. However, it modifies the learning method, and it offers less flexibility, as the number of local iterations in MAP is constrained by the number of ciphertext multiplications required in each iteration and by the available ciphertext levels. With **Hybrid bootstrap (HB)**, both GB and LB approaches are combined to reduce the total number of bootstrapping operations. The global weights are bootstrapped at each global iteration (GB) and the DPs can still perform many local iterations by relying on the LB. In our experiments (Section 5.7.2), we observed that the effect on the trained model's accuracy depends mainly on the data and that, in most cases, enabling DPs to perform more local iterations (LB and HB) between two global updates yields better accuracy. Even though LB incurs at least $|S|$ more executions of the DBootstrap(·), the DPs execute them in parallel and thus amortize the overhead on SPINDLE's execution time. However, if the training of a dataset requires frequent global updates, then GB (or HB) achieves a better trade-off, see Section 5.7.2. Taking into account these cryptographic transformations and the strategy to optimize their use in SPINDLE, we explain how to optimize the required number of ciphertext operations.

5.4.2 MAP Vector-Matrix Multiplications

As described in Section 1.4, each CKKS ciphertext encrypts (or packs) a vector of values, e.g., 8,192 elements if the ring dimension is $N = 2^{14}$. This packing enables us to simultaneously perform operations on all the vector values, by using a Single-Instruction Multiple Data (SIMD) approach for parallelization. To execute computations among values stored in different slots of

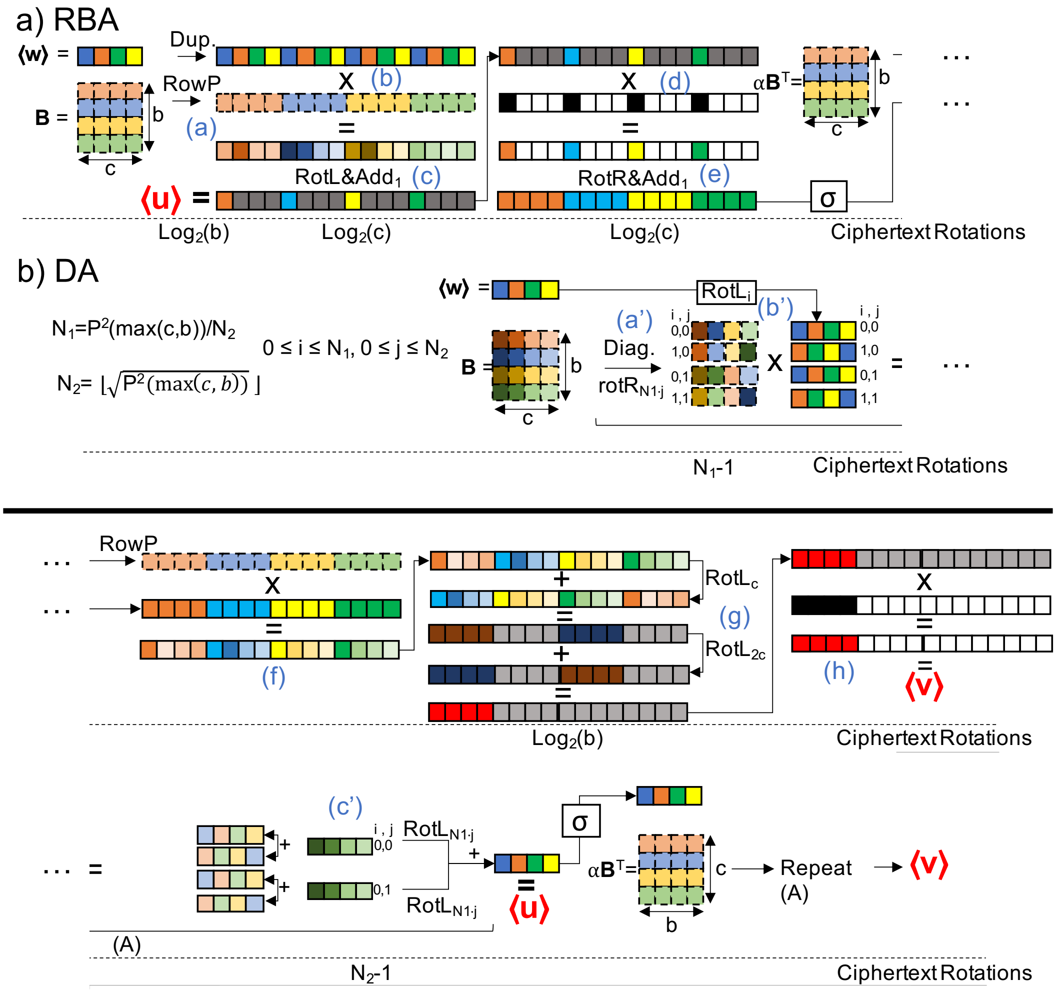


Figure 5.2 – Packing approaches ((a) RBA: Row-Based Approach & (b) DA: Diagonal Approach) for executing Protocol 5.2, lines 4 and 5. We assume that $c \cdot b < N/2$ and show an example with $c = b = 4$. Dash elements are plaintext values, everything else is encrypted. Dup duplicates and adds, rowP packs the rows in one ciphertext, RotL(/R)&Add$_i$ rotates the encrypted vector by $i, 2i, 4i, \ldots$ to the left(/right) and at each step, aggregates the result with the previous ciphertext, RotL(/R)$_j$ rotates a vector left(/right) by j positions. $\mathrm{P}^2(x)$ returns the next power of 2 larger than x.

the same ciphertext, e.g., an inner sum, we rely on ciphertext rotations that have a computation cost similar to a relinearization (Relin(·)). Recall that for the execution of stochastic gradient-descent, each local iteration in MAP involves two sequential multiplications between encrypted vectors and cleartext matrices (Protocol 5.2, lines 4 and 5). As a result, packing is useful for reducing the number of vector multiplications and rotations needed to perform these operations. To this end, SPINDLE integrates two packing approaches and automatically selects the most appropriate approach at each DP during the training. We now describe these two approaches and how to choose between them, depending on the settings, i.e., the learning parameters, the number of features, and the DP computation capabilities. Figure 5.2 depicts SPINDLE's packing approaches for a toy example of the computation of $\langle \boldsymbol{u} \rangle$ (Protocol 5.2, line 4) whose result is activated (i.e., $\sigma(\langle \boldsymbol{u} \rangle)$) before used in the computation of $\langle \boldsymbol{v} \rangle$ (Protocol 5.2, line 5), for a setting with $c = b = 4$. For clarity, we assume that a vector of c (number of features) or b (batch size) elements can be encoded in one ciphertext (or plaintext), i.e., $\max(c, b) \leq N/2$.

Row-Based Approach (RBA). This approach was proposed by Kim et al. [124]. The input matrices ($\boldsymbol{B}$ and $\alpha \boldsymbol{B}^T$) are packed row-wise, and multiple rows are packed in one plaintext ((a) in the upper part of Figure 5.2), i.e., the number of plaintexts required to encode the input matrix is $\lceil \frac{c \cdot b \cdot 2}{N} \rceil$. Each plaintext is then multiplied with a ciphertext containing the replicated (or duplicated, Dup) weights' vector (b), such that the number of replicas is equal to the number of rows in $\boldsymbol{B}$. The multiplication of all the rows is then performed in a single operation. To obtain the results of the dot products between each weights' vector and row of $\boldsymbol{B}$, a partial inner sum is performed by adding the resulting ciphertext with rotated versions of itself (c). The values in between the dot product results are eliminated (i.e., masked) through a multiplication with a binary vector (d), and the dot product results are duplicated in the ciphertext (e) such that it can be activated ($\sigma(\cdot)$) and used directly for the multiplication with $\alpha \boldsymbol{X}^T$ (f). The result is then rotated and added to itself (g) such that it can be masked (h) to obtain $\langle \boldsymbol{v} \rangle$. As shown in Figure 5.2, the total number of vector multiplications is $\lceil \frac{c \cdot b \cdot 2}{N} \rceil \cdot 4$, whereas the number of ciphertext rotations is $\lceil \frac{c \cdot b \cdot 2}{N} \rceil \cdot 2 \cdot (log_2(b) + log_2(c))$. This approach has a multiplicative depth of $a_m + 4$, where a_m denotes the depth of the activation function $\sigma(\cdot)$.

Diagonal Approach (DA). This approach was presented by Halevi and Shoup [105] as an optimized homomorphic vector-matrix-multiplication evaluation. It optimizes the number of ciphertext rotations by transforming the input plaintext matrix $\boldsymbol{B}$. In particular, $\boldsymbol{B}$ is diagonalized, and each line is rotated ((a') in lower part of Figure 5.2) so that they can be independently multiplied with the (rotated) weights' vector (b'). The resulting ciphertexts are aggregated and rotated to obtain $\langle \boldsymbol{u} \rangle$ (c'), and a similar approach is used to compute $\langle \boldsymbol{v} \rangle$ after the activation. As shown in Figure 5.2, DA only executes $2 \cdot ((N_1 - 1) + (N_2 - 1))$ rotations on the encrypted vector, with $N_1 = P^2(\max(c, b))/N_2$ and $N_2 = \lfloor \sqrt{P^2(\max(c, b))} \rfloor$, where $P^2(x)$ returns the next power of 2 larger than x. This approach involves $N_1 \cdot N_2$ plaintext-ciphertext multiplications on independent ciphertexts and does not require any masking, which results in a multiplicative depth of $a_m + 2$. Therefore, this approach consumes fewer levels than RBA.

In both approaches, the number of rotations and multiplications depends on the batch size b and the number of features c. DA almost always requires more multiplications than RBA and uses more rotations after a certain c (e.g., if $b = 8$, the break-even happens at $c = 64$). However, as DA is *embarrassingly parallelizable* for both multiplications and rotations (with rotations being the most time-consuming operations), the computations can be amortized on multiple threads. Taking this into account, SPINDLE automatically chooses, based on c, b, and the number of available threads, the best approach at each DP. We analyze these trade-offs in Section 5.7.

5.4.3 Optimized Activation Function

As described in Section 5.3.2, to enable their execution under FHE, we approximate the sigmoid (apSigmoid(·)) and softmax (apMax(·), apSoftN(·), apSoftD(·)) activation functions with least-squares and Chebyshev polynomial approximations (PA), respectively. We adapt the baby-step giant-step algorithm introduced by Han and Ki [106] to enable the minimum-complexity computation of degree-d polynomials (multiplicative depth of $\lceil log_2(d) \rceil$ for $d \leq 7$, and with depth $\lceil log_2(d) + 1 \rceil$ otherwise). Protocol C.1 (in Appendix) computes the (element-wise) exponentiation of the encrypted input vector before recursively computing the polynomial approximation.

5.5 System Configuration

We discuss how to parameterize SPINDLE by taking into account the interdependencies between the input data, and the learning and cryptographic parameters. We then discuss two modular functionalities of SPINDLE, namely *data outsourcing* and *model release*.

Parameter Selection. SPINDLE relies on the configuration of (a) cryptographic parameters that determine its security level, and (b) learning parameters that affect the accuracy of the training and evaluation of the models. Both are tightly linked, and we capture these relations in a graph-based model, displayed in Figure 5.3, where vertices and edges represent the parameters and their interdependence, respectively. For simplicity, we present a directed graph that depicts our empirical method for choosing the parameters (see Section 1.7 for notation symbols). We note here that the corresponding non-directed graph is more generic and simply captures the main relations among the parameters. We observe two main clusters: the cryptographic parameters on the upper part of the graph (dotted circles), and the learning parameters (circles) on the lower one. The input data and their intrinsic characteristics, i.e., the number of features c or precision (bits of precision required to represent the data), are connected with both clusters that are also interconnected through the plaintext scale Δ. As such, there are various ways to configure the overall system parameters.

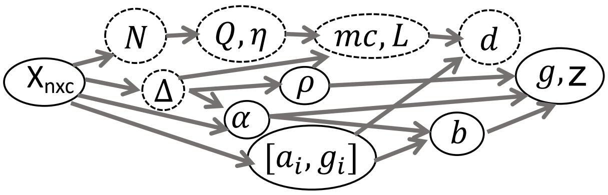


Figure 5.3 – System parameters graph. Circles and dotted circles represent learning and cryptographic parameters, respectively.

In our case, we decide to first choose N (ciphertext polynomial degree), such that at least c elements can be packed in one ciphertext. Q (ciphertext modulus) and η (fresh encryption noise) are then fixed to ensure a sufficient level of security (e.g., 128-bits) following the accepted parameterization from the homomorphic encryption standard whitepaper [5]. The scale Δ is configured to provide enough precision for the input data $\boldsymbol{X}$, and mc (moduli chain) and L (number of levels) are set accordingly. The intervals $[a_i, g_i]$ used for the approximations of the activation functions are defined according to $\boldsymbol{X}$. The approximation degrees $\boldsymbol{d}$ are then set depending on these intervals and the available number of levels L. The remaining learning parameters (α, ρ, b, g, m) are agreed upon by the data providers based on their observation of their part of the dataset. Note that the minimum values for the learning rate α and elastic rate ρ are limited by the scale Δ, and if they are too small the system might not have enough precision to handle their multiplication with the input data.

Data Outsourcing. SPINDLE's protocols (Section 5.3.2) seamlessly work with data providers (DPs) that either have their input data $\boldsymbol{X}$ in cleartext, or that obtain data $\langle \boldsymbol{X} \rangle_{pk}$ encrypted under the public collective key from their respective owners. In the latter case, SPINDLE enables both secure data storage and computation outsourcing to always-available untrusted cloud providers. It distributes the workload among multiple data providers and is still able to rely on efficient multiparty homomorphic-encryption operations, e.g., DBootstrap$(\cdot)$. We note that operating on encrypted input data affects the complexity of MAP, as all the multiplication operations (Protocol 5.2) would happen between ciphertexts, instead of between the cleartext inputs and ciphertexts.

Model Release. By default, the trained model in SPINDLE is kept secret from any entity, enabling privacy-preserving predictions on (private) evaluation-data input by the querier and offering end-to-end *model confidentiality*. If required by the application setting, SPINDLE can also reveal the trained model to the querier or to a third party. This is collectively enabled by the DPs, who perform a DKeySwitch$(\cdot)$.

5.6 Security Analysis

We demonstrate that SPINDLE achieves the data and model confidentiality requirements defined in Section 5.2.1 by relying on the real/ideal simulation paradigm [139] and showing that a computationally-bounded adversary that controls up to ($|S| - 1$)-out-of-$|S|$ DPs cannot distinguish a *real* world experiment, in which the adversary is given actual data (sent by honest DP(s)), and an *ideal* world experiment, in which the adversary is given random data generated by a simulator.

The semantic security of the CKKS scheme is based on the hardness of the decisional RLWE problem [47; 140; 150]. The achieved practical bit-security against state-of-the-art attacks can be computed using Albrecht's LWE-Estimator [5; 6]. Mouchet et al. [165] prove that their distributed protocols, i.e., Collective Encryption-Key Generation, Collective Relinearization-Key Generation (DKeyGen$(\cdot)$) and Collective Key Switching (DKeySwitch$(\cdot)$ and DDec$(\cdot)$) are secure under the simulator paradigm. They show that the distribution of the cryptoscheme preserves its security in the passive adversary model with all-but-one dishonest DPs, as long as the decisional-RLWE problem is hard. Their proofs, which are constructed using the BFV scheme, generalize to our adaptation of their protocols to CKKS, as they preserve the same computational assumptions, and the security of CKKS is based on the same hard problem as BFV. To avoid that information can be inferred on the encrypted values or on the encryption keys from the processed ciphertexts [136], we rely on existing countermeasures [46; 58; 165] and add fresh noise sampled from a distribution that has a variance significantly larger than that of the input ciphertext noise distribution to the processed ciphertext [136]. The security of DBootstrap(·) is based on Lemma C.2.1 which we state and prove in Appendix C.2.

Proposition 1. *Assume that* SPINDLE *uses CKKS encryptions with parameters* (N, Δ, η, mc) *ensuring a post-quantum security level* λ*. Given a passive adversary corrupting at most* $|S|-1$ *parties,* SPINDLE *achieves* data and model confidentiality *for training and prediction.*

Sketch of the Proof. We consider a real-world simulator $\mathscr{S}$ that simulates the view of a computationally-bounded adversary corrupting $|S|-1$ parties, i.e., it has access to the inputs and outputs of $|S|-1$ parties. In PREPARE and MAP, the data providers (DPs) locally compute on their data and only exchange encrypted information with each other to perform DKeyGen(·) and DBootstrap(·). In COMBINE, the DPs' MAP outputs, encrypted under the public collective key, are aggregated. These outputs are the encrypted results of multiple local iterations in which elements derived from each DP_i's local private data $(\boldsymbol{X}^{(i)}, \boldsymbol{y}^{(i)})$ are combined with its encrypted local model $\langle \boldsymbol{w}^{(i,\cdot)} \rangle$ and the current encrypted global model $\langle \boldsymbol{w}_G \rangle$. This result is re-randomized (i.e., added to a fresh encryption of **0**) to ensure that the outputs of successive MAP (i.e., the inputs to COMBINE) do not leak any information about the DPs' data. In REDUCE, the global model is updated by combining encrypted data, and a DBootstrap(·) is executed. All the information exchanged by the DPs is encrypted and the DPs rely on the aforementioned CPA-secure-proven protocols. We show in Appendix C.2 that DBootstrap(·) is also simulatable. In PREDICTION, only encrypted information is exchanged and the security-proven DKeyswitch(·) protocol is used. We consider two cases: (a) the adversary controls $|S|-1$ DPs and (b) it controls the querier and $|S|-2$ DPs. In (a), the encryption of the querier's input data (with the DPs common public key pk) can be simulated by $\mathscr{S}$ and SPINDLE ensures *Data Confidentiality* of the querier. In (b) the confidentiality of the adversary-controlled-querier's data is trivial as there is no information to hide form the adversary. The simulator has access to the prediction result and can produce all the intermediate (indistinguishable) encryptions that the adversary sees. Hence, $\mathscr{S}$ can simulate all the values communicated during the TRAINING and PREDICTION by generating random ciphertexts using the parameters (N, Δ, η, mc), such that the real outputs cannot be distinguished from the ideal ones. The sequential composition of all cryptographic functions remains simulatable by $\mathscr{S}$, as different random values are used in each step, and the exchanged ciphertexts are re-randomized, i.e., there is no dependency between the random values that an adversary can leverage on. Also, the adversary cannot decrypt collectively encrypted data unless all DPs collude, which would contradict the considered threat model. Following this, SPINDLE ensures the data and model confidentiality of the honest party/ies.

Finally, we note that by design, SPINDLE thwarts active attacks on federated learning [112; 157; 169; 265] and model inversion attacks [83], as intermediate and final model weights are never revealed during TRAINING.

5.7 System Evaluation

We first analyze the theoretical complexity of SPINDLE before moving to the empirical evaluation of its prototype and its comparison with existing solutions.

5.7.1 Theoretical Analysis

We refer to Table C.1a (Appendix C.3) for the full complexity analysis of SPINDLE's protocols. We discuss here its main outcomes.

Communication Complexity. SPINDLE's communication complexity depends linearly on the number of data providers $|S|$, iterations (global g and local z) and the ciphertext size $|ct|$. In MAP, the only communication between the DPs is due to the DBootstrap($\cdot$), which requires two rounds of communication of one ciphertext (ct) between the $|S|$ DPs (i.e., $2 \cdot (|S|-1) \cdot |ct|$). In COMBINE and REDUCE, the DPs exchange one ciphertext in respectively one and two rounds. Finally, the PREDICTION requires the exchange of one ciphertext between a DP and the querier and one DKeySwitch($\cdot$) operation, i.e., 2 ciphertexts are sent per DP.

Computation Complexity. SPINDLE's most intensive computational part is MAP; its complexity depends linearly on the number of DPs $|S|$ and the number of iterations, and logarithmically on the number of features c and batch size b; all these parameters depend also on the dataset size. As shown in Section 5.4.2, the DA packing approach incurs a higher computation complexity but is embarrassingly parallel and can be more time-efficient than RBA depending on the available threads. The activation function is the only operation that requires ciphertext-ciphertext multiplications; its complexity depends logarithmically on the approximation degree. We empirically study the link between the approximation degree and the training accuracy in Section 5.7.2. SPINDLE's other steps and protocols only involve lightweight operations, i.e., ciphertexts additions and multiplications with plaintext values.

5.7.2 Empirical Evaluation

We implemented SPINDLE in Go [98]. Our implementation builds on top of Lattigo [159], an open-source Go library for lattice-based cryptography, and Onet [182], an open-source Go library for building decentralized systems. The communication between data providers (DPs) is done through TCP with secure channels (using TLS). We evaluate our prototype on an emulated realistic network, with a bandwidth of 1 Gbps between every two nodes, using Mininet [160]. We deploy SPINDLE on 5 Linux machines with Intel Xeon E5-2680 v3 CPUs running at 2.5GHz with 24 threads on 12 cores and 256 Giga Bytes RAM, on which we evenly distribute the DPs. We first provide SPINDLE's cryptographic operations micro-benchmarks before assessing SPINDLE accuracy and performance by testing it on multiple publicly-available datasets: CalCOFI [36] for linear regression, BCW [19], PIMA [192] and ESR [77] for logistic regression, and MNIST [134] for multinomial regression (see Appendix C.3 for details on the datasets). We then show SPINDLE's scalability by using randomly generated (larger) datasets with up to 8,192 features and 4 million data samples. Our evaluation shows SPINDLE practicality for large-dimensional datasets, making it suitable for demanding learning tasks such as the training on imaging or genomic datasets [20; 76; 135].

We employ two sets of security parameters (SP), both ensuring 128-bit security: SP1: (ring dimension $N = 2^{14}$, fresh ciphertext modulus $Q = 2^{438}$, $\eta = 3.2$, number of levels $L = 9$, scale $\Delta = 2^{34}$, degree of the approximated activation function $d = 5$) and SP2: ($N = 2^{13}$, $Q = 2^{218}$, $\eta = 3.2$, $L = 6$, $\Delta = 2^{30}$, $d = 3$). SP2 is sufficient for linear regression and for specific logistic regression models that accept a low-degree d approximation. All recurrent symbols are summarized in Table 1.3. To account for a wider-range of solutions, we rely on SP1, unless otherwise stated. We employ the *local bootstrap* approach for all our experiments and for all datasets in *baseline comparison* except for ESR and CalCOFI, for which we use *global bootstrap*, as in most cases, doing multiple local iterations (z) between two global iterations yields a better accuracy. We further study the choice of bootstrapping strategy later in this section. Unless otherwise stated, we employ the *diagonal approach* (DA) for packing; we compare it with the *row-based approach* (RBA) in Figure 5.4a. In all our experiments, we consider SPINDLE's total time to train a regression model (including communication) without PREPARE, which is executed once and mostly involves light plaintext operations. E.g., the complete PREPARE takes 16.5s for a dataset of 40,000 samples distributed among 4 DPs. MAP accounts for up to 99.5% of SPINDLE's execution time. As shown in Section 5.4, the DPs perform most of the computations in MAP, which is the only step with multiple local iterations, involving two matrix-vector multiplications, which span most (up to 97%) of its execution time. The remaining time corresponds to the computation of activation function and collective bootstrapping.

Micro-benchmarks. Tables 5.1 show the execution time of each cryptographic operation. We observe that, as mentioned before, SPINDLE replaces the usually costly bootstrapping operation by an efficient interactive protocol DBootstrap(·). One of the most recent works on bootstrapping by Han and Ki [106] introduces a solution that only achieves around 108 bits of security (lower than the recommended 128 bits, due to recent attacks [45; 224]) and executes a CKKS bootstrapping in 26 seconds with ciphertexts that can encrypt 2^{13} values, corresponding to SP1, and about 20 seconds for SP2. This is two orders of magnitude slower than our DBootstrap(·), that achieves 128-bit security with execution times of 0.6 and 0.25 seconds for SP1 and SP2 (with 40 DPs).

Baseline Comparison. To evaluate SPINDLE, we compare its performance (execution time and accuracy) against an *ideal* baseline, i.e., a non-privacy-preserving centralized cleartext solution (CCS) where a DP obtains the full dataset and trains the model on it. We consider the training time on the complete dataset and use the training batch size b as the number

Encrypt	0.02
Add	$7 \cdot 10^{-4}$
Mult	$3 \cdot 10^{-3}$
Rot	0.08
Relin	0.07
Rescale	0.01

(a) Local Crypto. Ops.

D. Op.	**5 DPs**	**10 DPs**	**20 DPs**	**40 DPs**
DKeyGen	2.14	3.13	4.20	5.65
DBootstrap	0.26	0.37	0.47	0.61
DKeySwitch	0.26	0.36	0.45	0.57

(b) Distributed Crypto Ops.

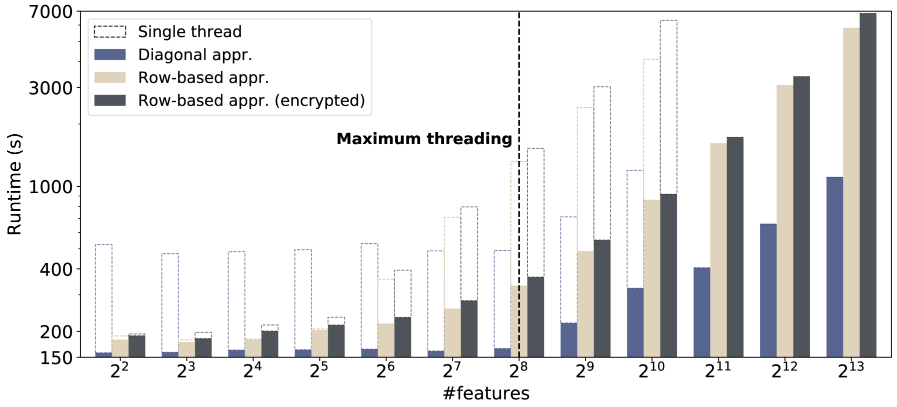
Table 5.1 – Micro-benchmarks of the cryptographic operations in seconds with SP1.

of data samples input for the prediction. Moreover, we compare SPINDLE with a distributed non-privacy-preserving (DNP) solution (cleartext values and exact activation functions), to show that our cryptographic approach and activation approximations introduce minimal accuracy degradation. Finally, to demonstrate the benefit of distributed learning approaches, we compare SPINDLE's accuracy with a case where one DP independently trains a model only on its local part of the distributed dataset (Independent Training, IT). In Table 5.2, we show SPINDLE's accuracy (Acc.) (resp., Mean Squared Error, MSE) and F1-score (F1) (resp., Mean Average Error, MAE) for logistic and multinomial (resp., linear) regressions, achieved on the above datasets when they are split among 10 DPs. Note that in Table 5.2, a linear regression is used on the CalCOFI dataset and the accuracy metrics are MSE and MAE. In that case, a DP alone obtains completely inaccurate predictions, i.e., large error values. We refer to Table C.1b in Appendix C.3 for the learning parameters description. We observe that SPINDLE's accuracy loss is very low, up to 0.8%, with respect to a non-private centralized (CCS) solution where the model is trained on the full dataset (using standard SGD) with a standard Python library [214]. For instance, on ESR, CCS yields 84.2%, and 83.9% with SPINDLE. Moreover, this loss is mainly due to the data not being centralized, as SPINDLE consistently achieves almost the same accuracy as the decentralized non-private (DNP) equivalent. SPINDLE's total training time (column T. in Table 5.2) is kept between 1 and 2 orders of magnitude higher than DNP, as the costly operations on encrypted data are partially amortized by SIMD operations enabled by the used packing. For instance, the training on the ESR dataset takes almost 3 seconds in DNP and 53.27 seconds in SPINDLE. We do not report the time for the centralized training (CCS and IT), as the settings are too different to be fairly comparable. Multinomial regression requires polynomial approximations of higher degree, i.e., between 15 and 19 (see Table C.1b); its training on 70,000 records of 784 features (MNIST) is executed in 558 seconds (column T. in Table 5.2). This time can be reduced to 187.8 seconds by performing 10 logistic regressions in parallel, one per label class (one-vs-all), at the cost of a 1% loss in accuracy. In all cases, when a DP independently trains on its part of the dataset (IT), i.e., with 1/10-th of the data, the achieved accuracy is worse than the one achieved on the entire distributed dataset. As for prediction (P.), SPINDLE's prediction on 10 data samples of 90 features (ESR dataset) requires only 0.35 seconds by packing the input data and executing parallel computations. This time can be further amortized if more predictions are run in parallel.

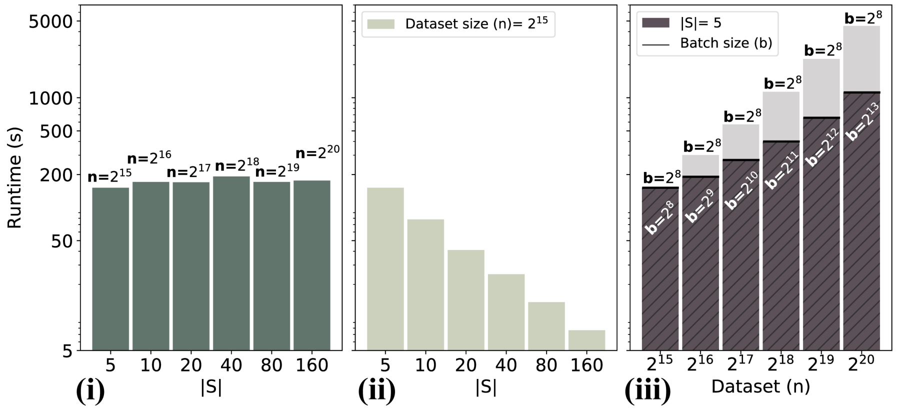
Dataset	Vers.	Acc./MSE	F1/MAE	T.	P.
CalCOFI [812,174x2]	CCS, [IT]	15.157, [408]	3.1, [19.67]	–	–
	DNP	17.679	3.45	6.71	$2 \cdot 10^{-4}$
	SPINDLE	17.938	3.62	65.31	0.23
PIMA [768x8]	CCS, [IT]	0.784, [0.720]	0.680, [0.604]	–	–
	DNP	0.781	0.679	0.038	$9 \cdot 10^{-5}$
	SPINDLE	0.780	0.677	11.28	0.18
BCW [699x9]	CCS, [IT]	0.962, [0.922]	0.947, [0.877]	–	–
	DNP	0.962	0.942	0.034	$5 \cdot 10^{-5}$
	SPINDLE	0.962	0.944	3.25	0.16
ESR [11,500x90]	CCS, [IT]	0.842, [0.838]	0.462, [0.396]	–	–
	DNP	0.840	0.460	2.89	$8 \cdot 10^{-5}$
	SPINDLE	0.839	0.456	53.27	0.35
MNIST [70,000 x 784] (multi.)	CCS, [IT]	0.873, [0.873]	0.871, [0.832]	–	–
	DNP	0.865	0.863	43.95	0.49
	SPINDLE	0.8617	0.86	558	4.33
MNIST [70,000 x 784] (1 vs. a)	CCS, [IT]	0.856, [0.827]	0.859, [0.822]	–	–
	DNP	0.853	0.858	43.98	0.49
	SPINDLE	0.852	0.850	187.8	4.33

Table 5.2 – Baseline Comparison with K-fold=5. CCS is a non-private centralized solution, DNP is a distributed non-privacy preserving approach and IT means independent training, i.e., a DP using only its local subset of the entire dataset. Time to train (T.) and to predict (P.) are in seconds. Mean-squared error (MSE) and mean averaged-error (MAE) are given for the linear regression on CalCOFI. Accuracy and F1-score are given for all the others.

Scalability. We study how Spindle's execution time evolves when increasing the number of: features (c), data providers ($|S|$), and dataset samples (n). By default, we set $|S| = 5$, each DP having 5,120 data records (synthetically generated) with $c = 32$ features; we use a batch size $b = 256$, with $g = 5$ global iterations, and $z = 20$ local iterations in MAP. When comparing different approaches, we ensure that the number of times that the dataset is fully processed is constant, and we set the learning parameters accordingly. Figure 5.4a displays Spindle's execution time with an increasing number of features c and shows that it scales logarithmically, in any of the used approaches. In this setting, we also study the influence of the multi-threading, the differences between the two packing approaches (Section 5.4.2) and the impact of having encrypted input data (Section 5.5). When the computations are single-threaded, the row-based approach (RBA) is more efficient than the diagonal approach (DA) up to $c = 128$ features, as RBA incurs fewer multiplications and rotations than DA. In contrast, the diagonal approach (DA) execution time in one or multiple threads is almost constant up to $c = 256$ features (with a batch size $b = 256$), as its complexity depends mainly on $\max(c, b)$ (Section 5.4.2). However, the DA is *embarrassingly parallelizable*, and it is always faster when the computations are executed on 24 threads. As an example, on multiple threads and for 256 features, DA yields an execution time of 165s against 330s for RBA, and 365s when the input data are encrypted and using RBA. For both approaches, the parallelization is efficient up to $c = 2^8$, where the maximum thread-utilisation is reached. Afterwards, both approaches scale linearly. When the data providers have encrypted input data (RBA-E), the execution time increases by 7% with



(a) Spindle's performance with the nbr. of features (c).



(b) Spindle's performance with the nbr. of DPs ($|S|$) & records (n).

Figure 5.4 – Spindle's scalability.

respect to RBA.

Figure 5.4b.i shows that when the number of DPs $|S|$ increases and each DP has a fixed amount of data, Spindle's execution time is constant. This means that Spindle scales independently of $|S|$. In Figure 5.4b.ii, where $|S|$ increases but the total amount of data remains constant, Spindle's execution time decreases linearly, as the workload is efficiently distributed among the DPs. In Figure 5.4b.iii, when $|S|$ is constant and the size of the DPs' datasets increases, Spindle's execution time increases linearly with the amount of data. If the batch size can be increased when the data providers have more records, then Spindle's execution time can be further reduced. In summary, Spindle scales independently of the number of data providers, and linearly with the DPs' dataset size. It is able to train models with a high number of features and thus remains practical for real-world sized datasets. We note that Spindle scales similarly to a DNP solution in the three cases. For example, in the case of Figure 5.4b.ii, DNP ranges from 0.69 seconds with 5 DPs to 0.42 seconds with 10 DPs, whereas Spindle's execution time decreases from 150 to 78 seconds.

Bootstrapping & Activation Function Approx. Degree. In Table 5.3, we observe that Spindle accuracy slightly improves with higher degree approximations of the sigmoid for the activation function. Relying on LB or HB, which require less global iterations and therefore less communication, and on low-degree approximations improves SPINDLE execution time but, it can lower the achieved accuracy. We remark that HB requires less bootstrapping operations, as the global weights are bootstrapped and assigned to the local weights. However, LB and HB execution times remain similar as in LB, the DP perform the bootstrappings in parallel.

$_{boot.} \backslash ^{activ.}$	**3**	**5**	**7**
LB	0.837 ; 120	0.839 ; 125	0.839 ; 128
GB	0.843 ; 140	0.843 ; 143	0.844 ; 146
HB	0.844 ; 120	0.846 ; 124	0.846 ; 128

Table 5.3 – SPINDLE accuracy and timing *(accuracy; execution time in sec)* to train on ESR with different bootstrapping strategies and degrees of the activation function. LB refers to the local bootstrap approach, GB the global approach and HB is the hybrid method, i.e., combining LB and GB.

Comparison with prior art. Here we briefly compare, both qualitatively and quantitatively (when applicable), SPINDLE against (a) centralized cryptographic approaches (e.g., [39; 124; 126]), (b) cryptographic distributed solutions (e.g., [52; 263]) and (c) federated learning solutions (e.g., [64; 114; 137; 220]). See Appendix C.3 for an extended analysis.

(a) SPINDLE consistently outperforms centralized HE-based solutions (CES) as SPINDLE distributes the workload among multiple DPs, achieves a high-level of security by using efficient cryptographic parameters, and replaces, as shown before, the costly centralized bootstrapping operation by a lightweight interactive protocol.

(b) None of the existing HE-SMC-based distributed solutions [52; 263] provides both data and model confidentiality, or covers the entire ML workflow (the trained model cannot be kept secret to perform oblivious predictions) or enables the distributed execution of the gradient descent. Moreover, existing solutions (e.g., [52]) and the system (DRYNX) presented in the previous chapter, leak more than only the trained model and rely on data encodings (or approximations) that lower the obtained accuracy, whereas in SPINDLE, we approximate only the activation functions. Finally, all previous solutions scale quadratically in at least one dimension, i.e., number of features c, samples n, or DPs $|S|$, whereas SPINDLE's execution time is almost independent of $|S|$, scales logarithmically with the number of features and linearly with the dataset size. Purely secret-sharing-based solutions [26; 49] consider substantially different settings as SPINDLE, as they require the DPs to communicate their data outside their premises and to a limited number of computing servers (typically, 2 to 4, depending on the setting). Whereas SPINDLE also works in this configuration, it is not specifically optimized for 2-4 parties and its execution time would be in the same order of magnitude but slower than secret-sharing-based solutions. This is due to the computation overhead introduced by operations on encrypted data. However, unlike secret-sharing-based solutions, SPINDLE efficiently scales to federated learning settings where many (hundreds of) DPs keep their data locally and can withstand up to N-1 out of N dishonest DPs.

(c) In basic federated learning solutions, data owners train and update the model on their local data and a server aggregates the model updates to obtain the global model [130; 153]. In this setting, the coordinating server has to be fully trusted, as some information can be inferred from the intermediate models, e.g., extracting participants' inputs [112; 243; 265] or membership inference [157; 169]. SPINDLE naturally thwarts federated-learning and model-inversion attacks, as the intermediate and final weights are never revealed. Federated learning approaches based on differential privacy (diffP), e.g., [137; 154; 220], train the model while introducing noise to the intermediate values to mitigate adversarial inferences. These approaches consider a different paradigm by introducing a tradeoff between privacy and accuracy, whereas in SPINDLE security is absolute, and the trade-off (accuracy vs execution time) is the same as for non-secure solutions, e.g., less training iterations can yield a less precise model. DiffP approaches can significantly degrade the data utility, and might require a high privacy budget for which it remains unclear what privacy protection is obtained in practice [119]. In Appendix C.4, we discuss how membership inference and reconstruction attacks from the prediction outputs can also be mitigated in SPINDLE by adding differentially-private noise during the DKeySwitch$(\cdot)$.

5.8 Conclusion

By extending the MapReduce abstraction, we have proposed a generic solution to the problem of privacy-preserving distributed ML model training and prediction. Our abstraction enables us to optimize the application of protection primitives from multiparty homomorphic encryption in a MapReduce workflow. We have proposed SPINDLE, a privacy-preserving system that enables the execution of a distributed stochastic gradient descent and we have instantiated our quantum-resistant solution for the training and oblivious prediction on generalized linear models. We have shown that SPINDLE achieves accuracy comparable to non-secure centralized solutions, and it scales independently of the number of DPs and linearly or better with the size of the DPs' local datasets and the number of features. This makes it particularly suitable for difficult and demanding learning tasks that have to be performed on sensitive data that cannot be shared. This is the case in many domains and particularly in medicine, where complex sensitive datasets partitioned across medical institutions need to be regularly analyzed, e.g., Genome Wide Association Studies. SPINDLE achieves better performance than existing centralized and distributed solutions by leveraging the data providers concurrent computation on their local data, and using a multiparty encryption scheme that replaces costly homomorphic operations (e.g., bootstrapping) by efficient collective protocols. To the best of our knowledge, SPINDLE is the first highly scalable system enabling the distributed execution of the gradient descent across hundreds of parties and large datasets in a privacy-preserving, post-quantum, and efficient way.

6 Federated Analytics for Precision Medicine

6.1 Introduction

In this chapter, we show that the solutions introduced in this thesis can be applied in the biomedical domain; this is a paradigmatic example where privacy is paramount and data sharing is needed. More precisely, we rely on the core method of the previous solutions and present a new approach to biomedical Federated Analytics (FA) based on Multiparty Homomorphic Encryption (MHE) (See Section 1.4) that effectively combines SMC and HE. We call our approach Federated Analytics with Multiparty Homomorphic Encryption (FAMHE). By combining the power of HE to perform computations on encrypted data without communication between the parties, with the benefits of interactive protocols, which can greatly simplify certain expensive HE operations, we build an efficient system for federated privacy-preserving FA. Building upon the MHE framework, we introduce a new approach to biomedical FA, where each participating institution performs local computation and encrypts the intermediate results by using MHE; the results are then combined (e.g., aggregated) and distributed back to each institution for further computations. This process is repeated until the desired analysis is completed. Contrary to diffP-based approaches that rely on obfuscation techniques to mitigate the leakage in intermediate results, by sharing only encrypted intermediate results, FAMHE provides full privacy protection, without sacrificing accuracy. By sharing only encrypted information, our approach guarantees that, whenever needed, a minimum level of obfuscation can be applied only to the final result in order to protect it from inference attacks, instead of being applied to all intermediate results. Furthermore, FAMHE improves over both SMC and HE approaches by minimizing communication, by scaling to large numbers of data providers, and by circumventing expensive non-interactive operations (e.g., bootstrapping in HE). Our work also introduces a range of optimization techniques for FAMHE, including optimization of the local vs. collective computation balance, ciphertext packing strategies, and polynomial approximation of complex operations; these techniques are instrumental in our efficient design of FAMHE solutions for survival analysis and GWAS.

We demonstrate the performance of FAMHE by replicating two published multi-centric studies that originally relied on data centralization. These include a study of metastatic cancer patients and their tumor mutational burden [207], and a host genetics study of HIV-1 infected patients [151]. By distributing each dataset across multiple data providers and by performing federated analyses using our approach, we successfully recapitulated the results of both original studies. Our solutions are efficient in terms of both execution time and communication, e.g., completing a GWAS over 20K patients and four millions variants in less than five hours. In contrast to most prior works on biomedical FA, which relied on artificial datasets [29; 115; 116], our results closely reflect the potential of our approach in real application settings. Furthermore, our approach has the potential to simplify the requirements for contractual agreements and the obligations of data controllers that often hinder multi-centric medical studies, because data processed by using MHE can be considered anonymous data under the General Data-Protection Regulation (GDPR) [210]. Our work shows that FAMHE is a practical framework for privacy-preserving FA for biomedical workflows and it has the power to enable a range of analyses beyond those demonstrated in this chapter.

Acknowledgements. This work is the result of a collaboration with Dr. Juan R. Troncoso-Pastoriza, Dr. Jean Louis Raisaro, Dr. Michel Cuendet, Joao Sa Sousa, Dr. Hyunghoon Cho, Prof. Bonnie Berger, Prof. Jacques Fellay, and Prof. Jean-Pierre Hubaux.

6.2 System Overview & Design

We assume similar system and threat models as in previous chapters. We introduce the two main workflows, survival curves and GWAS, in which we showcase the use of our proposed framework.

6.2.1 System Model & Proposed Workflow

In FAMHE, we rely on MHE to perform privacy-preserving FA by pooling the advantages of both interactive protocols and HE and by minimizing their disadvantages. In particular, by relying on MHE and on the distributed protocols for federated analytics proposed in Chapter 5, our approach enables several sites to compute on their local patient-level data and then encrypt (*Local Computation & Encryption* in Figure 6.1) and homomorphically combine their local results under MHE (*Collective Aggregation* in Figure 6.1). These local and global steps can be repeated (*Iterate* in Figure 6.1), depending on the analytic task. At each new iteration, participating sites use the encrypted combination of the results of the previous iteration to compute on their local data without the need of decryption, e.g., gradient descent steps in the training of a regression model. The collectively encrypted and aggregated final result is eventually switched (*Collective Key Switching* in Figure 6.1) from an encryption under the collective public key to an encryption under the querier's public key (the blue lock in Figure 6.1) such that only the querier can decrypt. The use of MHE ensures that the secret key of the underlying HE scheme never exists in full. Instead, the control over the decryption process is distributed across all participating sites, each one holding a fragment of the decryption key.

This means that all participating sites have to agree to enable the decryption of any piece of data, and that no single entity alone can decrypt the data. As described in Section 6.2.2, FAMHE is secure in a passive adversarial model in which all-but-one data providers can be dishonest and collude among themselves.

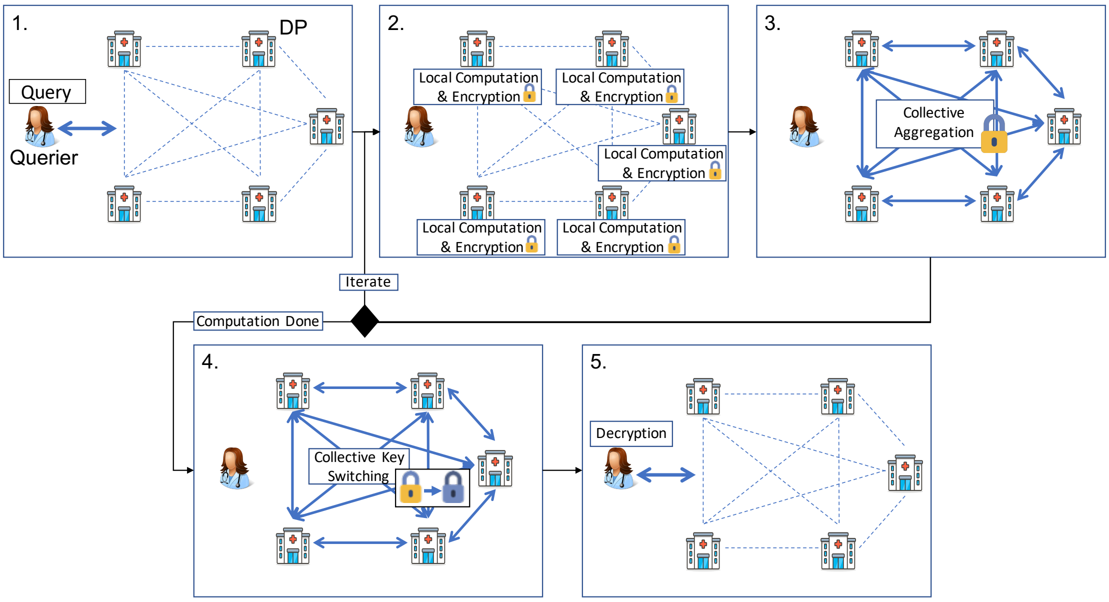


Figure 6.1 – System Model and FAMHE Workflow. All entities are interconnected (dashed lines), and the communication links at each step are shown by thick arrows. All entities (data providers (DPs) and querier) are honest-but-curious and do not trust each other. In **1.**, the querier sends the query (in clear) to all the DPs who (**2.**) locally compute on their cleartext data and encrypt their results with the collective public key. In **3.**, the DPs' encrypted local results are aggregated. For iterative tasks, this process is repeated (**Iterate**). In **4.**, the final result is then collectively switched by the DPs from the collective public key to the public key of the querier. In **5.**, the querier decrypts the final result.

6.2.2 Threat Model

FAMHE supports a network of mutually distrustful medical institutions that act as data providers (DPs) and hold subjects' records. An authorized querier (see Figure 6.1) can run queries without threatening the data confidentiality and the subjects' privacy. The DPs and the querier are assumed to follow the protocol and to provide correct inputs. All-but-one data providers can be dishonest, i.e., they can try to infer information about other data providers by using the protocol's outputs. We assume that the DPs are available during the complete execution of a computation. However, to account for unresponsive DPs, FAMHE can use a threshold-encryption scheme, where the DPs secret-share [217] their secret keys, thus enabling a subset of the DPs to perform the cryptographic interactive protocols.

6.2.3 FAMHE Design

In this section, we first discuss the main principles of FAMHE's design before describing how to instantiate the previously introduced generic workflow (Figure 6.1) in order to compute, in a federated and privacy-preserving manner, a Kaplan-Meier survival curve and GWAS.

6.2.3.1 FAMHE's Design Principles

FAMHE builds upon novel optimization techniques for enabling the efficient execution of complex iterative workflows (1) by relying on edge-computing and optimizing the use of computations on the data providers' cleartext data, (2) by relying on the packing ability of the MHE scheme to encrypt vector of values in a single ciphertext such that any computation on a ciphertext is performed simultaneously on all the vector values, i.e., Single Instruction, Multiple Data (SIMD), (3) by further building on this packing property to optimize the sequence of operations by formatting a computation output correctly for the next operation, (4) by approximating complex computations such as matrix inversion (i.e., division) by polynomial functions (additions and multiplications) to efficiently compute them under HE, and (5) by replacing expensive cryptographic operations by lightweight interactive protocols. Note that FAMHE avoids the use of centralized complex cryptographic operations that would require a much more conservative parameterization and would result in higher computational and communication overheads (e.g., due to the use of larger ciphertexts). Therefore, FAMHE efficiently minimizes the computation and communication costs for a high security level.

As discussed in Section 6.2.3.3, in the case of GWAS, FAMHE efficiently performs multiple subsequent large-dimensions matrix operations (see Protocols 6.1 and 6.2) by optimizing the data packing (see Figure D.1 in Appendix D.1) to perform several multiplications in parallel and minimize the amount of transformations required on the ciphertexts. FAMHE builds on the data providers' ability to compute on their cleartext local data and combine them with encrypted data, thus reducing the overall computation complexity. GWAS also requires non-polynomial functions, e.g., the inverse of a matrix, to be evaluated on ciphertexts, which is not directly applicable in HE. In FAMHE, these non-polynomial functions are efficiently approximated by relying on Chebyshev polynomials. We chose to rely on Chebyshev polynomials instead of least-square polynomial approximations in order to minimize the maximum approximation error and thus avoid that the function diverges on specific inputs. We have shown that this technique accurately approximates non-polynomial functions in the training of generalized models in Chapter 5, which further shows the generality and applicability of our proposed framework.

FAMHE combines the aforementioned features to efficiently perform FA with encrypted data. In GWAS, for example, we rely on the Gauss-Jordan (GJ) method [12] to compute the inverse of the covariance matrix. We chose this algorithm as it can be efficiently executed by relying on the aforementioned features: row operations can be efficiently parallelized with SIMD and divisions are replaced by polynomial approximations.

6.2.3.2 Multi-centric Kaplan-Meier Survival Analysis Using FAMHE

Kaplan-Meier survival analysis is a widely used method to assess patients' response (i.e., survival) over time to a specific treatment. Survival curves are generally estimated with the Kaplan-Meier estimator [99]:

$$\hat{S}(t) = \prod_{j,\ t_j \le T} \left(1 - \frac{d_j}{n_j}\right), \tag{6.1}$$

where t_j is a time when at least one event has occured, d_j is the number of events at time t_j, and n_j is the number of individuals known to have survived (or at risk) just before the time point t_j. A survival curve (see an example in Figure 6.3a) starts at 1 (or 100% at time 0); each step down is the occurrence of an event (e.g., death of a patient); and ticks indicate the presence of censored patients, i.e., patients who withdrew from the study. The number of censored patients at time t_j is indicated by c_j. As shown in Figure 6.2, to compute this curve in a federated manner, each data provider i locally computes, encodes and encrypts a vector of the form $n_0^{(i)}, c_0^{(i)}, d_0^{(i)}, ..., n_T^{(i)}, c_T^{(i)}, d_T^{(i)}$ containing the values $n_j^{(i)}, c_j^{(i)}, d_j^{(i)}$ corresponding to each time point t_j for $t_j = 0, ..., T$. All the DPs' vectors are then collectively aggregated. The encryption of the final result is then collectively switched from the collective public key pk to the querier's public key that can decrypt the result with its secret key and generate the curve following Eq. (6.1).

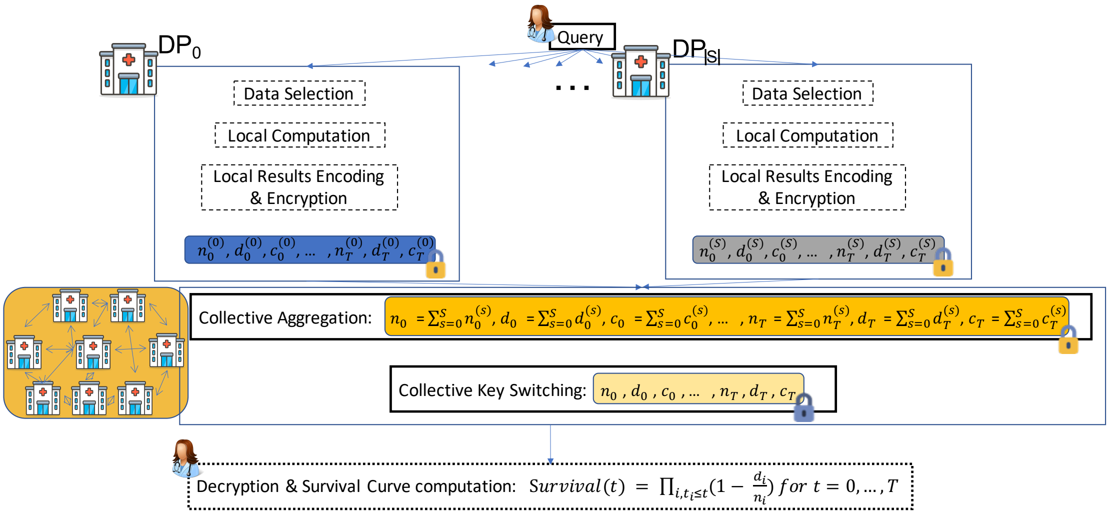

Query
DP0
DP|S|
Data Selection
Local Computation
Local Results Encoding & Encryption
Data Selection
Local Computation
Local Results Encoding & Encryption
$n_0^{(0)}, d_0^{(0)}, c_0^{(0)}, \dots, n_T^{(0)}, d_T^{(0)}, c_T^{(0)}$
$n_0^{(S)}, d_0^{(S)}, c_0^{(S)}, \dots, n_T^{(S)}, d_T^{(S)}, c_T^{(S)}$
Collective Aggregation: $n_0 = \sum_{s=0}^{S} n_0^{(s)}, d_0 = \sum_{s=0}^{S} d_0^{(s)}, c_0 = \sum_{s=0}^{S} c_0^{(s)}, \dots, n_T = \sum_{s=0}^{S} n_T^{(s)}, d_T = \sum_{s=0}^{S} d_T^{(s)}, c_T = \sum_{s=0}^{S} c_T^{(s)}$
Collective Key Switching: $n_0, d_0, c_0, \dots, n_T, d_T, c_T$
Decryption & Survival Curve computation: $Survival(t) = \prod_{i,t_i \le t}(1 - \frac{d_i}{n_i})$ $for\ t = 0, \dots, T$

Figure 6.2 – Secure & Distributed Computation of a Kaplan-Meier Survival Curve. As before, a yellow lock indicates a collective encryption, and the blue lock indicates an encryption under the querier's public key.

6.2.3.3 Multi-centric Genome-Wide Association Studies Using FAMHE

Genome-wide association studies (GWAS) are a fundamental analysis tool in medical genetics that identifies genetic variants that are statistically associated with given traits, such as disease status. GWAS have led to numerous discoveries about human health and biology, and efforts to collect larger and more diverse cohorts to improve the power of GWAS. Their relevance to diverse human populations continue to grow. As we progress toward precision medicine and genetic sequencing becomes more broadly incorporated into routine patient care, large-scale GWAS that span multiple medical institutions will become increasingly more valuable.

In GWAS, the p-value corresponding to each variant is computed to check for any association between the variant and the phenotype. This computation is usually performed on a large number of variants (e.g., four million in Section 6.3) hence requires the parallel training of many regression models.

We consider a data set of p samples, i.e., patients. We list the main recurrent symbols and acronyms in Table 1.1. Each patient is described by f features or covariates (with indexes 1 to f). Hence, we have a covariate matrix $X \in \mathbb{R}^{(p\times f)}$. Each patient also has a phenotype or label, i.e., $y \in \mathbb{R}^{(p\times 1)}$ and v variant values, i.e., one for each variant considered in the association test. The v variant values for all p patients form another matrix $V \in \mathbb{R}^{(p\times v)}$. To perform the GWAS, for each variant i, the matrix $X' = [\mathbf{1}, X, V[:,i]] \in \mathbb{R}^{(p\times(f+2))}$, i.e., the matrix X is augmented by a column of 1s (intercept) and the column of one variant i, is constructed. The vector $w \in \mathbb{R}^{(f+2)}$ is then obtained by $w = (X'^{T} X')^{(-1)} X'^{T} y$. The p-value for variant i is then obtained with $p_{val} = 2 \cdot pnorm(-|\frac{w[f+2]}{\sqrt{(MSE(y,y')\cdot(X'^{T} X')^{(-1)}[f+2;f+2]}}|)$ where $pnorm$ is the cumulative distribution function (CDF) of the standard normal distribution, $w[f+2]$ is the weight corresponding to the variant, $MSE(y,y')$ is the mean squared error obtained from the prediction y' computed with w, and $(X'^{T} X')^{(-1)}[f+2;f+2]$ corresponds to the standard error of the variant weight.

Although this computation has to be performed for each variant i, we remark that X is common to all variants. In order to compute $(X^T X)^{(-1)}$ only once before adjusting it for each variant thus obtaining $(X'^{T} X')^{(-1)}$. We rely on the Shermann-Morrison formula [219] and the method presented in the report on cryptographic and privacy-preserving primitives (page 52) of the WITDOM European project [245].

We describe two approaches to perform this workflow in a federated and privacy-preserving, i.e., under MHE, manner: (i) FAMHE-GWAS, in Protocol 6.1, an approach to perform an exact GWAS, and (ii) FAMHE-FastGWAS, in Protocol 6.2, a computationally-optimized approximated approach. In both protocols, each data provider DP_i has a subset of p_i patients. For efficiency, the DPs are organized in a tree structure and one DP is chosen as the root of the tree DP_R. We remark that, as any exchanged information is collectively encrypted, this does not have any security implications. In a *Collective Aggregation (CA)*, each DP encrypts ($E()$) its local results with the collective key, aggregates its children DPs' encrypted results with its encrypted local results, and sends the sum to its parent DP such that DP_R obtains the encrypted result aggregated among all DPs. We rely on SIMD to parallelize the operations at multiple levels: among the DPs, among the threads in each DP, and among the values in the ciphertexts. For a 128-bit security level, the computations are performed simultaneously for 512 variants with

FAMHE-GWAS (Protocol 6.1) and for 8192 with FAMHE-FastGWAS (Protocol 6.2). We describe, in Figure D.1 (in Appendix), how the (main) values used in both protocols are packed to optimize the communication and the amount of required operations (multiplications, rotations). As shown below, we perform permutations, duplications, and rotations on cleartext data that are held by the DPs (indicated in orange in Figure D.1 in Appendix); we optimize the balance between cleartext operations and operations on encrypted vectors. Note that rotations on ciphertexts are almost two orders of magnitude slower than multiplications and additions, and they should be avoided when possible. As ciphertexts have to be aggregated among DPs, a tradeoff has to be found between computation cost (e.g., rotations) and data packing, as a smaller packing density would require the exchange of more ciphertexts.

In both protocols, all operations for v variants are executed in parallel. We highlight (in bold) the main steps and the aggregated values in the protocol, and we note that DPs' local data are in cleartext, whereas all exchanged data are collectively encrypted ($E()$).

FAMHE-GWAS (Protocol 6.1). Protocol 6.1 depicts FAMHE secure and federated workflow for the computation of an exact Genome-Wide Association Study. First the covariance matrix ($X^T X$) is collectively computed (collective aggregation, CA) by the DPs, before being inversed in the encrypted domain by one data provider (DP_R) in **Step 1**. As mentioned before, we rely on the Gauss-Jordan (GJ) method [12] to compute the inverse of the encrypted covariance matrix. We chose this algorithm as it requires only row operations that can be efficiently performed with SIMD. The only operation that is not directly applicable in HE is the division that we approximated with a Chebyshev polynomial. Note that we avoid any other division in the protocol by pushing them to the last step that is executed, after decryption, by the querier Q. For example, in Protocol 6.1, we keep $1/c$ until decryption. In **Step 2**, the inversed matrix is augmented with the variant's contribution (vector u that contains the variant's values for each patient) by following the Shermann-Morrison formula [219] and the method presented in the report on Cryptographic and Privacy-Preserving Primitives (page 52) of the WITDOM European project [245]. We remark that, whenever possible, the DPs locally compute on their cleartext data before securely aggregating their partial results. The matrix $\boldsymbol{W}$ is made of 3 sub-matrices ($\boldsymbol{W_{11}}$, $\boldsymbol{W_{12}}$,$\boldsymbol{W_{21}}$) that are computed in Step 2. In **Step 3**, the model weights (or coefficients, w) are computed such that all elements required to obtain the p-value corresponding to the variant's coefficient can be computed in **Step 4**. In **Step 5**, the results are switched (Key Switching, KS) to the querier's public key such that they can compute the final result in **Step 6**.

Protocol 6.1 FAMHE-GWAS.

```
Step 1: Inverse of covariance matrix of covariates
Each $DP_i$: $X_i = [\mathbf{1}, X_i]$ with $X_i \in \mathbb{R}^{(p_i \times (f+1))}$
Each $DP_i$ computes $X_i^T X_i$ with $X_i^T X_i \in \mathbb{R}^{((f+1)\times(f+1))}$
CA → $\boldsymbol{E(X^T X)}$
$DP_R$: $\boldsymbol{E((X^T X)^{(-1)}) = GJ(E((X^T X)))}$ and broadcasts
Step 2: Variants contribution
For each column $u$ in $V$ do:
    Each $DP_i$ computes $u_i^T u_i$, $u_i^T X_i \times E((X^T X)^{(-1)})$
    CA → $\boldsymbol{E(u^T u)}$, $\boldsymbol{E(W_{21}) = E(u^T X (X^T X)^{(-1)})}$
    $DP_R$ broadcasts $E(u^T X (X^T X)^{(-1)})$
    Each $DP_i$ computes $E(u^T X (X^T X)^{(-1)}) \times X_i^T u_i$
    CA → $\boldsymbol{E(u^T X (X^T X)^{(-1)} X^T u)}$
    $DP_R$ computes $\boldsymbol{E(\frac{1}{c}) = E(u^T u - u^T X (X^T X)^{(-1)} X^T u)}$
    Each $DP_i$ computes $E((X^T X)^{(-1)}) \times X_i^T u_i$
    CA → $\boldsymbol{E(W_{12}) = E((X^T X)^{(-1)} X^T u)}$
    $DP_R$ computes $\boldsymbol{E(W_{11}) = E(\frac{1}{c}(X^T X)^{(-1)} + W_{12} W_{21})}$
    $\boldsymbol{E(W) = E\begin{pmatrix} W_{11} & W_{12} \\ W_{21} & 1 \end{pmatrix}}$
Step 3: All coefficients
    Each $DP_i$ computes $[X_i, u_i]^T y_i$
    CA → $\boldsymbol{E(w) = E(W \times [X, u]^T y)}$
Step 4: P-value elements
    Each $DP_i$: $E(y_i') = X_i \times E(w)$ and $E(mse_i) = \frac{1}{p}\sum_{j=0}^{p_i}(E(y_i'[j]) - y_i[j])^2$
    CA → $\boldsymbol{E(mse)}$
Step 5: Key Switching
    $\mathrm{KS}_Q(E(mse), E(\frac{1}{c}, E(w[f+2]))$ and send to $Q$
Step 6: Querier final result
    Querier decrypts and $p_{val} = 2 \cdot pnorm(-|\frac{w[f+2]}{\sqrt{(mse \cdot c)}}|)$
```

FAMHE-FastGWAS (Protocol 6.2). In this protocol, we show how the GWAS can be estimated hence computed with a lower complexity; this computation is done by avoiding the computation of the complete inverse matrix of the covariance matrix. We reduce the computation overhead by obtaining the covariates' weights w' with a lightweight federated gradient-descent (FGD), by reporting the obtained covariates' contributions in the phenotype y, which becomes y''. To compute the p-value, we then compute only one element of the covariance inverse matrix $(X'^T X')^{(-1)}[f+2; f+2]$, instead of the entire inverse. In **Step 2**, to perform the federated gradient descent on an encrypted model, we follow the method described in Chapter 5, without disclosing any intermediate values. The following steps are similar to Protocol 6.1. **Step 3** corresponds to a partial execution of steps 2 and 3 from Protocol 6.1. In **Step 4**, we compute the coefficient that corresponds to the variant and the other elements required to obtain the p-value.

Protocol 6.2 FAMHE-FastGWAS.

```
Step 1: Inverse of covariance matrix of covariates
  Each $DP_i$: $X_i = [\mathbf{1}, X_i]$ with $X_i \in \mathbb{R}^{(p_i \times (f+1))}$
  Each $DP_i$ computes $X_i^T X_i$ with $X_i^T X_i \in \mathbb{R}^{((f+1)\times(f+1))}$
  CA → $\boldsymbol{E(X^T X)}$
  $DP_R$: $\boldsymbol{E((X^T X)^{(-1)}) = GJ(E((X^T X)))}$ and broadcasts
Step 2: Covariates coefficients & error
  $\boldsymbol{E(w[0:f+1])} = SGD(X, y)$ and broadcasts
  Each $DP_i$: $E(y_i'') = y_i - E(X_i w[0:f+1])$
  Each $DP_i$: $E(\bar{y_i''}) = \frac{1}{p}\sum_{j=0}^{p_i} E(y_i''[j])$
Step 3: Variants contribution
  For each: column $u$ in $V$ do:
    Each $DP_i$ computes $u_i^T u_i$, $u_i^T X_i \times E((X^T X)^{(-1)})$
    CA → $\boldsymbol{E(u^T u)}$, $\boldsymbol{E(W_{21}) = E(u^T X(X^T X)^{(-1)})}$
    $DP_R$ broadcasts $E(u^T X(X^T X)^{(-1)})$
    Each $DP_i$ computes $E(u^T X(X^T X)^{(-1)}) \times X_i^T u_i$
    CA → $\boldsymbol{E(u^T X(X^T X)^{(-1)} X^T u)}$
Step 4: P-value elements
    $DP_R$ compute $\boldsymbol{E(\frac{1}{c}) = E(u^T u - u^T X(X^T X)^{(-1)} X^T u)}$
    Each $DP_i$: $E(\bar{u}_i) = \frac{1}{p}\sum_{j=0}^{p_i} E(u_i[j])$
    CA → $\boldsymbol{E(\bar{y''})}$, $\boldsymbol{E(\bar{u})}$ and broadcasts
    Each $DP_i$: $E(d_i) = \sum_{j=0}^{p_i}(u_i[j] - E(\bar{u}))^2$ and $E(t_i) = \sum_{j=0}^{p_i}(u_i[j] - E(\bar{u}))(y_i''[j] - E(\bar{y''}))$
    CA → $\boldsymbol{E(d)}$, $\boldsymbol{E(t)}$ and broadcasts
    Each $DP_i$: $E(mse_i') = \sum_{j=0}^{p_i}(E(d) y_i''[j] - E(t) u[j])^2$
    CA → $\boldsymbol{E(mse')}$
Step 5: Key Switching
    $\mathrm{KS}_Q(E(mse'), E(\frac{1}{c}), E(t), E(d)$ and send to $Q$
Step 6: Querier final result
    Querier decrypts and $p_{val} = 2 \cdot pnorm(-|\frac{t/d}{\sqrt{\frac{1}{d^2}(mse' \cdot c)}}|)$
```

6.3 Evaluation

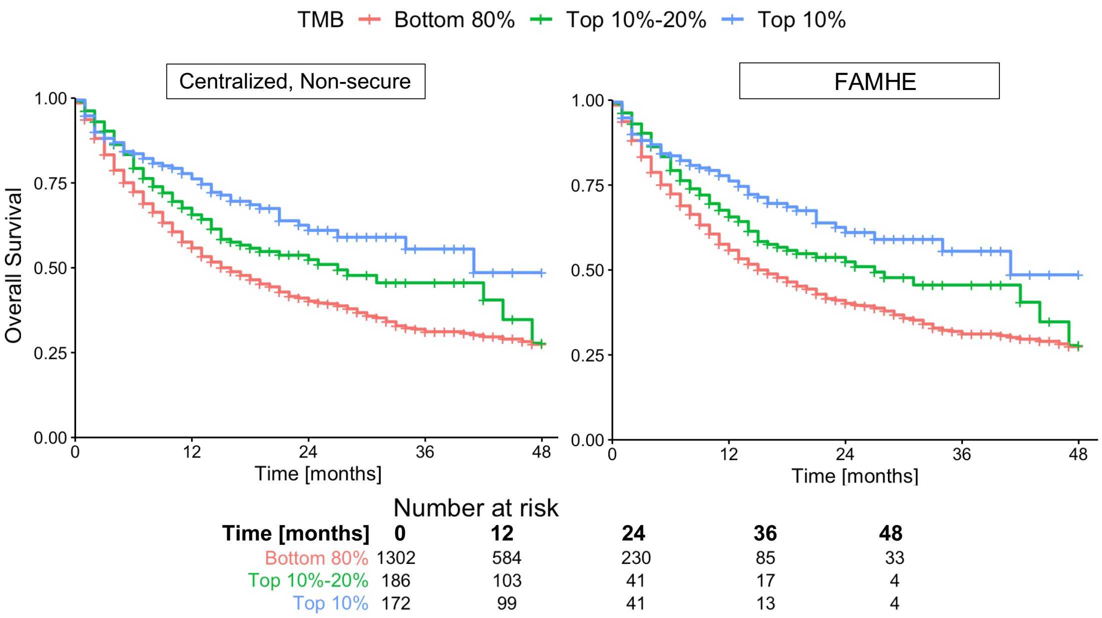
To demonstrate the performance of FAMHE, we test it on the two essential biomedical tasks introduced in Sections 6.2.3, i.e., Kaplan-Meier survival analysis and GWAS. We present the results for these two tasks performed on real datasets from two peer-reviewed studies that were originally conducted by centralizing the data from multiple institutions.

As in Chapter 5, we implemented our solution by building on top of Lattigo [159], an open-source Go library for lattice-based cryptography, and on Onet [182], an open-source Go library for building decentralized systems. The communication between data providers (DPs) is done through TCP, with secure channels (by using TLS). We evaluate our prototype on an emulated realistic network, with a bandwidth of 1 Gbps and a delay of 20 ms between every two nodes. We deploy our solution on 12 Linux machines with Intel Xeon E5-2680 v3 CPUs running at 2.5GHz with 24 threads on 12 cores and 256 Gigabytes of RAM, on which we evenly distribute the DPs. We choose security parameters to always achieve a security level of 128 bits.

6.3.1 Multi-centric Kaplan-Meier Survival Analysis

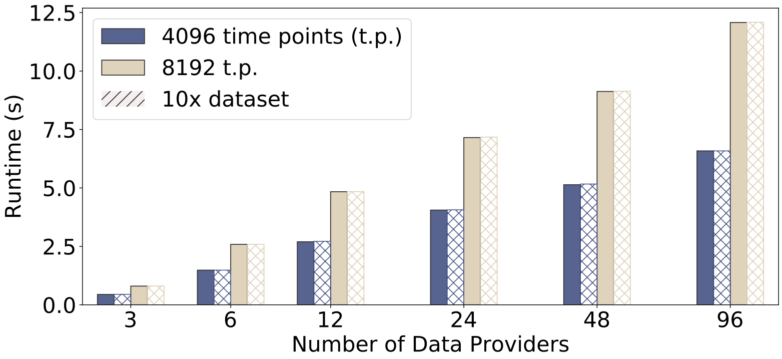
In a recent study, Samstein et al. [207] demonstrated that the tumor mutational burden (TMB) is a predictor of clinical response to immune checkpoint inhibitor (ICI) treatments in patients with metastatic cancers. To obtain this conclusion, they computed Kaplan-Meier overall survival (OS) curves of 1,662 advanced-cancer patients treated with ICI and that are stratified by TMB values. OS was measured from the date of first ICI treatment to time of death or the last follow-up. In Figure 6.3a, we show the survival curves obtained from the original centralized study (*Centralized, Non-secure*) and those obtained through our privacy-preserving federated workflow of FAMHE executed among three data providers (DPs). Note that for FAMHE, to illustrate the workflow of federated collaboration, we distributed the dataset across the DPs, each hosted on a different machine. FAMHE's analysis is then performed with each DP having access only to the locally held patient-level data, thus closely reflecting a real collaboration setting that involves independent healthcare centers. As a result, our federated solutions circumvent the privacy risks associated with data centralization in the original study. We observed that FAMHE produces survival curves identical to those of the original non-secure approach. By using either approach, we are able to derive the key conclusion that the benefits of ICI increase with TMB.

In Figure 6.3b, we show that FAMHE produces exact results while maintaining computational efficiency, as the computation of the survival curves shown in Figure 6.3a is executed in less than 12 seconds, even when the data are scattered among 96 DPs. We also observe that the execution time is almost independent of the DPs' dataset size, as the same experiment performed on a 10x larger dataset (replicated 10x) takes almost exactly the same amount of time. We show that FAMHE's execution time remains below 12 seconds for up to 8192 time points. We note that, in this particular study, the number of time points (instants at which an event can occur) is smaller than 200, due to the rounding off of survival times to months. In summary, the FAMHE-based Kaplan-Meier estimator produces precise results and scales efficiently with the number of time points, each DPs' dataset size, and with the number of DPs. We remark that the hazard ratio, which is often computed in survival curve studies, can be directly estimated by the querier, based on the final result [231]. It is also possible to compute the hazard ratios directly by following the general workflow of FAMHE described in Figure 6.1. This requires the training of proportional hazard regression models that are closely related to generalized linear models [133] that our GWAS solution in the next section also utilizes.



Time [months]	0	12	24	36	48
Bottom 80%	1302	584	230	85	33
Top 10%-20%	186	103	41	17	4
Top 10%	172	99	41	13	4

(a) Survival Curves



(b) FAMHE's runtime with an increasing number of DPs.

Figure 6.3 – Secure and Distributed Reproduction of a Survival-Curve Study. **(a)**: survival curves generated in a centralized non-secure manner and with FAMHE on the data used by Samstein et al. [207]. With FAMHE, the original data are split among three data providers, and the querier obtains exact results. The table in Figure 6.3a displays the number of patients at risk in a specific time. The exact same numbers are obtained with the centralized, non-secure solution and with FAMHE. **(b)**: FAMHE execution time for the computation of one (or multiple) survival curve(s) with a maximum of 8192 time points. For both the aggregation and key switching (from the collective public key to the querier's key), most of the execution time is spent in communication (up to 98%), as the operations on the encrypted data are lightweight and parallelized on multiple levels, i.e., among the data providers and among the encrypted values.

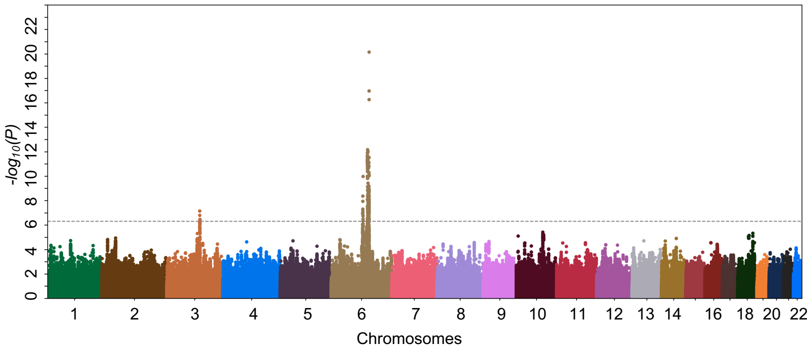
6.3.2 Multi-centric Genome-Wide Association Study (GWAS)

Here we demonstrate the potential of FAMHE to enable multi-centric GWAS that fully protect the privacy of patients' data throughout the analysis. We evaluated our approach on a GWAS dataset from McLaren et al. [151]; they studied the host genetic determinants of HIV-1 viral load in an infected population of European individuals. It is known that the viral load observed in an asymptomatic patient after primary infection positively correlates with the rate of disease progression; this is the basis for the study of how host genetics modulates this phenotype. We obtained the available data for a subset of the cohort including 1,857 individuals from the Swiss HIV Cohort Study, with 4,057,178 genotyped variants. The dataset also included 12 covariates that represent ancestry components, which we also used in our experiments to correct for confounding effects. To test our federated analysis approach, we distributed, in a manner analogous to the survival analysis experiments, the GWAS dataset across varying numbers of data providers.

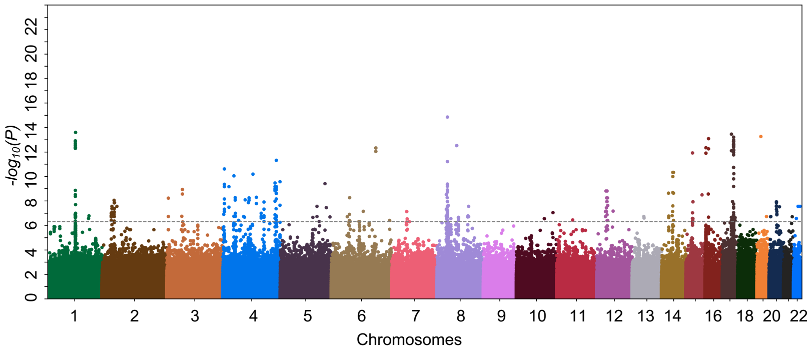
Following the approach of McLaren et al. [151], we performed GWAS using linear regression of the HIV-1 viral load on each of the more than four million variants, always including the covariates. To enable this large-scale analysis in a secure and federated manner, we developed two complementary approaches based on our system: FAMHE-GWAS and FAMHE-FastGWAS. FAMHE-GWAS performs exact linear regression and incurs no loss of accuracy, whereas FAMHE-FastGWAS achieves faster runtime through iterative optimization at a small expense of accuracy. We believe that both modes are practical and that the choice between them would depend on the study setting.

We compare FAMHE-GWAS and FAMHE-FastGWAS against (i) *Original*, the centralized non-secure approach adopted by the original study, albeit on the Swiss HIV Cohort Study datase, (ii) *Meta-analysis* [193], a solution in which each DP locally and independently performs GWAS to obtain summary statistics that are then shared and combined (through weighted-Z test) across DPs to produce a single statistic for each variant that represents its overall association with the target phenotype, and (iii) *Independent*, a solution in which a data provider uses only its part of the data set to perform GWAS. For all baseline approaches, we used the PLINK [193] software to perform the analysis. For (i) and (iii), we relied on PLINK 2.0 and its linear regression (`-glm` option) based association test. For (ii), we relied on PLINK 1.9 and used the weighted-Z test approach to perform the meta-analysis. Note that *Meta-analysis* can also be securely executed by first encrypting each DP's local summary statistics then following the federated-analytics workflow presented in Figure 6.1.

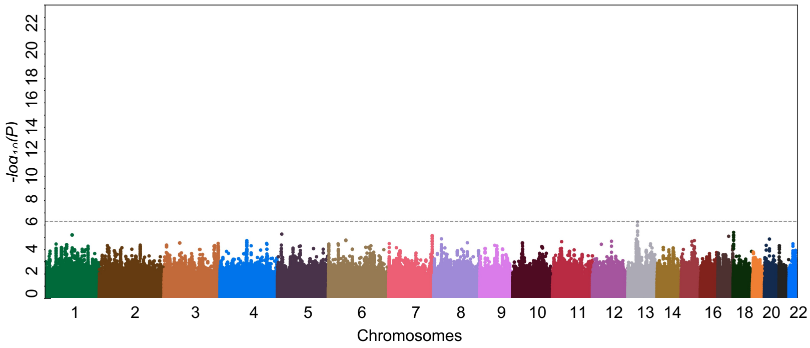
The Manhattan plots visualizing the GWAS results obtained by each method are shown in Figures 6.4 and 6.5. Both our methods (Figure 6.5), based on FAMHE, produced highly accurate outputs that are nearly indistinguishable from the *Original* results (Figure 6.4a). Consequently, our methods successfully implicated the same genomic regions with genome-wide significance found by *Original* (Figure 6.4a), represented by the strongest associated SNPs rs7637813 on chromosome 3 (nominal $p = 7.2 \times 10^{-8}$) and rs112243036 on chromosome 6 ($p = 7.0 \times 10^{-21}$). Notably, both these SNPs are in close vicinity of the two strongest signals reported by the original study [151]: rs1015164 with a distance of 9 Kbp and rs59440261 with a distance of 42 Kbp, respectively. The former is found in the major histocompatibility complex (MHC) region, and the latter is near the *CCR5* gene; both have established connections to HIV-1 disease



(a) *Original* Approach: Ground-truth obtained on a centralized cleartext dataset by relying on the PLINK [193] software.



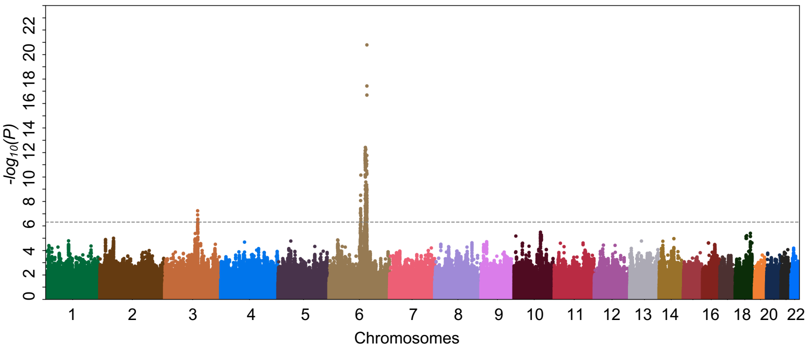
(b) *Meta-analysis* Approach: obtained with the PLINK [193] software.



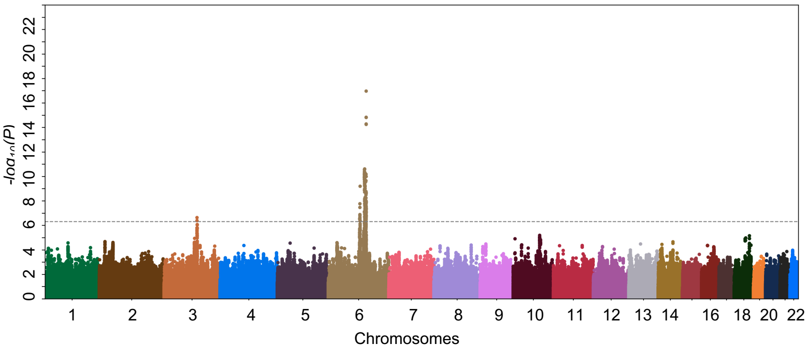
(c) Independent Approach: obtained, with the PLINK [193] software, by a single DP on its local subset of the global datsaset.

Figure 6.4 – Baseline GWAS results obtained with different non-secure approaches with 12 DPs (when applicable). (a) *Original* is considered as the ground-truth and is obtained on a centralized cleartext dataset by relying on the PLINK [193] software. (b) and (c) are also obtained with PLINK (See Figure D.2 for all DPs independent results). In the original study, genome-wide signals of association ($log_{10}(P) < 5 \times 10^{-7}$, dotted line) were observed on chromosomes 6 and 3.

progression. [151] Although the two previous SNPs were not available in our data subset to be analyzed, we reasonably posit that our findings capture the same association signals as in the original study, related through linkage disequilibrium. Regardless, we emphasize that our



(a) FAMHE-GWAS



(b) FAMHE-FastGWAS

Figure 6.5 – FAMHE GWAS results obtained with different approaches with 12 DPs. (a) and (b) are the results obtained with FAMHE-GWAS and FAMHE-FastGWAS, respectively. In our secure approach, as in the original study, genome-wide signals of association ($log_{10}(P) < 5 \times 10^{-7}$, dotted line) were observed on chromosomes 6 and 3.

federated analysis results closely replicated the centralized analysis of the same dataset we used in our analysis.

In contrast, the *Meta-analysis* approach (Figure 6.4b), though successfully applied in many studies, severely underperformed in our experiments by reporting numerous associations that are likely spurious. We believe this observation highlights the limitation of meta-analyses when the sample sizes of individual datasets are limited. Similarly, the *Independent* approach (Figure 6.4c) obtained noisy results, which was further compounded by the issue of limited statistical power (for results obtained by every data provider, see Figure D.2 in Appendix D.2). We complement these comparisons with Table 6.1 that quantifies the error in the reported negative logarithm of p-value (-$\log_{10}$(P-val)), as well as the regression weights (w), for all of the considered approaches compared to *Original*. We observed that FAMHE-FastGWAS yields an average absolute error always smaller than 10^{-2}, which ensures accurate identification of association signals. FAMHE-GWAS further reduces the error by roughly a factor of three to obtain even more accurate results. Whereas, *Meta-analysis* and *Independent* approaches result in considerably larger errors.

		Indep.		Meta-ana.		FAMHE-FastGWAS		FAMHE-GWAS	
		$-\log_{10}$(P-val)	w	$-\log_{10}$(P-val)	w	$-\log_{10}$(P-val)	w	$-\log_{10}$(P-val)	w
3 DPs	all	0.369	0.04	0.448	0.04	$6.7e^{-3}$	$1.5e^{-3}$	$2.72e^{-3}$	$7.3e^{-4}$
	peaks	4.14	0.055	7.9	0.19	0.71	$6.61e^{-3}$	0.1392	$1.88e^{-7}$
6 DPs	all	0.409	0.0665	0.45	0.041	$8.3e^{-3}$	$1.61e^{-3}$	$2.78e^{-3}$	$7.4e^{-4}$
	peaks	4.86	0.12	7.95	0.195	0.82	$6.63e^{-3}$	0.1393	$2.3e^{-7}$
12 DPs	all	0.425	0.104	0.453	0.048	$9e^{-3}$	$1.63e^{-3}$	$2.79e^{-3}$	$7.7e^{-4}$
	peaks	6.619	0.126	7.99	0.197	0.848	$6.69e^{-3}$	0.1399	$3.6e^{-7}$

Table 6.1 – GWAS Errors. This table displays the absolute averaged error on the logarithm of the p-values ($-\log_{10}$(P-val)) and on the model weights (w) between *Original* and federated approaches. It also shows that one data provider performing the GWAS alone, with only its local data (Indep.), obtains inaccurate results. For each number of data providers, we report the error averaged over all positions and the errors on the peaks identified with *Original* (see Figure 6.4a).

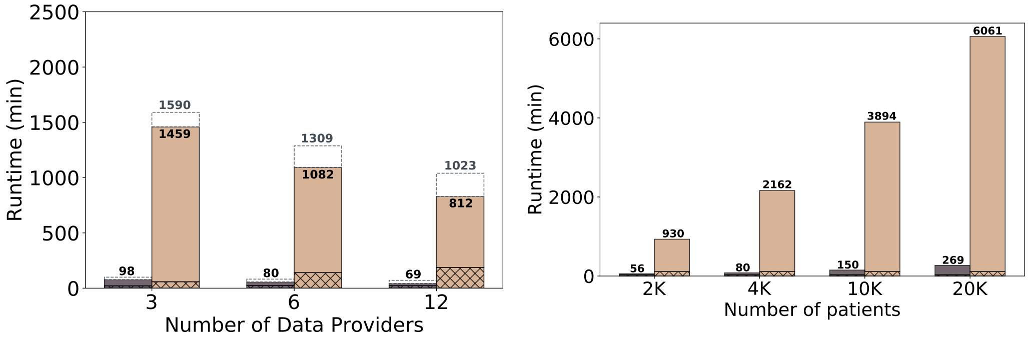
Figure 6.6 shows that FAMHE scales efficiently in all dimensions: number of data providers, samples and variants. As displayed by Figure 6.6a, FAMHE's runtime decreases when the workload is distributed among more data providers, and it is below one hour for a GWAS jointly performed by 12 data providers on more than 4 million variants with FAMHE-FastGWAS. It also shows that in a wide-area network (WAN) where the bandwidth is halved (from 1Gbps to 500Mbps) and the delay doubled (from 20ms to 40ms), FAMHE execution time increases by a maximum of 26% over all experiments.

Figures 6.6b and 6.6c show that FAMHE execution time grows linearly with the number of patients (or samples) and variants. In all experiments, the communication accounts for between 4 and 55 percent of FAMHE total execution time. As described in Section 6.2.3, FAMHE computes the p-values of multiple (between 512 and 8192) variants in parallel, due to the *Single Instruction, Multiple Data (SIMD)* property of the cryptoscheme and is further parallelized among the DPs and by multi-threading at each DP. FAMHE is therefore highly parallelizable, i.e., doubling the number of available threads would almost halve the execution time. Finally, FAMHE-GWAS, which performs exact linear regression, further reduces the error (by a factor of 3x compared to FAMHE-FastGWAS), but its execution times are generally higher than FAMHE-FastGWAS.

These results demonstrate the ability of FAMHE to enable the execution of FA workflows, on data held by large numbers of data providers who keep their data locally, while allowing for end-to-end privacy with no loss of accuracy.

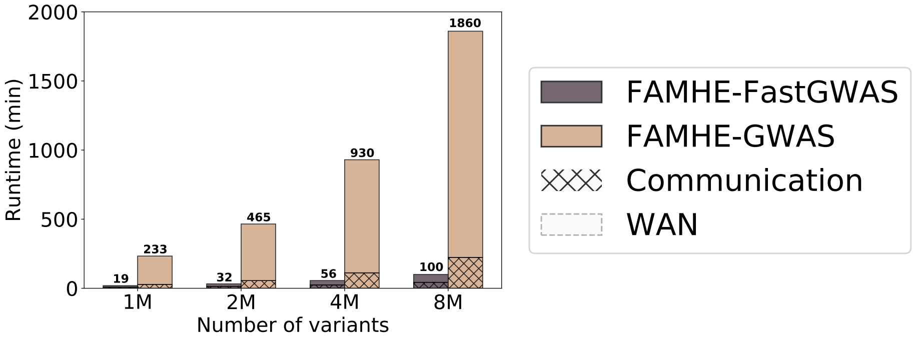
6.4 Comparison With Existing Works

Conceptually, FAMHE represents a new approach to federated analytics; it has not been previously explored for complex biomedical tasks. FAMHE combines the strengths of both conventional federated-learning approaches and cryptographic frameworks for secure compu-



(a) FAMHE's scaling with the number of DPs.

(b) FAMHE's scaling with the number of patients.



(c) FAMHE's scaling with the number of variants.

(d) Legend Box

Figure 6.6 – FAMHE Scaling. (a) FAMHE's scaling with the number of data providers, (b) with the size of the dataset and (d) with the number of variants considered in the GWAS. (d) is the legend box for ((a), (b), (c)). In (a), we also observe the effect of a reduced available bandwidth (from 1Gbps to 500Mbps) and increased communication delay (from 20ms to 40ms) on FAMHE's execution time. Unless otherwise stated, the original dataset containing 1857 samples and 4 million variants is evenly split among the data providers. By default, the number of DPs is fixed to 6.

tation. Like federated learning, FAMHE scales to large numbers of data providers and enables non-interactive local computation over each institution's dataset (available locally in cleartext), which minimizes the computational and communication burdens that cryptographic solutions [49; 116; 148; 202] typically suffer from. However, FAMHE draws from the cryptographic framework of MHE to enable secure aggregation and local computation of intermediate results in an encrypted form. This departs from the existing federated learning solutions [29; 90; 137; 168; 218; 223] that largely rely on data obfuscation to mitigate leakage in the intermediate data shared among the institutions. Our approach thus provides more rigorous privacy protection. In other words, in FAMHE accuracy is traded off only with performance, similarly to non-secure federated approaches, but differently from obfuscation-based solutions, FAMHE's security is absolute.

We summarize our comparison of FAMHE with existing works in Table 6.2. We compare certain existing solutions and FAMHE on multiple criterion: whether (i) patient levels and (ii) aggregated data are protected, (iii) the data protection satisfies the GDPR definition of anonymity, (iv) the solutions scale with the number of data providers and computing parties, (v) a majority of computing parties can be dishonest (without deviating from the protocol), (vi) the obtained results are the same as if they were computed on a centralized cleartext dataset (i.e., utility), (vii) the system enables multiple types of computations, and (viii) the solution is tested in a real application scenario. A "~" means that the property is partially fulfilled.

	Protect. Mecha.	Patient Level Data Protect.	GDPR Anonym.	Intermed. Val. Protect.	Scale w. Parties	Passive Dishonest Majority	Preserve Utility	Comput. Flexibility	Complex Comput.	Appl. To Real Cases
AllOfUs [8] Gen. Eng. [92] UKBio. [236]	None	No	No	No	~	No	Yes	Yes	Yes	Yes
DataSHIELD [90] Vantage6 [238] SHRINE [222] Splink [168] SWARM [244]	Aggr.	~	No	No	Yes	No	~	~	Yes	~
Bonomi et al. [29]	DiffP	Yes	No	~	Yes	Yes	No	No	No	~
Li et al. [137]	DiffP	Yes	No	~	Yes	Yes	No	No	Yes	No
Cho et al. [49]	SMC	Yes	Yes	Yes	No	Yes	~	No	Yes	Yes
Jagadeesh et al. [116]	SMC	Yes	Yes	Yes	No	No	~	No	No	Yes
Lu et al. [148]	MHE	Yes	Yes	Yes	Yes	Yes	~	~	No	No
FAMHE	MHE	Yes	Yes	Yes	Yes	Yes	~	~	Yes	Yes

Table 6.2 – Comparison of existing solutions for biomedical FA.

Quantitative comparison of FAMHE-GWAS and FAMHE-FastGWAS with existing approaches. FAMHE is specifically designed to benefit from its multiparty construction and to optimize its use of MHE to efficiently execute secure FA workflows among a large number of data providers that keep their data locally. As shown in Table 6.3, *a non-secure centralized solution* based on PLINK takes 14 seconds when performed by a single data provider on the pooled dataset. As shown in Figure 6.6a, FAMHE efficiently distributes its workload and achieves an execution time of 69 minutes when the data are split among 12 DPs for the same computation. *A non-secure federated solution* based on PLINK's meta-analysis method takes around 5 minutes when executed on 12 DPs but yields very imprecise results. We remark that in the meta-analysis the DPs exchange information only once at the end and a non-secure solution in which the DPs collaborate during the process would be slightly slower, e.g., between 5 and 10 minutes, depending on the communication settings. *Differential-privacy-based solutions* usually yield the same execution time as non-secure solutions. We estimate that a *centralized HE-based solution* would take at least 2228 (with the FastGWAS approach) and 30,781 minutes (with the GWAS approach), whereas FAMHE respectively takes 69 and 812 minutes when the same approaches are collaboratively executed by 12 data providers. The centralized approach, contrary to FAMHE, cannot distribute the workload among multiple DPs and suffers from a high overhead brought by centralized cryptographic operations. For example, in a centralized setting, a ciphertext is refreshed or bootstrapped in 26 seconds [106] for a security level of 108 bits, whereas the corresponding interactive protocol in FAMHE takes 0.6 seconds with a better security level of 128 bits. *SMC approaches* usually target settings with 2 to 4 parties due to a communication cost that becomes prohibitive for higher numbers of parties. FAMHE also works with 2 to 4 parties but is not specifically optimized for this scenario and its execution time would be in the same order of magnitude as secret-sharing-based solutions. However, unlike secret-sharing-based solutions, FAMHE efficiently scales to federated learning settings where many DPs keep their data locally. Furthermore, by designing an alternative MHE-friendly algorithm for the same task (e.g. FAMHE-FastGWAS), FAMHE can achieve faster execution times than SMC even in the setting with a small number of parties.

We summarize this comparison in Table 6.3. The two *Decentralized* approaches refer to cleartext adaptations of FAMHE's approaches. The results for *Centralized, Non-Secure* and *Meta-analysis, Non-Secure* are obtained through experiments with the PLINK software. The execution times for the *Decentralized* approaches are inferred from *Meta-analysis, Non-Secure* and their communication overheads are deduced from FAMHE's communication costs from which we removed the encryption overhead. We also relied on this encryption storage overhead to estimate *Centralized HE* communication cost, whereas its execution time is deducted from our previous observation on the centralized bootstrapping. We estimated the performance of SMC by extrapolating the published results of Cho et al. [49]. For a fair comparison, we considered only the Phase 3 results, i.e., the association tests, for which Cho et al. computed the χ^2 statistics of the generalized Cochran-Armitage trend test corrected for covariates. We note that, while Cho et al. take a different approach to computing the association results than FAMHE, the overall workflow is similar and represents a suitable reference point for comparison. Since the prior work considered a three-party setting, we expect the real execution time of SMC to be higher in our setting with 12 DPs. For the communication cost of SMC, we combined the estimates of both initial data sharing and the main computation step (Phase 3) from the original publication and assumed linear scaling with the number of DPs. FAMHE measurements are obtained through our experiments.

	Centralized, Non-Secure	Meta-analysis, Non-Secure	Decentralized Non-secure	Decentralized, DiffP.	Centralized HE	SMC	FAMHE
Execution Time (min)	0.24	5	10	10	2228; 30781	320	69; 812
Communication cost (GB)	4	0.5	39; 171	39; 171	12	3144	122; 684

Table 6.3 – Quantitative comparison of existing approaches for biomedical FA. In distributed cases, the data are split among 12 data providers. The communication cost corresponds to the amount of data that one DP has to send during the entire process. When two values are provided, the left one corresponds to an approach equivalent to FastGWAS and the right one to GWAS.

6.5 Conclusion

In this chapter, we demonstrate that efficient privacy-preserving federated-analysis workflows for complex biomedical tasks are attainable. Our efficient solutions for survival analysis and GWAS, based on FAMHE, accurately reproduced published peer-reviewed studies while keeping the dataset distributed across multiple sites and ensuring that the shared intermediate-data do not leak any private information. Alternative approaches based on meta-analysis or independent analysis of each dataset led to noisy results in our experiments, illustrating the benefits of our federated solutions. The fact that FAMHE led to practical federated algorithms for both the statistical calculations required by Kaplan-Meier curves and the large-scale regression tasks of GWAS reflects the ability of FAMHE to enable a wide range of other analyses in biomedical research, such as cohort exploration and the training and evaluation of disease risk prediction models.

The fact that FAMHE shares only encrypted data among the data providers has an important implication for its suitability to regulatory compliance and its potential to catalyze future efforts for multi-centric biomedical studies. In recent work, it has been established by privacy law experts that data processed using MHE can be considered ‘anonymous’ data under the General Data Protection Regulation (GDPR) [210]. Anonymous data, which refers to data that require unreasonable efforts to reidentify the source individuals, lies outside the jurisdiction of GDPR. Therefore, our approach has the potential to significantly simplify the requirements for contractual agreements and the obligations of data controllers with respect to regulations that, such as GDPR, often hinder multi-centric medical studies. In contrast, existing federated-analytics solutions, where the intermediate results are openly shared, present more complicated paths toward compliance, as intermediate results could still be considered personal data [157; 169; 243].

In cases where the potential leakage of privacy in the final output of federated analysis is a concern, differential privacy techniques can be easily incorporated into FAMHE by adding a small perturbation to the final results before they are revealed. In contrast to the conventional federated learning approach, which requires each data provider to perturb its local results before aggregating them with other parties, FAMHE enables the data providers to keep the local results encrypted and reveals only the final aggregated results. Therefore, FAMHE can use a smaller amount of added noise and achieve the same level of privacy [125]. Notably, the choices of differential-privacy parameters suitable for analyses with a high-dimensional output, such as GWAS, can be challenging, and needs to be further explored.

There are several directions in which our work could be extended to facilitate the adoption of FAMHE. Although, we reproduced published studies by distributing a pooled dataset across a group of data providers, jointly analyzing multiple datasets by using FAMHE that could not be combined otherwise would be a challenging yet important milestone for this endeavour. Our work demonstrates FAMHE’s applicability on a reliable baseline and constitutes an important and necessary step towards building trust of our technology and fostering its adoption, thus enabling its use for the discovery of new scientific insights. Furthermore, we will extend the capabilities of FAMHE by developing additional protocols for a broader range of standard analysis tools and machine-learning algorithms in biomedical research (e.g., proportional-hazard regression models). A key step in this direction is to make our implementation of FAMHE easily configurable by practitioners for their own applications. Specifically, connecting FAMHE to existing user-friendly platforms such as MedCo [155] to make it widely available would help empower the increasing efforts to launch multi-centric medical studies and accelerate scientific discoveries.

7 Conclusion

In this thesis, we have proposed a new approach for privacy-preserving federated data analytics. In order to facilitate their adoption, we have designed secure solutions that behave (e.g., scale) similarly to federated non-secure approaches for data analysis; they also provide equivalent flexibility and features. In other words, we have proposed solutions that scale to a large number of data providers, do not introduce by-design bias in the final results, and enable data providers to keep control over their data. These features are attained with solutions that also ensure strong security guarantees in restrictive threat models. Our solutions provide a similar trade-off between performance and accuracy, e.g., depending on the number of training iterations in ML, as non-secure solutions. They do not impose a trade-off between privacy and accuracy as, for example, in the case of differential privacy-based solutions. The performance overhead imposed by the added-security in our solutions can be tightly controlled because our systems are designed to be modular and their security features, e.g., proofs of correctness, can be enforced or not. We have shown that our solutions follow the same scaling trend with respect to the number of data providers, the computation complexity, and the input-data sizes, as non-secure solutions and that they can achieve execution times between 0 and 2 orders of magnitude slower than insecure solutions.

Contrary to most of existing cryptography-based solutions, our systems scale to a large number of data providers that can keep their data locally and efficiently collaborate to perform data analytics that range from simple sums to complex and parallelized ML models training, e.g., GWAS. This is made possible by the combination of secure multiparty computation and homomorphic encryption; this is at the core of all our solutions and enables us to propose efficient and secure decentralized systems.

We hope that the proposed generic systems, e.g., enabling the widely-applicable gradient descent, will pave the way to even more flexible secure solutions that will provide high-security guarantees with a negligible and almost imperceptible cost to the users, thus favoring their adoption. Moreover, privacy law experts established that data processed using MHE can be considered ‘anonymous’ and thus lie outside the jurisdiction of the General Data Protection Regulation (GDPR). Our approach has therefore the potential to significantly simplify the requirements for contractual agreements and the obligations of data controllers with respect to regulations that often hinder data sharing.

In Chapter 3, we have introduced a first system that, by building on multiparty additive homomorphic encryption (AHE) and differential privacy, enables the privacy-preserving execution of SQL-like queries on a dataset jointly held by a multitude of data providers. We have shown that, by designing a modular system, we are able to provide a solution that is efficient in a strong threat model and whose query-response time can be further reduced depending on the settings, on the security features, and on the threat model considered, e.g., with or without local data encryption and with an honest or malicious threat model.

In Chapter 4, we have further built on the solution proposed in Chapter 3. We designed a system that provides richer functionalities, e.g., complex statistic computations, and that preserves and enhances the provided security guarantees. As in Chapter 3, DRYNX is modular and can be optimized for specific settings or threat models. Furthermore, we have proposed an efficient and probabilistic method for verifying the system's inputs and computations, thus making it fully auditable without affecting its response time, i.e., the query response is sent independently of the verification that is performed in parallel and a posteriori.

In Chapter 5, we have adapted a state-of-the-art quantum-resistant fully-homomorphic encryption cryptoscheme to our multiparty framework. By relying on this multiparty construction and computation capabilities, we were able to efficiently perform a federated stochastic gradient descent on data held by many data providers and to preserve data and model confidentiality. SPINDLE covers the entire ML workflow, as the trained model can remain secret to perform oblivious predictions. SPINDLE notably optimizes its performance by selecting the best packing and multiplication method, depending on the input-data dimensions and the learning parameters, and by replacing costly cryptographic operations, e.g., bootstrapping, with lightweight interactive protocols.

In Chapter 6, we have shown that the previously introduced framework can be used to efficiently and accurately perform computationally demanding tasks, such as the training in parallel of millions of regression models, and that it can be used in domains, such as precision medicine, where accuracy is crucial. We have demonstrated that, by relying on multiparty homomorphic encryption, we are able to faithfully and accurately reproduce biomedical studies, from the computation of survival curves to the execution of a GWAS with more than 4 millions variants, even when the data are distributed among many data providers.

Our approach improves on state-of-the-art secure solutions by scaling to large numbers of data providers that can keep control of and benefit from their local data and by providing strong security guarantees, e.g., quantum-resistance, in strong threat models. We provide these features with solutions that remain modular in terms of enabled-computations (e.g., statistics and ML models training) and that can be adapted to many scenarios, as some of their security features (e.g., differential privacy) can be ensured in a modular way. Our approach can conceptually be used for any federated computation. We are currently building on this work to develop a solution for principal component analysis (e.g., for dimension reduction) that is a key pre-processing operation in many ML workflows, (e.g., GWAS). We have also relied on this approach for the training and evaluation of neural-network models [208]. The extension of our approach, for the efficient training of more complex ML models, is also an interesting and promising direction for future work. Other crucial directions of research are towards usability and parametrization of secure systems for federated analytics. In Chapter 5, we have described the intrinsic links between the security (or cryptography) parameters and the learning parameters, but their setup remains non-trivial and requires the user to understand both the cryptographic and ML constraints. Learning how to automatize cross-domain parameters selection in a federated and privacy-preserving manner is a key-step towards the design of more user-friendly solutions.

In conclusion, we believe that our multiparty homomorphic encryption-based framework can be used in a multitude of scenarios and applications due to its computational and flexibility features that bring it closer to standard insecure solutions. Personalized and precision medicine are paradigmatic examples in which our solutions can help to accelerate progress by providing a combination of analytical and security features that are absolutely needed to enable large-scale studies and collaborations with sensitive data. We hope to facilitate the adoption of secure solutions for federated analytics in general by ensuring strong security guarantees that do not have a negative impact on the usability and the performance for the users. We believe that our approach can be a key enabler in a multitude of domains in which data sharing or federated analytics are currently difficult or non-existent due to the sensitivity of the data.

Appendix Part I

A Federated Data Exploration

In this appendix, we extend the explanations, of Chapter 3, on zero-knowledge proofs and the ‘`SELECT *`’ feature of UNLYNX.

A.1 Zero-Knowledge Proofs

We complement the explanation of the zero-knowledge proofs introduced in Section 3.3.2.

Distributed Deterministic-Tag

We recall the distributed deterministic-tag sub-protocol below.

This sub-protocol is made of two consecutive rounds. It starts with $E_K(x) = (C_1,\ C_2) = (rB,\ x + rK)$, the ciphertext tuple corresponding to an ElGamal encryption of message x that uses the collective authority's (CA) public key K.

In the first round, each computing node sequentially generates a fresh secret s_i and adds the value derived from its secret s_iB to C_2. This eliminates the possibility to have a deterministic tag of 0 as an output of the protocol when the input message is zero. After this first round, the encrypted message is $(C_1,\ C_2) = (rB,\ x + rK + \sum_{i=1}^{|CN|} s_iB)$. Let $(\tilde{C}_{1,0},\ \tilde{C}_{2,0}) = (C_1,\ C_2)$ be a ciphertext resulting from the first round.

In the second round, each computing node partially and sequentially modifies this ciphertext. More specifically, when computing node CN_i receives the modified ciphertext $(\tilde{C}_{1,i-1},\ \tilde{C}_{2,i-1})$ from computing node CN_{i-1}, it computes $(\tilde{C}_{1,i},\ \tilde{C}_{2,i})$ as

$$\tilde{C}_{1,i} = s_i\tilde{C}_{1,i-1} \tag{A.1}$$

and

$$\tilde{C}_{2,i} = s_i\left(\tilde{C}_{2,i-1} - \tilde{C}_{1,i-1}k_i\right) \tag{A.2}$$

Once all of these computations are done, we discard the first component $\tilde{C}_{1,|CN|}$ and obtain

$$\tilde{C}_{2,|CN|} = sx + \sum_{i=1}^{|CN|} s_i sB \tag{A.3}$$

where $s = \prod_{i=1}^{|CN|} s_i$ is a short-term collective secret corresponding to the product of each computing node's fresh secret. $\tilde{C}_{2,|CN|}$ is the deterministic tag collectively computed from the original ciphertext (C_1, C_2).

Each time a computing node does the computations in the first round and in Equations (A.1) and (A.2), it must also compute a zero-knowledge proof to prove that the computations have been done correctly. In this case, when adding a secret value and when computing $(\tilde{C}_{1,i}, \tilde{C}_{2,i})$, computing node CN_i is the prover and anybody can act as a verifier. In the first round, the prover proves that he knows s_i the discrete logarithm of $s_i B$. Coming back to Equation (1.1) in Section 1.2, it is easy to see that for Equation (A.2), $y_1 = s_i$, $y_2 = k_i s_i$ are the discrete logarithms of $s_i B$ and $k_i s_i B = s_i K_i$, respectively. The points $s_i K_i$, $s_i B$, $A = \tilde{C}_{2,i}$, $A_1 = \tilde{C}_{2,i-1}$ and $A_2 = -\tilde{C}_{1,i-1}$ on $\mathscr{G}$ are public and are part of the proof. The publication of $s_i B$ also guarantees that computing node CN_i has used the same secret s_i for all data during a given query. This means that for each query, computing node CN_i will pick a value s_i that will be used throughout the query and will be different for the next query.

For Equation (A.1), $y_1 = s_i$ is the discrete logarithm of $s_i \tilde{C}_{1,i-1}$. The points $\tilde{C}_{1,i}$, $s_i B$, $A = \tilde{C}_{1,i}$ and $A_1 = \tilde{C}_{1,i-1}$on $\mathscr{G}$ are public and are part of the proof.

Key Switch

We recall the key switch sub-protocol below.

We start with $\mathrm{E}_K(x) = (C_1, C_2) = (rB, x + rK)$, a ciphertext tuple corresponding to the ElGamal encryption of message x using the CA's public key K. Let $(\tilde{C}_{1,0}, \tilde{C}_{2,0}) = (0, C_{2,j})$ be the initial modified ciphertext tuple. Each computing node will partially and sequentially modify this element. Specifically, when computing node CN_i receives $(\tilde{C}_{1,i-1}, \tilde{C}_{2,i-1})$ from computing node CN_{i-1}, it generates a fresh random nonce v_i and computes $(\tilde{C}_{1,i}, \tilde{C}_{2,i})$ as

$$\tilde{C}_{1,i} = \tilde{C}_{1,i-1} + v_i B \tag{A.4}$$

and

$$\begin{aligned} \tilde{C}_{2,i} &= \tilde{C}_{2,i-1} - (r_j B) k_i + v_i U \\ &= \tilde{C}_{2,i-1} - r_j K_i + v_i U. \end{aligned} \tag{A.5}$$

where $v = v_1 + \ldots + v_{|CN|}$.

Each time a computing node does the computations in Equations (A.4) and (A.5), it must also compute a zero-knowledge proof to prove that the computations have been done correctly. Again, at each step i, computing node CN_i is the prover and anybody can be the verifier. Coming back to Equation (1.1) in Section 1.2, it is easy to see that for Equation (A.5), $y_1 = k_i$, $y_2 = v_i$ are the discrete logarithms of $k_i B = K_i$ and $v_i B$, respectively. All points K_i, $v_i B$, $A = \tilde{C}_{2,i} - \tilde{C}_{2,i-1}$, $A_1 = -r_j B$ and $A_2 = U$ are made public and do not leak any information about underlying secrets.

For Equation (A.4), $y_1 = v_i$ is the discrete logarithm of $v_i B$. All points $v_i B$, $A = \tilde{C}_{1,i} - \tilde{C}_{1,i-1}$ and $A_1 = U$ are made public and do not leak any information about underlying secrets.

Dynamic Collective Authority

We recall the sub-protocol allowing to add/remove a computing node from the collective authority.

Let $CN_{|CN|}$ be the computing node that needs to be added. Any data provider that stored data encrypted using the CA's previous public key K_{prev} must execute the corresponding sub-protocol in order to have its data encrypted under the CA's new public key K_{new}. When adding a new computing node $CN_{|CN|}$ to the collective authority $CN_1,\ldots,CN_{|CN|-1}$, $K_{\text{prev}} = K_1+\ldots+K_{|CN|-1}$ and $K_{\text{new}} = K_1+\ldots+K_{|CN|}$. Starting from a message x encrypted under K_{prev}, $(C_1,\ C_2) = (rB,\ x+rK_{\text{prev}})$, computing node $CN_{|CN|}$ multiplies C_1 by its private key $k_{|CN|}$

$$C_1 k_{|CN|} = (rB)k_{|CN|} = rK_{|CN|} \tag{A.6}$$

and adds the result to C_2

$$\tilde{C}_2 = C_2 + rK_{|CN|} = x + rK_{\text{prev}} + rK_{|CN|} = x + rK_{\text{new}}. \tag{A.7}$$

Component $\tilde{C}_1$ remains the same, i.e.,

$$\tilde{C}_1 = C_1.$$

Coming back to Equation (1.1) in Section 1.2, we see that for Equation (A.7), $y_1 = k_{|CN|}$ is the discrete logarithm of $k_{|CN|}B = K_{|CN|}$. All points $K_{|CN|}$, $A = \tilde{C}_2 - C_2$ and $A_1 = C_1$ are made public and do not leak any information about underlying secrets.

Now, assume a collective authority of $|CN|$ computing nodes $CN_1,\ldots,CN_{|CN|}$ and let $CN_{|CN|}$ be the computing node that needs to be removed. In this case, $K_{\text{prev}} = K_1+\ldots+K_{|CN|}$ and $K_{\text{new}} = K_1+\ldots+K_{|CN|-1}$. In order to update the encryption of message x to the CA's new public key K_{new}, computing node $CN_{|CN|}$ must compute $\tilde{C}_2$ as

$$\tilde{C}_2 = C_2 - rK_{|CN|} = x + rK_{\text{prev}} - rK_{|CN|} = x + rK_{\text{new}}. \tag{A.8}$$

Again, using Equation (1.1) in Section 1.2, we see that for Equation (A.8), $y_1 = k_{|CN|}$ is the discrete logarithm of $k_{|CN|}B = K_{|CN|}$. All points $K_{|CN|}$, $A = \tilde{C}_2 - C_2$ and $A_1 = -C_1$ are made public and do not leak any information about underlying secrets. For Equation (A.6), $y_1 = k_{|CN|}$ is the discrete logarithm of $k_{|CN|}rB$ and the point $k_{|CN|}rB$ is public and do not leak any information about underlying secrets.

A.2 SELECT query

As mentioned in Section 3.2.1, UnLynx's design enables the system to respond to queries of the form '`SELECT *`'. The system would handle this query by executing all the steps of the Decentralized Data Sharing protocol 3.3.1 except Steps 5 and 6 that would be skipped.

While this can be useful for a data provider wishing to retrieve/decrypt parts/all of his database, UnLynx should not allow these queries on a distributed database held by multiple data providers. In fact, it is not possible to answer this request while preserving the privacy and data confidentiality of data providers because differential privacy cannot be ensured on non-aggregated data. Moreover, an external querier would have access to DP's raw data. Hence, our system should only allow this operation for a data provider querying his own database.

B Federated Statistical Analysis

In this appendix of Chapter 4, we detail the error probability of the bit-wise operations in DRYNX.

B.1 Error Probability

In Section 4.5, we notice that the result of bit-wise operations, when DPs are requested to answer with random values $R_i s$, can be erroneous with a probability smaller than $1/(\#G-1)$. We demonstrate here this result and provide an expression for the probability of error $P_{|S|}$ where $|S|$ is the number of DPs.

$$\begin{aligned}
P_{|S|} &= P(\sum_{i=1}^{|S|} R_i = 0) = \sum_{a=0}^{\#G-1} P(\sum_{i=1}^{|S|} R_i = 0 \mid \sum_{i=1}^{|S|-1} R_i = a) \cdot P(\sum_{i=1}^{|S|-1} R_i = a) \\
&= P(\sum_{i=1}^{|S|} R_i = 0 \mid \sum_{i=1}^{|S|-1} R_i = 0) \cdot P(\sum_{i=1}^{|S|-1} R_i = 0) \\
&+ \sum_{a=1}^{\#G-1} P(\sum_{i=1}^{|S|} R_i = 0 \mid \sum_{i=1}^{|S|-1} R_i = a) \cdot P(\sum_{i=1}^{|S|-1} R_i = a) \\
&= \sum_{a=1}^{\#G-1} P(\sum_{i=1}^{|S|} R_i = 0 \mid \sum_{i=1}^{|S|-1} R_i = a) \cdot P(\sum_{i=1}^{|S|-1} R_i = a) \\
&= P(R_n = -a) \cdot \sum_{a=1}^{\#G-1} P(\sum_{i=1}^{|S|-1} R_i = a) \\
&= \frac{1}{\#G-1} \cdot \sum_{a=1}^{\#G-1} P(\sum_{i=1}^{|S|-1} R_i = a) = \frac{1}{\#G-1} \cdot (1 - P_{|S|-1}).
\end{aligned}$$

We have $P_{|S|} = \frac{1}{\#G-1} \cdot (1 - P_{|S|-1}) \leq \frac{1}{\#G-1}$ and $P_{|S|} = \sum_{i=2}^{|S|} (-1)^i \cdot (\frac{1}{\#G-1})^{i-1}$.

C Federated Learning

In this appendix, we complement Section 5.4.3 in Chapter 5 by detailing how we approximate non-polynomial functions and how we rely on this technique to approximate the maximum function. We complement SPINDLE's security analysis (Section 5.6) by proving the security of the DBootstrap operation. We provide more details on SPINDLE's evaluation and propose some modular extensions that can be integrated in the system.

C.1 Activation Functions

We describe how we evaluate a polynomial approximation and how we approximate the maximum function.

Algorithm C.1 Encrypted Polynomial Approximation Evaluation AF$(\cdot)$.

```
Func. AF($\langle \boldsymbol{u} \rangle, d, \boldsymbol{r}$) outputs $\langle \boldsymbol{a} \rangle$ the evaluated poly. approx. of $\langle \boldsymbol{u} \rangle$
1: Choose the smallest $\omega$ such that $2^\omega > d$ and define $k = \lfloor \omega/2 \rfloor$
2: Compute $\{u_i\} = \langle \boldsymbol{u}^1 \rangle, \langle \boldsymbol{u}^2 \rangle, \dots, \langle \boldsymbol{u}^{2^k\text{-}1} \rangle, \langle \boldsymbol{u}^{2^k} \rangle, \langle \boldsymbol{u}^{2^{k+1}} \rangle, \dots, \langle \boldsymbol{u}^{2^{\omega\text{-}1}} \rangle$ inductively and call
   paRecu($\boldsymbol{r}, d, \{u_i\}$)
3: Function paRecu($\boldsymbol{r}, d, \{u_i\}$):
4:   Choose the smallest $\omega$ such that $2^\omega > d$
5:   Find polynomials $q(\langle \boldsymbol{u} \rangle)$ and $R(\langle \boldsymbol{u} \rangle)$ with $\langle \boldsymbol{a} \rangle =$
     $\langle \boldsymbol{u}^{2^{\omega-1}} \rangle q(\langle \boldsymbol{u} \rangle) + R(\langle \boldsymbol{u} \rangle)$ such that
     $\langle \boldsymbol{a} \rangle = \sum_{i=1,2,..,d} \boldsymbol{r}[i] \langle \boldsymbol{u}^i \rangle$
6:   If $d(q), d(R) > 2$:
       Evaluate $q(\langle \boldsymbol{u} \rangle) =$ paRecu($\boldsymbol{r}, d = d(q), \{u_i\}$)
       and $R(\langle \boldsymbol{u} \rangle) =$ paRecu($\boldsymbol{r}, d = d(R), \{u_i\}$)
7:   Else Return $\langle \boldsymbol{a} \rangle$
```

Polynomial Approximation. Algorithm C.1 inductively computes the (element-wise) exponentiation of the encrypted input vector $\langle \boldsymbol{u} \rangle$: $\langle \boldsymbol{u}^1 \rangle$, $\langle \boldsymbol{u}^2 \rangle$, …, $\langle \boldsymbol{u}^{2^k-1} \rangle$, $\langle \boldsymbol{u}^{2^k} \rangle$, $\langle \boldsymbol{u}^{2^{k+1}} \rangle$, …, $\langle \boldsymbol{u}^{2^{\omega-1}} \rangle$ (Algorithm C.1, line 2), where ω is the smallest value satisfying $2^\omega > d(p(\langle \boldsymbol{u} \rangle))$ and $k = \lfloor \omega/2 \rfloor$. Then, it recursively evaluates $p(\langle \boldsymbol{u} \rangle) = \sum_{i=1,2,3...,d} r_i \langle \boldsymbol{u}^i \rangle = \langle \boldsymbol{u}^{2^{\omega-1}} \rangle q(\langle \boldsymbol{u} \rangle) + R(\langle \boldsymbol{u} \rangle)$

(Algorithm C.1, line 3). Note that $p(\cdot)$, $q(\cdot)$, and $R(\cdot)$ are functions of $\langle \boldsymbol{u} \rangle$ and of the approximation coefficients $\boldsymbol{r}$, $q(\cdot)$ is the quotient of the division of the actual activation function $p(\cdot)$ by $\langle \boldsymbol{u}^{2^{\omega-1}} \rangle$, and $R(\cdot)$ is the remainder of the division. $d(x)$ is a function that outputs the degree of x.

Approximation of the maximum function. Algorithm C.2 computes the approximation of the maximum function. It takes an encrypted matrix $\langle \boldsymbol{U}_{|cl| \times c} \rangle$, the approximations intervals $[a_i, g_i]$ and degrees $\boldsymbol{d}$, and computes an encrypted vector $\langle \boldsymbol{m} \rangle$ that contains a close approximation of the max of each column of $\langle \boldsymbol{U} \rangle$.

Algorithm C.2 Approximation of the max function apMax(·).

```
$\langle \boldsymbol{m} \rangle \leftarrow$ apMax($\langle \boldsymbol{U} \rangle, [a_i, g_i], \boldsymbol{d}$)
$\langle \boldsymbol{u'} \rangle = \sum_{\lambda=0}^{|cl|} \langle \boldsymbol{U}[\lambda, \cdot] \rangle$
for $\lambda = 1, \ldots, |cl|$ do
    $\langle \boldsymbol{U}[\lambda, \cdot] \rangle = (\langle \boldsymbol{U}[\lambda, \cdot] \rangle - \langle \boldsymbol{u'}[\lambda, \cdot] \rangle)$
end for
$\boldsymbol{r} \leftarrow$ GetAFCoefficients($(1/h')e^{(x/h)}, [a_1, g_1], \boldsymbol{d}[1]$), where $h, h'$ are predefined constants
for $\lambda = 1, \ldots, |cl|$ do
    $\langle \boldsymbol{U''}[\lambda, \cdot] \rangle = \mathrm{AF}(\langle \boldsymbol{U}[\lambda, \cdot] \rangle, \boldsymbol{d}[1], \boldsymbol{r})$
end for
$\langle \boldsymbol{o} \rangle = \sum_{\lambda=0}^{|cl|} \langle \boldsymbol{U''}[\lambda, \cdot] \rangle$
$\boldsymbol{r'} \leftarrow$ GetAFCoefficients($\{1/x\}, [a_2, g_2], \boldsymbol{d}[2]$)
$\langle \boldsymbol{o} \rangle = \mathrm{AF}(\langle \boldsymbol{o} \rangle, \boldsymbol{d}[2], \boldsymbol{r'})$
for $\lambda = 1, \ldots, |cl|$ do
    $\langle \boldsymbol{U}[\lambda, \cdot] \rangle = \mathrm{M}(\langle \boldsymbol{U}[\lambda, \cdot] \rangle, \langle \boldsymbol{U''}[\lambda, \cdot] \rangle)$
end for
$\langle \boldsymbol{m} \rangle = \sum_{\lambda=0}^{|cl|} (\langle \boldsymbol{U}[\lambda, \cdot] \rangle, \langle \boldsymbol{o} \rangle)$
```

C.2 Security of DBootstrap

The original distributed bootstrapping protocol for BFV [79] is presented by Mouchet et al. [165]. In this protocol, the data providers produce an additive sharing of the encrypted ciphertext by masking their share in the decryption, before collectively encrypting their share to collectively produce a new (fresh) encryption of the same value. We adapted this protocol to the CKKS scheme [47]. The protocol steps remain the same but the underlying computational assumptions are different. In fact, in CKKS the shares created by the data providers are not unconditionally hiding, but statistically or computationally hiding due to the incomplete support of the used masks. The proof for the protocol's CKKS version (DBootstrap(·)) follows from the proof provided by Mouchet et al. in the passive-adversary model of the BFV bootstrapping protocol with the additional assumption that Lemma C.2.1 is true. This lemma guarantees the statistical indistinguishablity of the shares in $\mathbb{C}$. The RLWE problem is hard if the adversary is computationally-bounded, whereas Lemma C.2.1 relies on a statistical argument. However, both share the same security bound given the same security parameter and DBootstrap(·)

provides the same computational security as Mouchet et al. [165] original protocol.

Lemma C.2.1. *Given the distribution* $P_0 = (a+b)$ *and* $P_1 = c$ *with* $0 \leq a < 2^{\delta}$ *and* $0 \leq b, c < 2^{\lambda+\delta}$ *and* b, c *uniform, then the distributions* P_0 *and* P_1 *are* λ*-indistinguishable; i.e., a probabilistic polynomial adversary* $\mathscr{A}$ *cannot distinguish between both with probability greater than* $2^{-\lambda}$*:* $|Pr[\mathscr{A} \rightarrow 1 | P = P_1] - Pr[\mathscr{A} \rightarrow 1 | P = P_0]| \leq 2^{-\lambda}$.

Proof: We refer to Algesheimer et al. [7] Section 3.2 and Schoenmakers and Tuyls [211], Appendix A, for the proof of the statistical λ-indistinguishability.

We recall that an encoded message msg of $N/2$ complex numbers with the CKKS scheme is an integer polynomial of $\mathbb{Z}[X]/(X^N+1)$. Given that $||msg|| < 2^{\delta}$, and a second polynomial M of N integer coefficients with each coefficient uniformly sampled and bounded by $2^{\lambda+\delta} - 1$ for a security parameter λ, Lemma C.2.1 suggests that $\Pr[||msg^{(i)} + M^{(i)}|| \geq 2^{\lambda+\delta}] \leq 2^{-\lambda}$, for $0 \leq i < N$ and where i denotes the i^{th} coefficient of the polynomial. That is, the probability of a coefficient of $msg + M$ to be distinguished from a uniformly sampled integer in $[0, 2^{\lambda+\delta})$ is bounded by $2^{-\lambda}$. In Mouchet et al. protocol, each party samples its polynomial mask M with uniform coefficients in $[0, 2^{\lambda+\delta})$.

The parties, however, should have an estimate of the magnitude of msg to derive δ, which can be derived from the plaintext scale, integer precision and previous homomorphic operations. The masks M_i are added to the ciphertext at its current level τ in the domain R_{Q_τ} during the switch to the secret-shared domain. To avoid a modular reduction of the masks in R_{Q_τ} and ensure a correct re-encryption at the maximum level L, i.e., in domain R_{Q_L}, the modulus Q_τ should be large enough for the additions of N masks.

C.3 Evaluation Complement

We first provide the complete complexity analysis of SPINDLE. We describe in more details the datasets used to assess SPINDLE's training accuracy and show in Table C.1b an extended version of Table 5.2 including the learning and approximation parameters.

Theoretical Analysis

In Table C.1a, we provide the complete complexity analysis of SPINDLE. $|ct|$ represents the maximum size of a ciphertext, i.e., the size of a fresh ciphertext at level L, E and D stand for encryption and decryption workloads. The DA packing approach incurs a higher computation complexity (with notably more plaintext-ciphertext multiplications and rescaling (M), and rotations (R)) but, as shown in Section 5.7.2, it is embarrassingly parallel, i.e., operations can be amortized by a factor $N_1 \cdot N_2$ (defined in Section 5.4.2) depending on the available threads.

Communication

With the security parameters SP1 (see Section 5.7.2), the size of a ciphertext is 2.6MB and each DP receives and sends one ciphertext per global iteration. One ciphertext is also exchanged for each DBootstrap(·) (e.g., every two global iterations).

	Comm. (tot)	Comput. (per DP_i)
MAP (RBA)	$\tilde{B_M}$	$g\,\mathsf{z}\,(4M + 2(log_2(b) + log_2(c))R + \sigma)\#ct + \tilde{B_M}$
MAP (DA)	$\tilde{B_M}$	$(2(N_1 N_2 M + (N_1 + N_2 - 2)R) + \sigma)g\,\mathsf{z} + \tilde{B_M}$
COMBINE	$g(\|S\| - 1)\|ct\|$	$3A$
REDUCE	$2g(\|S\| - 1)\|ct\| + \tilde{B_R}$	$M + \tilde{B_R}$
PRED	$2(\|ct\| + (\|S\| - 1)\|ct\|)$	$M' + \sigma + D$

DBoot.	$\tilde{B_R} = 2\|ct\|(\|S\| - 1)f(g)$,
	$\tilde{B_M} = 2\|ct\|\|S\|(\|S\| - 1)f(g, \mathsf{z})$
	if LB: $\tilde{B_R} = 0$, **if GB:** $\tilde{B_M} = 0$,
	if HB: $f(g) \rightarrow f(g, \mathsf{z})$
		$\tilde{B_R} = (D + E)f(g))$
		$\tilde{B_M} = \|S\|(D + E)f(g, \mathsf{z})$
		if LB: $\tilde{B_R} = 0$, **if GB:** $\tilde{B_M} = 0$,
		if HB: $f(g) \rightarrow f(g, \mathsf{z})$

$\boldsymbol{\sigma}$		$(log_2(d) + 1)(M + M' + A)$

(a) Theoretical Analysis. The list of symbols is given in Table 1.3. The number of bootstrap operations in the hybrid bootstrap approach (HB) always depends on both the number of global *g* and local *z* iterations. *M* is the plaintext-ciphertext multiplication and rescaling complexity and *M′* is the ciphertext-ciphertext multiplication complexity. *b* and *c* are the batch size and number of features and *R* is the rotation computation complexity. N_1 and N_2 are defined in Section 5.4.2, *E* and *D* stand for encryption and decryption workloads.

Dataset	**Vers.**	**SP**, α, ρ, **b**	**g, m**	$\{[\mathbf{a_i}, \mathbf{g_i}], \mathbf{d_i}\}$	**Acc./MSE**	**F1/MAE**	**T.**	**P.**
CalCOFI [812,174x2]	CCS, [IT]	$-, 10^{-1}, -, 1300$	–	–	15.157, [408]	3.1, [19.67]	–	–
	DNP	$-, 10^{-1}, 10^{-2}, 1300$	50, 1	–	17.679	3.45	6.71	$2 \cdot 10^{-4}$
	SPINDLE	$2, 10^{-1}, 10^{-2}, 1300$	50, 1	–	17.938	3.62	65.31	0.23
PIMA [768x8]	CCS, [IT]	$-, 10^{-2}, -, 50$	–	–	0.784, [0.720]	0.680, [0.604]	–	–
	DNP	$-, 10^{-2}, 10^{-2}, 50$	1, 30	–	0.781	0.679	0.038	$9 \cdot 10^{-5}$
	SPINDLE	$2, 10^{-2}, 10^{-2}, 50$	1, 30	[±7], 3	0.780	0.677	11.28	0.18
BCW [699x9]	CCS, [IT]	$-, 10^{-1}, -, 20$	–	–	0.962, [0.922]	0.947, [0.877]	–	–
	DNP	$-, 10^{-1}, 10^{-1}, 20$	1, 3	–	0.962	0.942	0.034	$5 \cdot 10^{-5}$
	SPINDLE	$2, 10^{-1}, 10^{-1}, 20$	1, 3	[±1], 3	0.962	0.944	3.25	0.16
ESR [11,500x90]	CCS, [IT]	$-, 6^{-3}, -, 10$	–	–	0.842, [0.838]	0.462, [0.396]	–	–
	DNP	$-, 6^{-3}, 10^{-1}, 10$	92, 1	–	0.840	0.460	2.89	$8 \cdot 10^{-5}$
	SPINDLE	$2, 6^{-3}, 10^{-1}, 10$	92, 1	[±15], 5	0.839	0.456	53.27	0.35
MNIST [70,000 x 784] (multi.)	CCS, [IT]	$-, 10^{-4}, -, 1024$	–	–	0.873, [0.873]	0.871, [0.832]	–	–
	DNP	$-, 10^{-4}, 10^{-1}, 1024$	3, 6	–	0.865	0.863	43.95	0.49
	SPINDLE	$1, 10^{-4}, 10^{-1}, 1024$	3, 6	SM:[−30, 4], 15 [1, 40], 19 M:[±15], 15 [1, 40], 15	0.8617	0.86	558	4.33
MNIST [70,000 x 784] (1 vs. a)	CCS, [IT]	$-, 10^{-4}, -, 1024$	–	–	0.856, [0.827]	0.859, [0.822]	–	–
	DNP	$-, 10^{-4}, 10^{-1}, 1024$	3, 6	–	0.853	0.858	43.98	0.49
	SPINDLE	$1, 10^{-4}, 10^{-1}, 1024$	3, 6	[±15], 15	0.852	0.850	187.8	4.33

(b) Baseline Comparison with K-fold=5. CCS is a non-private centralized solution, DNP is a distributed non-privacy preserving approach and IT means independent training, i.e., a DP using only its local subset of the entire dataset. Time to train (T.) and predict (P.) is in seconds.

Table C.1 – SPINDLE Evaluation.

Evaluation Datasets

For linear regression, we use the CalCOFI dataset (with $n = 812,174$ records and $c = 2$ features) [36]. It contains oceanographic data (e.g., salinity) that can be used to predict the water temperature. For logistic regression, we use three different datasets: (a) the Breast Cancer

Wisconsin dataset (BCW, $n = 699$, $c = 9$) [19] contains patients' data that is employed to predict the presence of a breast cancer, (b) the PIMA dataset ($n = 768$, $c = 8$) [192] contains medical observations collected from an Indian community that can be used to predict the presence of diabetes, and (c) the Epileptic Seizure Recognition dataset (ESR, $n = 11{,}500$, $c = 179$) [77] contains patients' data that can be used to predict a seizure. For multinomial regression, we test SPINDLE on the MNIST dataset ($n = 70{,}000$, $c = 784$) [134], where the goal is to identify single-digits out of grey-scale images. We rely on these datasets to compare SPINDLE with various baselines.

Comparison with Prior Art

In Table C.2, we perform a qualitative and quantitative comparison of SPINDLE with existing works. We consider a generic privacy-preserving centralized encrypted solution (CES) and distributed solutions: Prio [52], which relies on secret sharing, and Helen [263], a solution that employs a different distributed approach, the Alternating Direction Method of Multipliers (ADMM) proposed by Boyd et al. [35], to train regularized linear models. CES represents a centralized solution, similar to existing works [39; 124; 126], in which one DP outsources its encrypted data to a server that trains and evaluates a model. For a fair comparison, we estimate the execution time (without communication) of a generic centralized (outsourced) solution (CES) relying on the non-multiparty CKKS scheme with security parameters that enable packing of the same number of values in one ciphertext. We use as a reference one of the most recent works on bootstrapping by Han and Dohyeong [106]. In Helen, the DPs perform the ADMM optimization locally under a quantum-vulnerable additive HE cryptoscheme, and combine their results under secret-sharing. ADMM is less widespread than SGD, it is primarily designed for linear models and does not provide the same stability and convergence guarantees than the cooperative gradient-descent [33; 241; 242; 260], for which convergence can be derived from

	Confident.	Non-lin. Mod.	Scal. w. $\lvert S\rvert, c, n_i$	Acc. w.r.t central.	$\lvert S\rvert, c, n_i$	**Lin.**	**Log.**	**Multi.**
CES	data+model	✔	- lin. lin.	$\approx$	$25.6k$, 2^{12}, -	$12k$	$14k$	-
Prio	partially for data	✔	lin. quadra. indep.	$<$	$25.6k$, 2^{12}, 5	inf	inf	-
Helen	data	✗	quadra. quadra. indep.	$\approx$	$400k, 100, 4$ $400k, 10, 10$ $4M, 90, 4$	$9k$ $1.7k$ $6.5k$	-	-
SPINDLE	data+model	✔	indep. log. lin.	$\approx$	$25.6k, 2^{12}, 5$ $400k, 100, 4$ $400k, 10, 10$ $4M, 90, 4$	480 528 513 $5.8k$	658 – – –	$33.6k$ – – –

Table C.2 – Comparison with existing solutions on whether the solutions (i) support the training of non-linear models, (ii) scale in multiple dimensions, and (iii) achieve similar accuracy as a centralized non-secure approach. $|S|$, c are the number of DPs and features, n_i the size of each DP_i local dataset. Timings for linear (lin.), logistic (log.), multinomial (multi.) regressions training are in seconds.

SGD. Since Helen's implementation is not available, we aim at providing an intuition on how it quantitatively compares with SPINDLE. To this end, we used results reported in Helen [263] and performed similar experiments in SPINDLE ((2),(3),(4) in Table C.2). We highlight here that the experiment environment is different, and these results provide only an idea of how these systems compare. For a fair comparison, we excluded the proof generation time in Helen and we notice that as SPINDLE, Helen reported similar accuracy results as a non-secure centralized solution. We observe that SPINDLE scales better than Helen when increasing the number of data providers $|S|$, as its execution time is almost the same in ((2) and (3)), and it also scales better with the number of features (almost 10x better in (4)).

C.4 Modular Extensions

We discuss here a set of extensions that can be (optionally) integrated and combined in SPINDLE depending on the application. We describe how SPINDLE can be extended to withstand malicious adversaries and support more complex ML models. We also discuss extensions to SPINDLE that can be employed to support dynamic DPs, optimize the model training and enable quality control.

C.4.1 Malicious Adversaries

Malicious DPs Interfering with TRAINING. To limit the extent to which a malicious DP could interfere with its TRAINING, SPINDLE can require from the DPs to publish transcripts of their computations [88] and to produce proofs of correct inputs. These features combined would enable SPINDLE to be fully auditable. Mechanisms to avoid model poisoning attacks when the input data are encrypted (and have to remain confidential) are an open research problem. However, SPINDLE can partially mitigate this threat by, similarly to what we present in Chapter 4, constraining the DPs' inputs and requiring zero knowledge proofs of range [16; 17; 138; 254] from the DPs. This would substantially limit the extent to which a malicious DP could interfere with SPINDLE's TRAINING. However, we note that this does not thwart all possible attacks, as, for example, poisoning attacks would still be possible with in-range input data.

Malicious DPs Interfering with PREDICTION. As for the TRAINING, computation correctness can be verified through computations' transcripts published by the DPs. To prevent a malicious DP from learning a victim querier's prediction outputs via a replay attack (i.e., reusing the querier's encrypted data in a new query), SPINDLE can require queriers to provide signed proofs of knowledge of the input data [149].

Malicious Querier Inferring Information from PREDICTION's Output. SPINDLE naturally covers federated learning attacks [112; 157; 169] and model inversion attacks [83], as the intermediate and final weights are never revealed. Moreover, SPINDLE can also mitigate inference attacks, e.g., membership inference [221], by limiting the number of prediction requests on the trained model. This solution can be improved by adding noise to the prediction output to achieve differential privacy guarantees. In fact, a mechanism that ensures differential privacy can be used for all the outputs of SPINDLE: on the predictions $\boldsymbol{y}'$ and on the trained model, if it is released after training (Section 5.5). This would ensure that a passive adversary (e.g., trying to infer information from the system's outputs) or an active adversary controlling a subset of the DPs cannot learn information, e.g., data or local model of honest parties, about a subset of the DPs. To ensure differential privacy, SPINDLE should add some collectively generated noise [125], similarly to the solution introduced in Chapter 3 to the query result before performing DKeySwitch(·). However, the choice of the privacy parameters is not trivial and is an interesting direction for future work. Furthermore, the use of differential privacy in dynamic systems presents serious limitations; minimizing the released non-encrypted information (which also reduces the noise magnitude required to meet a target differential privacy level) is much more effective and practical. This is the approach taken in SPINDLE, contrarily to federated learning systems, where the intermediate outputs of each training iteration are always disclosed.

C.4.2 More Complex ML Models.

We first remark that the extended, privacy-preserving MapReduce abstraction on which we rely to build SPINDLE can actually capture many of existing solutions for secure distributed ML training [26; 28; 49; 52; 95; 163; 178; 220; 263], including the training functionality of DRYNX introduced in Chapter 4. We also remark that, even though we rely on the widely applicable distributed stochastic gradient descent (SGD), other distributed approaches for training ML models such as ADMM [35] could also be expressed in the same abstraction. However, by relying on SGD, we aim at designing a system that can then be extended to other models, as SGD can be used to minimize many cost functions [131; 233; 261]. In particular, it can be extended to more complex models such as neural networks, which are usually trained using SGD [65]. SPINDLE supports any activation function that can be "practically" approximated by a polynomial; hence, the challenges for its extension to more complex models reside in trading-off precision for efficiency when approximating non-polynomial functions, and efficiently packing the data depending on the operations. This is particularly important for neural networks in which the computations are sequentially performed through multiple layers. Thus, each SGD iteration would involve higher multiplicative-depth circuits and their evaluation under encryption. We made a first step towards the privacy-preserving federated training of neural networks in our subsequent work [208], which is based on the framework introduced in this chapter.

Threshold-encryption Scheme. To account for unresponsive DPs, SPINDLE can use a threshold-encryption scheme, where the DPs secret-share [217] their secret keys, thus enabling a subset of the DPs to perform the cryptographic interactive protocols (DBootstrap(·) or DKeySwitch(·)).

Dynamic Roles. The role of DP_R played by one DP has no security implications and only incurs small computation overhead for one DP. This role can be dynamically assigned (e.g., round robin) at each global iteration or whenever the DP playing DP_R becomes unavailable.

Asynchronous Learning & Performance Optimizations. We experimentally observed that an uneven distribution of the data across DPs does not affect the training accuracy. However, and as expected, in order to obtain similar accuracy as a centrally trained model, the labels of the DPs' local datasets should be similarly distributed. For this, SPINDLE can integrate optimizations of the stochastic gradient descent (SGD) that can be expressed as a polynomial; in particular, SGD asynchronous variants that account for imbalances in DPs' response times or data distribution, or for sparse networks adaptations (e.g., Koloskova et al. [129]).

To avoid over- or under-fitting, which often happens when the number of training iterations is predefined, SPINDLE can integrate a collective stop-test protocol. This protocol enables the DPs to collectively decrypt the absolute difference between the (global) weights of two subsequent (global) iterations, or a statistic derived from these values. The decrypted value is compared to a chosen threshold to stop the training.

Data Preparation & Quality Control. As mentioned before, the training on a distributed dataset can be optimized according to how the data are distributed among the DPs. This information can also serve for data standardization and quality control, and its leakage can be mitigated by relying on differential privacy or on HE-based interactive protocols [88]. SPINDLE's PREPARE phase can be extended to include these solutions and it is up to the DPs to choose the configuration that achieves the required balance between privacy and performance.

D Federated Analytics for Precision Medicine

In this appendix, we show how the values are packed in ciphertexts, in order to efficiently perform the GWAS.

D.1 Packing for the Secure & Federated Computation of a GWAS

In Figure D.1, we describe how the values are packed in order to perform the computations of the main variables used in the protocols 6.1 and 6.2 (Section 6.2.3. We note that 13 covariates are considered in the study that we replicate [151].

We provide here the main intuition behind the packing of each element (in Figure D.1) and its high level cost in terms of operations:

1. The inverse computation (line 4 in Protocol 6.1 & 6.2) requires row operations on the input matrix. The matrix is therefore row-wise encrypted and its rows are duplicated to enable an efficient use of SIMD and to perform the subsequent operations for multiple variants simultaneously.
2. The vectors multiplication results are packed such that they are in the right positions to perform the subtraction in line 11 for Protocol 6.1 and line 14 for Protocol 6.2. This packing can be performed by the data providers on their cleartext data and does not require any rotation of encrypted ciphertexts.
3. Each element of the plaintext $u^T X$ is replicated to be multiplied with a row of $(X^T X)^{(-1)}$ and the 13 multiplication results are aggregated. This operation does not require any rotation either.
4. The dot product between the plaintext vector $X^T u$ and each encrypted row of W_{21} is performed and the result is packed such that it can be used as such in the subsequent subtraction (line 11 for Protocol 6.1 and line 14 for Protocol 6.2). The dot product requires $log_2(vectorsize)$ rotations. We note here that all vectors are padded with zeros

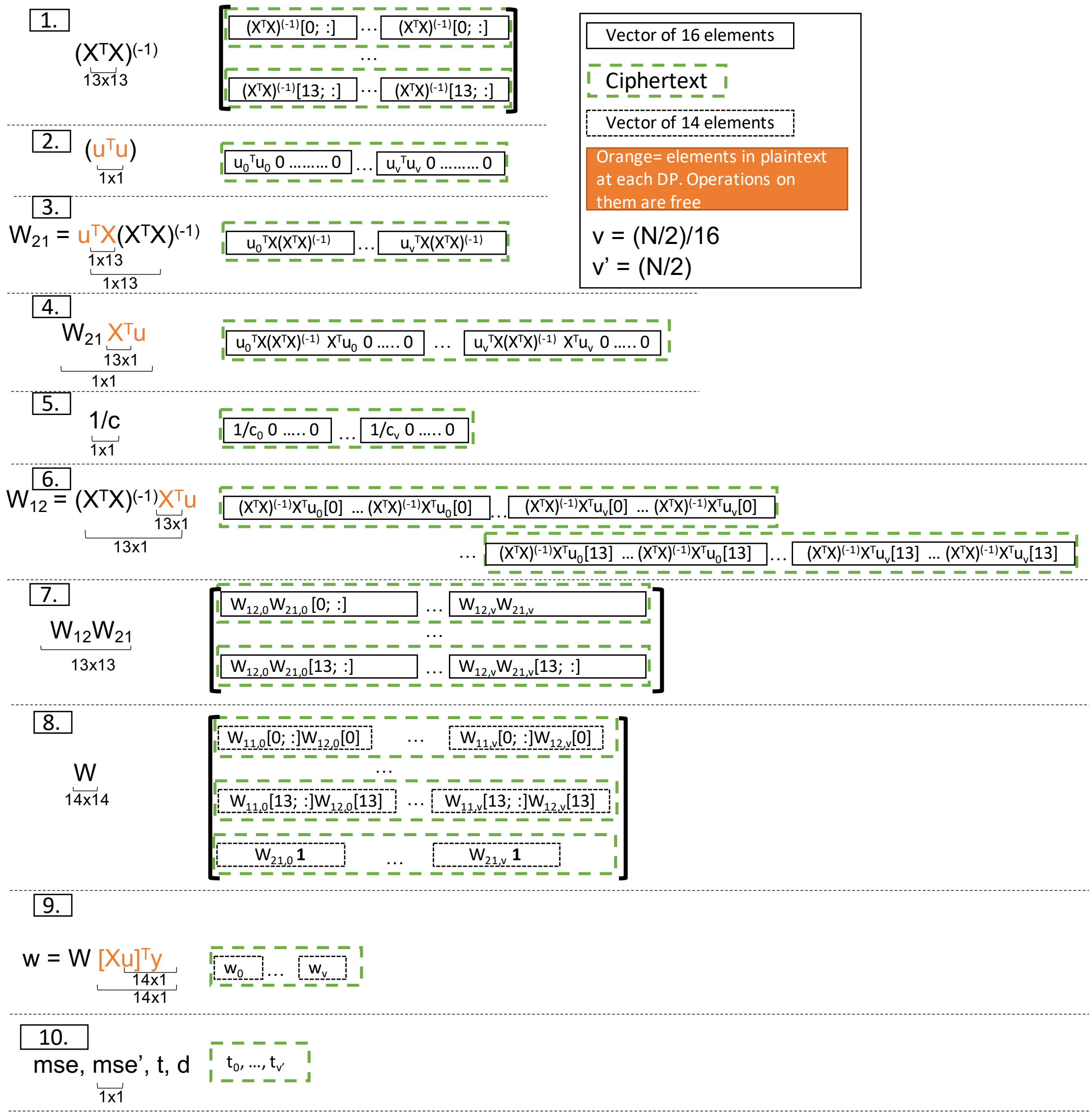

1.
(XTX)(-1)
13x13
(XTX)(-1)[0; :] … (XTX)(-1)[0; :]
…
(XTX)(-1)[13; :] … (XTX)(-1)[13; :]
Vector of 16 elements
Ciphertext
Vector of 14 elements
Orange= elements in plaintext at each DP. Operations on them are free
v = (N/2)/16
v' = (N/2)
2.
(uTu)
1x1
u0Tu0 0 .... 0 … uvTuv 0 0
3.
W21 = uTX(XTX)(-1)
1x13
1x13
u0TX(XTX)(-1) … uvTX(XTX)(-1)
4.
W21 XTu
13x1
1x1
u0TX(XTX)(-1) XTu0 0 ..... 0 … uvTX(XTX)(-1) XTuv 0 0
5.
1/c
1x1
1/c0 0 ..... 0 … 1/cv 0 0
6.
W12 = (XTX)(-1)XTu
13x1
13x1
(XTX)(-1)XTu0[0] ... (XTX)(-1)XTu0[0] … (XTX)(-1)XTuv[0] ... (XTX)(-1)XTuv[0]
… (XTX)(-1)XTu0[13] ... (XTX)(-1)XTu0[13] … (XTX)(-1)XTuv[13] ... (XTX)(-1)XTuv[13]
7.
W12W21
13x13
W12,0W21,0 [0; :] … W12,vW21,v
…
W12,0W21,0[13; :] … W12,vW21,v[13; :]
8.
W
14x14
W11,0[0; :]W12,0[0] … W11,v[0; :]W12,v[0]
…
W11,0[13; :]W12,0[13] … W11,v[13; :]W12,v[13]
W21,0 1 … W21,v 1
9.
w = W [Xu]Ty
14x1
14x1
w0 … wv
10.
mse, mse', t, d
1x1
t0, …, tv'

Figure D.1 – Optimized Packing for a Simultaneous Computation on v Variants.

such that their size is a power of 2. This is done to minimize the number of rotations required.

5. As before, the scalar values for v variants are packed to simplify the subsequent subtraction.

6. As in 4, the dot product is performed between the plaintext vector $X^T u$ and each encrypted row of the matrix $(X^T X)^{(-1)}$. The results are then duplicated to prepare the multiplication of line 14 in Protocol 6.1. This operation requires $2 \times log_2(vectorsize)$ rotations.

7. Each of the 13 ciphertexts of W_{12} is multiplied with W_{21}. This operation does not

require any rotation as the values have been prepared (packed) for this multiplication beforehand (in 6).

8. To construct the matrix W for v variants, we include W_{12} directly in the padding of W_{11}. Due to W_{12} packing, this is done in one mask (multiplication with binary vector) and one addition and does not require any rotation. Similarly, one addition is performed to include the 1 values in W_{21}.
9. The dot product between each encrypted row of W and $[Xu]^T y$ is performed such that the result can be packed to prepare the multiplication of line 17 in Protocol 6.1. This requires $log_2(vector/rowsize) + (vector/row - 1)$ rotations.
10. All elements are computed for v variants simultaneously.

D.2 Data Providers Independent Results

Figure D.2 complements Figure 6.4c by showing the independent GWAS result obtained by each of the 12 data providers on its own subset of the data.

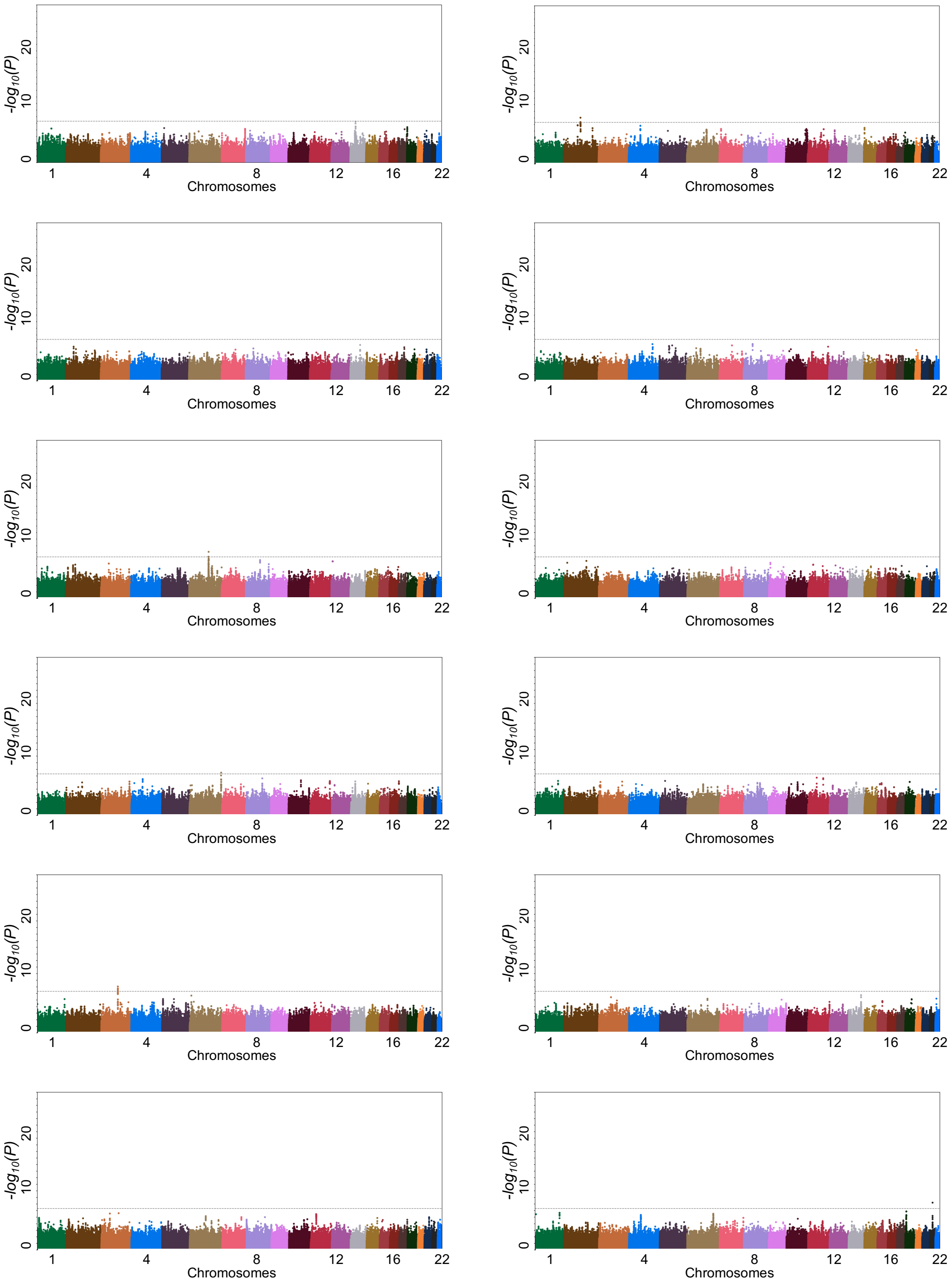


Figure D.2 – Independent GWAS result obtained by each data provider using only its own subset of the data.

Bibliography

[1] A New Data Breach May Have Exposed ... Every American Adult. https://tinyurl.com/ydz7jpdk, (accessed: 4.02.2019).

[2] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, et al. Tensorflow: A System for Large-Scale Machine Learning. In *12th USENIX Symposium on Operating Systems Design and Implementation, OSDI 16*, pages 265–283, 2016.

[3] M. Abadi, A. Chu, I. Goodfellow, H. B. McMahan, I. Mironov, K. Talwar, and L. Zhang. Deep Learning with Differential Privacy. In *ACM Conference on Computer and Communications Security (CCS)*, 2016.

[4] A. Akavia, H. Shaul, M. Weiss, and Z. Yakhini. Linear-Regression on Packed Encrypted Data in the Two-Server Model. In *ACM WAHC*, 2019.

[5] M. R. Albrecht, M. Chase, H. Chen, J. Ding, S. Goldwasser, S. Gorbunov, S. Halevi, J. Hoffstein, K. Laine, K. E. Lauter, et al. Homomorphic Encryption Security Standard. Technical report, HomomorphicEncryption.org, 2018.

[6] M. R. Albrecht, R. Player, and S. Scott. On the Concrete Hardness of Learning with Errors. *J. of Mathematical Cryptology*, 2015.

[7] S. V. Algesheimer J., Camenisch J. Efficient Computation Modulo a Shared Secret with Application to the Generation of Shared Safe-Prime Products. In *CRYPTO*, 2002.

[8] All of Us Research Program, NIH. https://allofus.nih.gov/, (accessed: 30.01.2021).

[9] B. Anandan and C. Clifton. Laplace Noise Generation for Two-party Computational Differential Privacy. In *13th Annual Conference on Privacy, Security and Trust (PST)*, pages 54–61, 2015.

[10] Y. Aono, T. Hayashi, L. Trieu Phong, and L. Wang. Scalable and Secure Logistic Regression via Homomorphic Encryption. In *Proceedings of the Sixth ACM Conference on Data and Application Security and Privacy*. ACM, 2016.

[11] Apple Watch Heart Monitoring. https://tinyurl.com/y7ctnauc, (accessed: 29.01.2019).

[12] K. E. Atkinson. *An Introduction to Numerical Analysis.* John wiley & sons, 2008.

[13] D. B. Baker, J. Kaye, and S. F. Terry. Privacy, Fairness, and Respect for Individuals. *eGEMS (Generating Evidence & Methods to Improve Patient Outcomes)*, 4(2), 2016.

[14] P. S. Barreto and M. Naehrig. Pairing-Friendly Elliptic Curves of Prime Order. In *International Workshop on Selected Areas in Cryptography*. Springer, 2005.

[15] J. Bater, G. Elliott, C. Eggen, S. Goel, A. Kho, and J. Rogers. SMCQL: Secure Querying for Federated Databases. *VLDB*, 10(6):673–684, 2017.

[16] C. Baum, I. Damgård, S. Oechsner, and C. Peikert. Efficient Commitments and Zero-Knowledge Protocols from Ring-SIS with Applications to Lattice-based Threshold Cryptosystems. *IACR Cryptology ePrint Archive*, 2016:997, 2016.

[17] C. Baum and A. Nof. Concretely-efficient Zero-knowledge Arguments for Arithmetic Circuits and their Application to Lattice-based Cryptography. In *PKC*, 2020.

[18] S. Bayer and J. Groth. Efficient Zero-knowledge Argument for Correctness of a Shuffle. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 263–280. Springer, 2012.

[19] Breast Cancer Wisconsin (Original). https://archive.ics.uci.edu/ml/datasets/breast+cancer+wisconsin+(original), (accessed: 14.02.2020).

[20] A. L. Beam and I. S. Kohane. Big Data and Machine Learning in Health Care. *Jama*, 2018.

[21] M. Bellare, V. T. Hoang, S. Keelveedhi, and P. Rogaway. Efficient Garbling from a Fixed-Key Blockcipher. In *IEEE Symposium on Security and Privacy (SP)*, pages 478–492, May 2013.

[22] D. J. Bernstein, N. Duif, T. Lange, P. Schwabe, and B.-Y. Yang. High-speed High-security Signatures. *Journal of Cryptographic Engineering 2*, pages 77–89, 2012.

[23] V. Bindschaedler, R. Shokri, and C. A. Gunter. Plausible Deniability for Privacy-preserving Data Synthesis. *VLDB*, 10(5), 2017.

[24] F. Boemer, A. Costache, R. Cammarota, and C. Wierzynski. nGraph-HE2: A High-Throughput Framework for Neural Network Inference on Encrypted Data. In *ACM WAHC*, 2019.

[25] D. Bogdanov, L. Kamm, B. Kubo, R. Rebane, V. Sokk, and R. Talviste. Students and Taxes: a Privacy-Preserving Study Using Secure Computation. In *Proceedings on Privacy Enhancing Technologies Symposium*, 2016.

[26] D. Bogdanov, L. Kamm, S. Laur, and V. Sokk. Rmind: a Tool for Cryptographically Secure Statistical Analysis. *IEEE TDSC*, 2016.

[27] D. Bogdanov, S. Laur, and J. Willemson. Sharemind: A Framework for Fast Privacy-preserving Computations. In *European Symposium on Research in Computer Security*, pages 192–206. Springer, 2008.

[28] K. Bonawitz, H. Eichner, W. Grieskamp, D. Huba, A. Ingerman, V. Ivanov, C. Kiddon, J. Konečnỳ, S. Mazzocchi, H. B. McMahan, et al. Towards Federated Learning at Scale: System Design. In *SysML*, 2019.

[29] L. Bonomi, X. Jiang, and L. Ohno-Machado. Protecting Patient Privacy in Survival Analyses. *Journal of the American Medical Informatics Association*, 27(3):366–375, 2020.

[30] C. Bonte and F. Vercauteren. Privacy-preserving Logistic Regression Training. *BMC medical genomics*, 2018.

[31] J. W. Bos, K. Lauter, J. Loftus, and M. Naehrig. Improved Security for a Ring-based Fully Homomorphic Encryption Scheme. In *IMACC*, 2013.

[32] R. Bost, R. A. Popa, S. Tu, and S. Goldwasser. Machine Learning Classification over Encrypted Data. In *The Network and Distributed System Security Symposium (NDSS)*, 2015.

[33] L. Bottou, F. E. Curtis, and J. Nocedal. Optimization Methods for Large-Scale Machine Learning. *Siam Review*, 2018.

[34] C. Boura, I. Chillotti, N. Gama, D. Jetchev, S. Peceny, and A. Petric. High-Precision Privacy-Preserving Real-Valued Function Evaluation. *FC '18*, 2018.

[35] S. Boyd, N. Parikh, E. Chu, B. Peleato, J. Eckstein, et al. Distributed Optimization and Statistical Learning via the Alternating Direction Method of Multipliers. *Foundations and Trends in Machine learning*, 2011.

[36] CalCOFI, over 60 Years of Oceanographic Data. https://www.kaggle.com/sohier/calcofi, (accessed: 05.03.2020).

[37] J. Camenisch, R. Chaabouni, and a. Shelat. Efficient Protocols for Set Membership and Range Proofs. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 234–252. Springer, 2008.

[38] J. Camenisch and M. Stadler. Proof Systems for General Statements about Discrete Logarithms. *Technical Report*, 1997.

[39] S. Carpov, N. Gama, M. Georgieva, and J. R. Troncoso-Pastoriza. Privacy-preserving Semi-Parallel Logistic Regression Training with Fully Homomorphic Encryption. *IACR Cryptology ePrint Archive*, 2019.

[40] M. Castro, B. Liskov, et al. Practical Byzantine Fault Tolerance. In *OSDI*, volume 99, pages 173–186, 1999.

[41] T. Caulfield and B. Murdoch. Genes, Cells, and Biobanks: Yes, There's Still a Consent Problem. *PLoS biology*, 15(7):e2002654, 2017.

[42] K. Chaudhuri and C. Monteleoni. Privacy-preserving Logistic Regression. In *Advances in neural information processing systems (NIPS)*, 2009.

[43] H. Chen, R. Gilad-Bachrach, K. Han, Z. Huang, A. Jalali, K. Laine, and K. Lauter. Logistic Regression over Encrypted Data from Fully Homomorphic Encryption. *BMC medical genomics*, 2018.

[44] R. Chen, A. Reznichenko, P. Francis, and J. Gehrke. Towards Statistical Queries over Distributed Private User Data. In *NSDI*, volume 12, pages 13–13, 2012.

[45] J. H. Cheon, M. Hhan, S. Hong, and Y. Son. A Hybrid of Dual and Meet-in-the-Middle Attack on Sparse and Ternary Secret LWE. *IEEE Access*, 2019.

[46] J. H. Cheon, S. Hong, and D. Kim. Remark on the security of ckks scheme in practice. *IACR Cryptol. ePrint Arch.*, 2020:1581, 2020.

[47] J. H. Cheon, A. Kim, M. Kim, and Y. Song. Homomorphic Encryption for Arithmetic of Approximate Numbers. In *ASIACRYPT*, 2017.

[48] H. Cho, S. Simmons, R. Kim, and B. Berger. Privacy-Preserving Biomedical Database Queries with Optimal Privacy-Utility Trade-offs. *Cell systems*, 10(5):408–416, 2020.

[49] H. Cho, D. J. Wu, and B. Berger. Secure Genome-Wide Association Analysis using Multiparty Computation. *Nature biotechnology*, 36(6):547–551, 2018.

[50] B. Chor, S. Goldwasser, S. Micali, and B. Awerbuch. Verifiable Secret Sharing and Achieving Simultaneity in the Presence of Faults. In *26th Annual Symposium on Foundations of Computer Science*, pages 383–395. IEEE, 1985.

[51] C.-T. Chu et al. Map-reduce for Machine Learning on Multicore. In *NIPS*, 2007.

[52] H. Corrigan-Gibbs and D. Boneh. Prio: Private, Robust, and Computation of Aggregate Statistics. In *NSDI*, pages 259–282, 2017.

[53] V. Costan and S. Devadas. Intel SGX Explained. *IACR Cryptol. ePrint Arch.*, 2016(86):1–118, 2016.

[54] Why Is It So Hard to Review the Johnson & Johnson Vaccine? Data. https://tinyurl.com/485wx7rp, 15.04.2021, 2021.

[55] J. L. Crawford, C. Gentry, S. Halevi, D. Platt, and V. Shoup. Doing Real Work with FHE: The Case of Logistic Regression. In *ACM WAHC*, 2018.

[56] I. Damgård, V. Pastro, N. Smart, and S. Zakarias. Multiparty Computation from Somewhat Homomorphic Encryption. In *CRYPTO*, 2012.

[57] G. Danezis and S. Meiklejohn. Centrally Banked Cryptocurrencies. *Proceedings of the 24rd Network and Distributed System Security Symposium*, 2016.

[58] L. de Castro, C. Juvekar, and V. Vaikuntanathan. Fast Vector Oblivious Linear Evaluation from Ring Learning with Errors. *IACR Cryptol. ePrint Arch.*, 2020:685, 2020.

[59] T. de Souza, J. Wright, P. O'Hanlon, and I. Brown. Set Difference Attacks in Wireless Sensor Networks. *International Conference on Security and Privacy in Communication Systems*, 2012.

[60] J. Dean and S. Ghemawat. MapReduce: Simplified Data Processing on Large Clusters. *Communications of the ACM*, 2008.

[61] DeDiS Research Lab at EPFL, advanced crypto library for the Go language. https://github.com/DeDiS/crypto, (accessed: 22.04.2021).

[62] X. Dong, J. Yu, Y. Luo, Y. Chen, G. Xue, and M. Li. Achieving an Effective, Scalable and Privacy-preserving Data Sharing Service in Cloud Computing. *Computers & security*, 42:151–164, 2014.

[63] Data Protection in Personalized Health. https://dpph.ch/, (accessed: 21.04.2021).

[64] M. Du, Q. Wang, M. He, and J. Weng. Privacy-Preserving Indexing and Query Processing for Secure Dynamic Cloud Storage. *IEEE Transactions on Information Forensics and Security*, 13(9):2320–2332, 2018.

[65] S. S. Du, J. D. Lee, H. Li, L. Wang, and X. Zhai. Gradient Descent Finds Global Minima of Deep Neural Networks. *arXiv preprint arXiv*, abs/1811.03804, 2018.

[66] S. S. Du, X. Zhai, B. Poczos, and A. Singh. Gradient Descent Provably Optimizes Over-parameterized Neural Networks. *arXiv preprint arXiv:1810.02054*, 2018.

[67] Y. Duan, J. Canny, and J. Zhan. Efficient Privacy-preserving Association Rule Mining: P4P Style. In *IEEE Symposium on Computational Intelligence and Data Mining*, pages 654–660. IEEE, 2007.

[68] C. Dwork. Differential Privacy. In *https://www.microsoft.com/en-us/research/publication/differential-privacy/*, Venice, Italy, July 2006. Springer Verlag.

[69] C. Dwork. A Firm Foundation for Private Data Analysis. In *Communications of the ACM, 54(1)*, pages 86–95, 2011.

[70] C. Dwork, K. Kenthapadi, F. McSherry, I. Mironov, and M. Naor. Our Data, Ourselves: Privacy via Distributed Noise Generation. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 486–503. Springer Berlin Heidelberg, 2006.

[71] C. Dwork, F. McSherry, K. Nissim, and A. Smith. Calibrating Noise to Sensitivity in Private Data Analysis. In *Theory of Cryptography Conference*, pages 265–284. Springer, 2006.

[72] C. Dwork, A. Roth, et al. The Algorithmic Foundations of Differential Privacy. *Foundations and Trends® in Theoretical Computer Science*, 9(3–4):211–407, 2014.

[73] Dyadic Security. https://www.dyadicsec.com/, (accessed: 10.09.2020).

[74] T. ElGamal. A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms. *IEEE transactions on information theory*, 31(4):469–472, 1985.

[75] Equifax Breach. https://tinyurl.com/y9h4pgsk, (accessed: 4.02.2019).

[76] B. J. Erickson, P. Korfiatis, Z. Akkus, and T. L. Kline. Machine Learning for Medical Imaging. *Radiographics*, 2017.

[77] Epileptic Seizure Recognition Dataset. https://archive.ics.uci.edu/ml/datasets/Epileptic+Seizure+Recognition, (accessed: 14.02.2020).

[78] B. Fabian, T. Ermakova, and P. Junghanns. Collaborative and Secure Sharing of Healthcare Data in Multi-clouds. *Information Systems*, 48:132–150, Mar. 2015.

[79] J. Fan and F. Vercauteren. Somewhat Practical Fully Homomorphic Encryption. *IACR Cryptology ePrint Archive*, 2012:144, 2012.

[80] A. Fiat and A. Shamir. How to Prove Yourself: Practical Solutions to Identification and Signature Problems. In *Conference on the Theory and Application of Cryptographic Techniques*, pages 186–194. Springer, 1986.

[81] B. Ford, J.-P. Hubaux, P. Egger, J.-L. Raisaro, and Z. Huang. System and Method for Providing a Collective Decentralized Authority for Sharing Sensitive Data. *PCT/EP2016/079649*, 2015.

[82] Why Fraud Detection Needs a Reboot. https://www.forbes.com/sites/forbestechcouncil/2019/01/15/why-fraud-detection-needs-a-reboot/, (accessed: 12.05.2021).

[83] M. Fredrikson, S. Jha, and T. Ristenpart. Model Inversion Attacks that Exploit Confidence Information and Basic Countermeasures. In *ACM Conference on Computer and Communications Security (CCS)*, 2015.

[84] D. Froelicher, P. Egger, J. S. Sousa, J. L. Raisaro, Z. Huang, C. Mouchet, B. Ford, and J.-P. Hubaux. UnLynx: A Decentralized System for Privacy-Conscious Data Sharing. *Proceedings on Privacy Enhancing Technologies Symposium*, 2017(4):232–250, 2017.

[85] D. Froelicher, J. R. Troncoso-Pastoriza, A. Pyrgelis, S. Sav, J. S. Sousa, J.-P. Bossuat, and J.-P. Hubaux. Scalable Privacy-Preserving Distributed Learning. *Proceedings on Privacy Enhancing Technologies Symposium*, 2021.

[86] D. Froelicher, J. R. Troncoso-Pastoriza, A. Pyrgelis, S. Sav, J. S. Sousa, J.-P. Bossuat, and J.-P. Hubaux. System and Method for Privacy-Preserving Distributed Training of Machine Learning Models on Distributed Datasets. *PCT/EP2020/062810*, 2021.

[87] D. Froelicher, J. R. Troncoso-Pastoriza, J. L. Raisaro, M. Cuendet, J. S. Sousa, H. Cho, B. Berger, J. Fellay, and J.-P. Hubaux. Truly Privacy-Preserving Federated Analytics for Precision Medicine with Multiparty Homomorphic Encryption. *Accepted in Nature Communications*, 2021.

[88] D. Froelicher, J. R. Troncoso-Pastoriza, J. S. Sousa, and J. Hubaux. Drynx: Decentralized, Secure, Verifiable System for Statistical Queries and Machine Learning on Distributed Datasets. *IEEE Transactions on Information Forensics and Security (TIFS)*, 2020.

[89] A. Gascón, P. Schoppmann, B. Balle, M. Raykova, J. Doerner, S. Zahur, and D. Evans. Privacy-Preserving Distributed Linear Regression on High-Dimensional Data. *Proceedings on Privacy Enhancing Technologies Symposium*, 2017(4):345–364, 2017.

[90] A. Gaye, Y. Marcon, J. Isaeva, P. LaFlamme, A. Turner, E. M. Jones, J. Minion, A. W. Boyd, C. J. Newby, M.-L. Nuotio, et al. DataSHIELD: Taking the Analysis to the Data, not the Data to the Analysis. *International journal of epidemiology*, 43(6):1929–1944, 2014.

[91] The EU General Data Protection Regulation. https://eugdpr.org/, (accessed: 10.11.2020).

[92] Genomics England. https://www.genomicsengland.co.uk/, (accessed: 30.01.2021).

[93] R. C. Geyer, T. Klein, and M. Nabi. Differentially Private Federated Learning: A Client Level Perspective. *arXiv preprint arXiv:1712.07557*, 2017.

[94] A. Ghosh, T. Roughgarden, and M. Sundararajan. Universally Utility-maximizing Privacy Mechanisms. *SIAM Journal on Computing*, 41(6):1673–1693, 2012.

[95] I. Giacomelli, S. Jha, M. Joye, C. D. Page, and K. Yoon. Privacy-preserving Ridge Regression with Only Linearly-homomorphic Encryption. In *ACNS*, 2018.

[96] R. Gilad-Bachrach, N. Dowlin, K. Laine, K. Lauter, M. Naehrig, and J. Wernsing. Cryptonets: Applying Neural Networks to Encrypted Data with High Throughput and Accuracy. In *ICML*, 2016.

[97] GA4GH. https://genomicsandhealth.org, (accessed: 30.11.2020).

[98] Go programming language. https://golang.org, (accessed: 10.11.2020).

[99] M. K. Goel, P. Khanna, and J. Kishore. Understanding Survival Analysis: Kaplan-Meier Estimate, 2010.

[100] The Go Programming Language. https://golang.org, (accessed: 22.04.2021).

[101] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[102] T. Graepel, K. Lauter, and M. Naehrig. ML Confidential: Machine Learning on Encrypted Data. In *ICISC*, 2012.

[103] D. Grishin, J. L. Raisaro, J. R. Troncoso-Pastoriza, K. Obbad, K. Quinn, M. Misbach, J. Gollhardt, J. Sa, J. Fellay, G. M. Church, and J.-P. Hubaux. Citizen-centered, Auditable and Privacy-Preserving Population Genomics. *Nature Computational Science*, 1(3):192–198, 2021.

[104] J. Groth. A Verifiable Secret Shuffe of Homomorphic Encryptions. In *International Workshop on Public Key Cryptography*, pages 145–160. Springer, 2003.

[105] S. Halevi and V. Shoup. Algorithms in HElib. In *CRYPTO*, 2014.

[106] K. Han and D. Ki. Better Bootstrapping for Approximate Homomorphic Encryption. In *CT-RSA*, 2020.

[107] J. A. Hartigan and M. A. Wong. Algorithm AS 136: A K-means Clustering Algorithm. *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, 28(1):100–108, 1979.

[108] HealthLNK. https://tinyurl.com/y7dqhws6, (accessed: 29.01.2020).

[109] E. Hesamifard, H. Takabi, M. Ghasemi, and R. N. Wright. Privacy-Preserving Machine Learning as a Service. *Proceedings on Privacy Enhancing Technologies Symposium*, 2018.

[110] B. Hie, B. D. Bryson, and B. Berger. Leveraging Uncertainty in Machine Learning Accelerates Biological Discovery and Design. *Cell Systems*, 11(5):461–477, 2020.

[111] B. Hie, H. Cho, and B. Berger. Realizing Private and Practical Pharmacological Collaboration. *Science*, 362(6412):347–350, 2018.

[112] B. Hitaj, G. Ateniese, and F. Perez-Cruz. Deep Models under the GAN: Information Leakage from Collaborative Deep Learning. In *ACM Conference on Computer and Communications Security (CCS)*, 2017.

[113] X. Hu, M. Yuan, J. Yao, Y. Deng, L. Chen, Q. Yang, H. Guan, and J. Zeng. Differential Privacy in Telco Big Data Platform. *VLDB*, 8(12), 2015.

[114] Z. Huang, R. Hu, Y. Guo, E. Chan-Tin, and Y. Gong. DP-ADMM: ADMM-based Distributed Learning with Differential Privacy. *IEEE Transactions on Information Forensics and Security (TIFS)*, 2019.

[115] iDash Competition. http://www.humangenomeprivacy.org/2020/, (accessed: 11.01.2021).

[116] K. A. Jagadeesh, D. J. Wu, J. A. Birgmeier, D. Boneh, and G. Bejerano. Deriving Genomic Diagnoses without Revealing Patient Genomes. *Science*, 357(6352):692–695, 2017.

[117] K. A. Jagadeesh, D. J. Wu, J. A. Birgmeier, D. Boneh, and G. Bejerano. Revealing the Causative Variant in Mendelian Patient Genomes without Revealing Patient Genomes. *bioRxiv*, page 103655, 2017.

[118] K. A. Jagadeesh, D. J. Wu, J. A. Birgmeier, D. Boneh, and G. Bejerano. Keeping Patient Phenotypes and Genotypes Private while Seeking Disease Diagnoses. *bioRxiv*, 2019.

[119] B. Jayaraman and D. Evans. Evaluating Differentially Private Machine Learning in Practice. In *USENIX Security Symposium*, 2019.

[120] B. Jayaraman, L. Wang, D. Evans, and Q. Gu. Distributed Learning Without Distress: Privacy-Preserving Empirical Risk Minimization. In *Advances in Neural Information Processing Systems (NIPS)*, 2018.

[121] Y. Jiang et al. SecureLR: Secure Logistic Regression Model via a Hybrid Cryptographic Protocol. *IEEE TCB*, 2019.

[122] N. Johnson, J. P. Near, and D. Song. Towards Practical Differential Privacy for SQL Queries. *VLDB*, 11(5), 2018.

[123] C. Juvekar, V. Vaikuntanathan, and A. Chandrakasan. GAZELLE: A Low Latency Framework for Secure Neural Network Inference. In *27th USENIX Security Symposium (USENIX Security 18)*, pages 1651–1669, 2018.

[124] A. Kim, Y. Song, M. Kim, K. Lee, and J. H. Cheon. Logistic Regression Model Training Based on the Approximate Homomorphic Encryption. *BMC genomics*, 2018.

[125] M. Kim, J. Lee, L. Ohno-Machado, and X. Jiang. Secure and Differentially Private Logistic Regression for Horizontally Distributed Data. *IEEE Transactions on Information Forensics and Security (TIFS)*, 2019.

[126] M. Kim, Y. Song, S. Wang, Y. Xia, and X. Jiang. Secure Logistic Regression Based on Homomorphic Encryption: Design and Evaluation. *JMIR*, 2018.

[127] E. K. Kogias, P. Jovanovic, N. Gailly, I. Khoffi, L. Gasser, and B. Ford. Enhancing Bitcoin Security and Performance with Strong Consistency via Collective Signing. In *25th USENIX Security Symposium*, pages 279–296, 2016.

[128] L. Kokoris-Kogias, L. Gasser, I. Khoffi, P. Jovanovic, N. Gailly, and B. Ford. Managing Identities using Blockchains and CoSi. In *HotPETs*, 2016.

[129] A. Koloskova, S. U. Stich, and M. Jaggi. Decentralized Stochastic Optimization and Gossip Algorithms with Compressed Communication. *arXiv preprint arXiv*, abs/1902.00340, 2019.

[130] J. Konečný, H. McMahan, D. Ramage, and P. Richtárik. Federated Optimization: Distributed Machine Learning for On-Device Intelligence. *arXiv preprint arXiv:1610.02527*, 2016.

[131] A. Kumar, J. Naughton, and J. M. Patel. Learning Generalized Linear Models Over Normalized Data. In *ACM SIGMOD*, 2015.

[132] T.-T. Kuo, H.-E. Kim, and L. Ohno-Machado. Blockchain Distributed Ledger Technologies for Biomedical and Health Care Applications. *Journal of the American Medical Informatics Association*, 24(6):1211–1220, 2017.

[133] N. Laird and D. Olivier. Covariance Analysis of Censored Survival Data using Log-linear Analysis Techniques. *Journal of the American Statistical Association*, 76(374):231–240, 1981.

[134] Y. LeCun and C. Cortes. MNIST Handwritten Digit Database. *http://yann.lecun.com/exdb/mnist/*, 2010.

[135] M. K. Leung, A. Delong, B. Alipanahi, and B. J. Frey. Machine Learning in Genomic Medicine: A Review of Computational Problems and Data Sets. *Proceedings of the IEEE*, 2015.

[136] B. Li and D. Micciancio. On the security of homomorphic encryption on approximate numbers. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 648–677. Springer, 2021.

[137] W. Li et al. Privacy-Preserving Federated Brain Tumour Segmentation. In *MLMI*, 2019.

[138] B. Libert, S. Ling, K. Nguyen, and H. Wang. Lattice-based Zero-knowledge Arguments for Integer Relations. In *CRYPTO*, 2018.

[139] Y. Lindell. How to Simulate It–A Tutorial on the Simulation Proof Technique. In *Tutorials on the Foundations of Cryptography*, pages 277–346. Springer, 2017.

[140] R. Lindner and C. Peikert. Better Key Sizes (and Attacks) for LWE-Based Encryption. In *CT-RSA*, 2011.

[141] J. K. Lindsey. *Applying Generalized Linear Models.* Springer Science & Business Media, 2000.

[142] C. Liu, X. S. Wang, K. Nayak, Y. Huang, and E. Shi. ObliVM: A Programming Framework for Secure Computation. In *IEEE Symposium on Security and Privacy (SP)*, pages 359–376, May 2015.

[143] F. Liu, Y. Yarom, Q. Ge, G. Heiser, and R. B. Lee. Last-Level Cache Side-Channel Attacks are Practical. In *IEEE Symposium on Security and Privacy*, pages 605–622, May 2015.

[144] J. Liu, M. Juuti, Y. Lu, and N. Asokan. Oblivious Neural Network Predictions via MiniONN Transformations. In *ACM Conference on Computer and Communications Security (CCS)*, 2017.

[145] X. Liu, Y. Zhang, B. Wang, and J. Yan. Mona: Secure Multi-owner Data Sharing for Dynamic Groups in the Cloud. *IEEE Transactions on Parallel and Distributed Systems*, 24(6):1182–1191, 2013.

[146] L. Lorgis, M. Zeller, P. Jourdain, J. Beaune, J.-P. Cambou, B. Vaisse, B. Chamontin, and Y. Cottin. Heart Rate Distribution and Predictors of Increased Heart Rate among French Hypertensive Patients with Stable Coronary Artery Disease. Data from the LHYCORNE Cohort. *Archives of cardiovascular diseases*, 2009.

[147] Low Birth Weight Dataset. https://tinyurl.com/yd6mclh6, (accessed: 21.07.2018).

[148] Y. Lu, T. Zhou, Y. Tian, S. Zhu, and J. Li. Web-Based Privacy-Preserving Multicenter Medical Data Analysis Tools Via Threshold Homomorphic Encryption: Design and Development Study. *Journal of medical Internet research*, 22(12):e22555, 2020.

[149] V. Lyubashevsky, N. K. Nguyen, and G. Seiler. Practical Lattice-Based Zero-Knowledge Proofs for Integer Relations. In *ACM Conference on Computer and Communications Security (CCS)*, 2020.

[150] V. Lyubashevsky, C. Peikert, and O. Regev. On Ideal Lattices and Learning with Errors Over Rings. In *EUROCRYPT*, 2010.

[151] P. J. McLaren, C. Coulonges, I. Bartha, T. L. Lenz, A. J. Deutsch, A. Bashirova, S. Buchbinder, M. N. Carrington, A. Cossarizza, J. Dalmau, et al. Polymorphisms of Large Effect Explain the Majority of the Host Genetic Contribution to Variation of HIV-1 Virus Load. *Proceedings of the National Academy of Sciences*, 112(47):14658–14663, 2015.

[152] H. B. McMahan, E. Moore, D. Ramage, S. Hampson, et al. Communication-efficient Learning of Deep Networks from Decentralized Data. *arXiv preprint arXiv:1602.05629*, 2016.

[153] H. B. McMahan, E. Moore, D. Ramage, and B. A. y Arcas. Federated Learning of Deep Networks using Model Averaging. *arXiv preprint arXiv*, abs/1602.05629, 2016.

[154] H. B. McMahan, D. Ramage, K. Talwar, and L. Zhang. Learning Differentially Private Recurrent Language Models. In *ICLR*, 2018.

[155] Medco Software. https://medco.epfl.ch/, (accessed: 10.01.2021).

[156] L. Melis, G. Danezis, and E. De Cristofaro. Efficient Private Statistics with Succinct Sketches. *The Network and Distributed System Security Symposium (NDSS)*, 2015.

[157] L. Melis, C. Song, E. De Cristofaro, and V. Shmatikov. Exploiting Unintended Feature Leakage in Collaborative Learning. In *IEEE Symposium on Security and Privacy (SP)*, 2019.

[158] L. Melis, C. Song, E. De Cristofaro, and V. Shmatikov. Exploiting Unintended Feature Leakage in Collaborative Learning. In *IEEE Symposium on Security and Privacy (SP)*, pages 691–706. IEEE, 2019.

[159] Lattigo: A Library for Lattice-based Homomorphic Encryption in Go. https://github.com/ldsec/lattigo, (accessed: 14.02.2021).

[160] Mininet, An Instant Virtual Network. http://mininet.org, (accessed: 22.04.2021).

[161] N. Mohammed, D. Alhadidi, B. Fung, and M. Debbabi. Secure Two-Party Differentially Private Data Release for Vertically Partitioned Data. In *IEEE Transactions on Dependable and Secure Computing 11*, pages 59–71, 2014.

[162] P. Mohassel and P. Rindal. ABY 3: A Mixed Protocol Framework for Machine Learning. In *ACM Conference on Computer and Communications Security (CCS)*, 2018.

[163] P. Mohassel and Y. Zhang. SecureML: A System for Scalable Privacy-Preserving Machine Learning. In *38th IEEE Symposium on Security and Privacy*, 2017.

[164] M. Mostert, A. Bredenoord, M. Biesaart, and J. Delden. Big Data in Medical Research and EU Data Protection Law: Challenges to the Consent or Anonymise Approach. *European Journal of Human Genetics*, 2016.

[165] C. Mouchet, J. R. Troncoso-pastoriza, J.-P. Bossuat, and J.-P. Hubaux. Multiparty Homomorphic Encryption from Ring-Learning-With-Errors. In *Proceedings on Privacy Enhancing Technologies Symposium*, 2021.

[166] S. Nakamoto. Bitcoin: A Peer-to-peer Electronic Cash System, 2008.

[167] A. Narayan and A. Haeberlen. DJoin: Differentially Private Join Queries over Distributed Databases. In *10th USENIX Symposium on Operating Systems Design and Implementation, OSDI 12*, pages 149–162, 2012.

[168] R. Nasirigerdeh, R. Torkzadehmahani, J. Matschinske, T. Frisch, M. List, J. Späth, S. Weiß, U. Völker, N. K. Wenke, T. Kacprowski, et al. sPLINK: A Federated, Privacy-Preserving Tool as a Robust Alternative to Meta-Analysis in Genome-Wide Association Studies. *BioRxiv*, 2020.

[169] M. Nasr, R. Shokri, and A. Houmansadr. Comprehensive Privacy Analysis of Deep Learning: Passive and Active White-box Inference Attacks against Centralized and Federated Learning. In *IEEE Symposium on Security and Privacy (SP)*, 2019.

[170] K. Nayak, X. S. Wang, S. Ioannidis, U. Weinsberg, N. Taft, and E. Shi. GraphSC: Parallel Secure Computation Made Easy. In *IEEE Symposium on Security and Privacy (SP)*, pages 377–394, May 2015.

[171] The Network and Distributed System Security Symposium (NDSS) 2021. https://www.ndss-symposium.org/ndss2021/, (accessed: 19.04.2021).

[172] C. A. Neff. A Verifiable Secret Shuffle and its Application to E-Voting. In *ACM Conference on Computer and Communications Security (CCS)*, pages 116–125, 2001.

[173] C. A. Neff. Verifiable Mixing (Shuffling) of ElGamal Pairs, 2004.

[174] J. A. Nelder and R. W. M. Wedderburn. Generalized Linear Models. *Journal of the Royal Statistical Society*, 1972.

[175] Y. Nesterov. Smooth Minimization of Non-smooth Functions. *Mathematical programming*, 2005.

[176] W. S. Ng, B. C. Ooi, K.-L. Tan, and A. Zhou. PeerDB: A P2P-based System for Distributed Data Sharing. In *19th International Conference on Data Engineering*, pages 633–644. IEEE, 2003.

[177] K. Nikitin, E. Kokoris-Kogias, P. Jovanovic, N. Gailly, L. Gasser, I. Khoffi, J. Cappos, and B. Ford. CHAINIAC: Proactive Software-Update Transparency via Collectively Signed Skipchains and Verified Builds. In *26th USENIX Security*, pages 1271–1287, 2017.

[178] V. Nikolaenko, U. Weinsberg, S. Ioannidis, M. Joye, D. Boneh, and N. Taft. Privacy-Preserving Ridge Regression on Hundreds of Millions of Rs. In *2013 IEEE Symposium on Security and Privacy*, pages 334–348, 2013.

[179] OAuth 2.0. accessed:https://oauth.net, (04.09.2019).

[180] O. Ohrimenko, M. Costa, C. Fournet, C. Gkantsidis, M. Kohlweiss, and D. Sharma. Observing and Preventing Leakage in MapReduce. In *Proceedings of the 22Nd ACM SIGSAC Conference on Computer and Communications Security*, pages 1570–1581, 2015.

[181] O. Ohrimenko, F. Schuster, C. Fournet, A. Mehta, S. Nowozin, K. Vaswani, and M. Costa. Oblivious Multi-party Machine Learning on Trusted Processors. In *25th USENIX Security Symposium (USENIX Security 16)*, 2016.

[182] Cothority network library. https://github.com/dedis/onet, (accessed: 22.04.2021).

[183] OpenID Connect. https://openid.net/connect/, (accessed: 04.09.2019).

[184] P4MI. http://p4mi.org, (accessed: 29.01.2019).

[185] P. Paillier. Public-key Cryptosystems Based on Composite Degree Residuosity Classes. In *EUROCRYPT*, 1999.

[186] A. Papadimitriou, R. Bhagwan, N. Chandran, R. Ramjee, A. Haeberlen, H. Singh, A. Modi, and S. Badrinarayanan. Big Data Analytics over Encrypted Datasets with Seabed. In *12th USENIX symposium on operating systems design and implementation (OSDI 16)*, pages 587–602, 2016.

[187] A. Paszke et al. Automatic Differentiation in PyTorch, 2017.

[188] M. Pathak, S. Rane, and B. Raj. Multiparty Differential Privacy via Aggregation of Locally Trained Classifiers. In *Advances in Neural Information Processing Systems (NIPS)*, 2010.

[189] The 17th Privacy Enhancing Technologies Symposium. https://petsymposium.org/2017/, (accessed: 19.04.2021).

[190] The 21st Privacy Enhancing Technologies Symposium. https://petsymposium.org/index.php, (accessed: 19.04.2021).

[191] L. T. Phong, Y. Aono, T. Hayashi, L. Wang, and S. Moriai. Privacy-Preserving Deep Learning via Additively Homomorphic Encryption. *IEEE Transactions on Information Forensics and Security (TIFS)*, 2018.

[192] Pima Indians Diabetes Dataset. https://tinyurl.com/y8o3x8me, (accessed: 14.04.2018).

[193] Plink Software. https://www.cog-genomics.org/plink/, (accessed: 30.11.2020).

[194] R. A. Popa, C. Redfield, N. Zeldovich, and H. Balakrishnan. CryptDB: Protecting Confidentiality with Encrypted Query Processing. In *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*, pages 85–100. ACM, 2011.

[195] K. Powell. The Broken Promise that Undermines Human Genome Research. *Nature*, 590(7845):198–201, 2021.

[196] M. Pratyush, R. Lehmkuhl, A. Srinivasan, W. Zheng, and R. A. Popa. DELPHI: A Cryptographic Inference Service for Neural Networks. In *USENIX Security Symposium*, 2020.

[197] Prio Implementation. https://github.com/henrycg/prio, (accessed: 1.07.2018).

[198] Prostate Cancer Data. https://tinyurl.com/ycsc8f9d, (accessed: 14.03.2018).

[199] R. Rachuri and A. Suresh. Trident: Efficient 4PC Framework for Privacy Preserving Machine Learning. In *The Network and Distributed System Security Symposium (NDSS)*, 2020.

[200] J. Raisaro, F. Marino, J. Troncoso-Pastoriza, R. Beau-Lejdstrom, R. Bellazzi, R. Murphy, E. V. Bernstam, H. Wang, M. Bucalo, Y. Chen, et al. SCOR: A Secure International Informatics Infrastructure to Investigate COVID-19. *Journal of the American Medical Info. Association*, 2020.

[201] J. L. Raisaro, J.-P. Bossuat, J. Troncoso-Pastoriza, V. Junod, and J.-P. Hubaux. MedCo: Enabling the Secure Sharing of Medical Data. A Technical and Legal Perspective. *https://medco.epfl.ch/wp-content/uploads/2021/04/MedCo-Legal-perspective-Swiss.pdf*, 2021.

[202] J. L. Raisaro, J. Troncoso-Pastoriza, M. Misbach, J. S. Sousa, S. Pradervand, E. Missiaglia, O. Michielin, B. Ford, and J.-P. Hubaux. Medco: Enabling Secure and Privacy-Preserving Exploration of Distributed Clinical and Genomic Data. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 2018.

[203] A. Rastogi, M. A. Hammer, and M. Hicks. Wysteria: A Programming Language for Generic, Mixed-Mode Multiparty Computations. In *IEEE Symposium on Security and Privacy*, pages 655–670, May 2014.

[204] M. S. Riazi et al. Chameleon: A Hybrid Secure Computation Framework for Machine Learning Applications. In *ASIACCS*, 2018.

[205] M. S. Riazi, M. Samragh, H. Chen, K. Laine, K. E. Lauter, and F. Koushanfar. XONN: XNOR-based Oblivious Deep Neural Network Inference. In *USENIX Security Symposium*, 2019.

[206] B. D. Rouhani, M. S. Riazi, and F. Koushanfar. Deepsecure: Scalable Provably-Secure Deep Learning. In *ACM DAC*, 2018.

[207] R. M. Samstein, C.-H. Lee, A. N. Shoushtari, M. D. Hellmann, R. Shen, Y. Y. Janjigian, D. A. Barron, A. Zehir, E. J. Jordan, A. Omuro, et al. Tumor Mutational Load Predicts Survival After Immunotherapy Across Multiple Cancer Types. *Nature genetics*, 51(2):202–206, 2019.

[208] S. Sav, A. Pyrgelis, J. R. Troncoso-Pastoriza, D. Froelicher, J.-P. Bossuat, J. S. Sousa, and J.-P. Hubaux. POSEIDON: Privacy-Preserving Federated Neural Network Learning. *The Network and Distributed System Security Symposium (NDSS)*, 2021.

[209] S. Sav, A. Pyrgelis, J. R. Troncoso-Pastoriza, D. Froelicher, J.-P. Bossuat, J. S. Sousa, and J.-P. Hubaux. System and Method for Privacy-Preserving Distributed Training of Neural Network Models on Distributed Datasets. *PCT/EP2020/074031*, 2021.

[210] J. Scheibner, J. L. Raisaro, J. R. Troncoso-Pastoriza, M. Ienca, J. Fellay, E. Vayena, and J.-P. Hubaux. Revolutionizing Medical Data Sharing Using Advanced Privacy Enhancing Technologies: Technical, Legal and Ethical Synthesis. *Journal of Medical Internet Research (in press). doi:10.2196/25120*, 2021.

[211] B. Schoenmakers and P. Tuyls. Efficient Computation Modulo a Shared Secret with Application to the Generation of Shared Safe-Prime Products. In *EUROCRYPT*, 2006.

[212] P. Schoppmann, A. Gascon, M. Raykova, and B. Pinkas. Make Some ROOM for the Zeros: Data Sparsity in Secure Distributed Machine Learning. In *ACM Conference on Computer and Communications Security (CCS)*, 2019.

[213] F. Schuster, M. Costa, C. Fournet, C. Gkantsidis, M. Peinado, G. Mainar-Ruiz, and M. Russinovich. VC3: Trustworthy Data Analytics in the Cloud Using SGX. In *IEEE Symposium on Security and Privacy*, pages 38–54, May 2015.

[214] Scikit-learn, Machine Learning in Python. https://scikit-learn.org/stable/, (accessed: 29.02.2020).

[215] European Comission: Information Exchange. https://ec.europa.eu/home-affairs/what-we-do/policies/law-enforcement-cooperation/information-exchange_en, (accessed: 13.05.2021).

[216] J. V. Selby, A. C. Beal, and L. Frank. The Patient-Centered Outcomes Research Institute (PCORI) National Priorities for Research and Initial Research Agenda. *Jama*, 307(15):1583–1584, 2012.

[217] A. Shamir. How to Share a Secret. *Communications of the ACM*, 1979.

[218] M. J. Sheller, B. Edwards, G. A. Reina, J. Martin, S. Pati, A. Kotrotsou, M. Milchenko, W. Xu, D. Marcus, R. R. Colen, et al. Federated Learning in Medicine: Facilitating Multi-institutional Collaborations Without Sharing Patient Data. *Scientific reports*, 10(1):1–12, 2020.

[219] J. Sherman and W. J. Morrison. Adjustment of an Inverse Matrix Corresponding to a Change in One Element of a Given Matrix. *The Annals of Mathematical Statistics*, 21(1):124–127, 1950.

[220] R. Shokri and V. Shmatikov. Privacy-Preserving Deep Learning. In *Proceedings of the 22nd ACM SIGSAC CCS*, 2015.

[221] R. Shokri, M. Stronati, C. Song, and V. Shmatikov. Membership Inference Attacks Against Machine Learning Models. In *IEEE Symposium on Security and Privacy (SP)*, 2017.

[222] Data Sharing Network (SHRINE). https://www.i2b2.org/work/shrine.html, (accessed: 11.01.2021).

[223] S. Simmons, C. Sahinalp, and B. Berger. Enabling Privacy-Preserving GWASs in Heterogeneous Human Populations. *Cell systems*, 3(1):54–61, 2016.

[224] Y. Son and J. H. Cheon. Revisiting the Hybrid Attack on Sparse and Ternary Secret LWE. *Technical Report https://eprint.iacr.org/2019/1019,*, 2019.

[225] E. M. Songhori, S. U. Hussain, A. R. Sadeghi, T. Schneider, and F. Koushanfar. TinyGarble: Highly Compressed and Scalable Sequential Garbled Circuits. In *IEEE Symposium on Security and Privacy*, pages 411–428, May 2015.

[226] SPECTF. https://archive.ics.uci.edu/ml/datasets/SPECTF+Heart, (accessed: 14.04.2018).

[227] SPHN. https://www.sphn.ch/en.html, (accessed: 29.10.2020).

[228] I. Stoica, D. Song, R. A. Popa, D. Patterson, M. W. Mahoney, R. Katz, A. D. Joseph, M. Jordan, J. M. Hellerstein, J. E. Gonzalez, et al. A Berkeley View of Systems Challenges for AI. *arXiv preprint arXiv:1712.05855*, 2017.

[229] L. Sweeney. k-anonymity: A Model for Protecting Privacy. *International Journal on Uncertainty, Fuzziness and Knowledge-based Systems*, 10(5):557–570, 2002.

[230] E. Syta, I. Tamas, D. Visher, D. I. Wolinsky, P. Jovanovic, L. Gasser, N. Gailly, I. Khoffi, and B. Ford. Keeping Authorities" Honest or Bust" with Decentralized Witness Cosigning. *arXiv preprint arXiv:1503.08768*, 2015.

[231] J. F. Tierney, L. A. Stewart, D. Ghersi, S. Burdett, and M. R. Sydes. Practical Methods for Incorporating Summary Time-to-Event Data into Meta-Analysis. *Trials*, 8(1):16, 2007.

[232] R. Toshniwal, K. Dastidar, and A. Nath. Big Data Security Issues and Challenges. *International Journal of Innovative Research in Advanced Engineering*, 2015.

[233] P. Toulis, E. Airoldi, and J. Rennie. Statistical Analysis of Stochastic Gradient Methods for Generalized Linear Models. In *ICML*, 2014.

[234] S. Truex, N. Baracaldo, A. Anwar, T. Steinke, H. Ludwig, R. Zhang, and Y. Zhou. A Hybrid Approach to Privacy-preserving Federated Learning. In *ACM AISec*, 2019.

[235] S. Tu, M. F. Kaashoek, S. Madden, and N. Zeldovich. Processing Analytical Queries over Encrypted Data. In *VLDB*, volume 6, 2013.

[236] UK Biobank. https://www.ukbiobank.ac.uk/, (accessed: 30.01.2021).

[237] UnLynx experimental implementation. https://github.com/ldsec/UnLynx, (accessed: 22.04.2020).

[238] Vantage6: priVAcy preserviNg federaTed leArninG infrastructurE for Secure Insight eXchange. https://distributedlearning.ai/, (accessed: 11.01.2021).

[239] J. Verbraeken, M. Wolting, J. Katzy, J. Kloppenburg, T. Verbelen, and J. S. Rellermeyer. A Survey on Distributed Machine Learning. *arXiv preprint arXiv:1912.09789*, 2019.

[240] S. Wagh, D. Gupta, and N. Chandran. SecureNN: 3-Party Secure Computation for Neural Network Training. *Proceedings on Privacy Enhancing Technologies Symposium*, 2019.

[241] J. Wang and G. Joshi. Cooperative SGD: A Unified Framework for the Design and Analysis of Communication-efficient SGD Algorithms. *arXiv preprint arXiv*, abs/1808.07576, 2018.

[242] J. Wang and G. Joshi. Cooperative SGD: A Unified Framework for the Design and Analysis of Communication-Efficient SGD Algorithms. In *ICML CodML Workshop*, 2019.

[243] Z. Wang, M. Song, Z. Zhang, Y. Song, Q. Wang, and H. Qi. Beyond Inferring Class Representatives: User-Level Privacy Leakage From Federated Learning. In *IEEE INFOCOM*, 2019.

[244] S. Warnat-Herresthal, H. Schultze, K. P. L. Shastry, S. Manamohan, S. Mukherjee, V. Garg, R. Sarveswara, K. Haendler, P. Pickkers, N. A. Aziz, et al. Swarm Learning as a Privacy-preserving Machine Learning Approach for Disease Classification. *bioRxiv*, 2020.

[245] WITDOM: empoWering prIvacy and securiTy in non-trusteD envirOnMents. https://cordis.europa.eu/project/id/644371/results, (accessed: 30.01.2021).

[246] D. I. Wolinsky, H. Corrigan-Gibbs, B. Ford, and A. Johnson. Scalable Anonymous Group Communication in the Anytrust Model. In *5th European Workshop on System Security*, 2012.

[247] G. Wood et al. Ethereum: A Secure Decentralised Generalised Transaction Ledger. *Ethereum project yellow paper*, 151(2014):1–32, 2014.

[248] OIG Audit Finds Privacy Gaps in NIH All of Us National Research Project. https://tinyurl.com/3xy9ajjy, (accessed: 17.05.2021).

[249] WPF Report – Privacy, the Precision Medicine Initiative, & the All of Us Research Program: Will Any Legal Protections Apply? https://tinyurl.com/wbfttdra, (accessed: 17.05.2021).

[250] Y. Xu, W. Cui, and M. Peinado. Controlled-Channel Attacks: Deterministic Side Channels for Untrusted Operating Systems. In *IEEE Symposium on Security and Privacy*, pages 640–656, 2015.

[251] M. Yaghini, B. Kulynych, G. Cherubin, and C. Troncoso. Disparate vulnerability: On the unfairness of privacy attacks against machine learning. *arXiv preprint arXiv:1906.00389*, 2019.

[252] H. Yang, W. Shin, and J. Lee. Private Information Retrieval for Secure Distributed Storage Systems. *IEEE Transactions on Information Forensics and Security*, 13(12):2953–2964, 2018.

[253] M. Yang and Y. Yang. An Efficient Hybrid Peer-to-peer System for Distributed Data Sharing. *IEEE Transactions on computers*, 59(9):1158–1171, 2010.

[254] R. Yang, M. H. Au, Z. Zhang, Q. Xu, Z. Yu, and W. Whyte. Efficient Lattice-based Zero-knowledge Arguments with Standard Soundness: Construction and Applications. In *CRYPTO*, 2019.

[255] A. C.-C. Yao. How to Generate and Exchange Secrets. In *IEEE SFCS*, 1986.

[256] M. Yin, D. Malkhi, M. K. Reiter, G. G. Gueta, and I. Abraham. HotStuff: BFT Consensus in the Lens of Blockchain. *arXiv preprint arXiv:1803.05069*, 2018.

[257] M. Zamani, M. Movahedi, and J. Saia. Millions of Millionaires: Multiparty Computation in Large Networks. *IACR Cryptology ePrint Archive*, 2014:149, 2014.

[258] D. Zhang. Big Data Security and Privacy Protection. In *ICMCS*, 2018.

[259] N. Zhang, M. Li, and W. Lou. Distributed Data Mining with Differential Privacy. In *IEEE International Conference on Communications (ICC)*, pages 1–5, 2011.

[260] S. Zhang, A. E. Choromanska, and Y. LeCun. Deep Learning with Elastic Averaging SGD. In *NIPS*, 2015.

[261] T. Zhang. Solving Large Scale Linear Prediction Problems Using Stochastic Gradient Descent Algorithms. In *ICML*, 2004.

[262] W. Zheng, R. Deng, W. Chen, R. A. Popa, A. Panda, and I. Stoica. Cerebro: A platform for multi-party cryptographic collaborative learning. In *USENIX Security Symposium*, 2021.

[263] W. Zheng, R. Popa, J. E. Gonzalez, and I. Stoica. Helen: Maliciously Secure Coopetitive Learning for Linear Models. In *IEEE Symposium on Security and Privacy (SP)*, pages 915–929. IEEE Computer Society, 2019.

[264] L. Zhu and S. Han. Deep Leakage from Gradients. In *Federated Learning*, pages 17–31. Springer, 2020.

[265] L. Zhu, Z. Liu, and S. Han. Deep Leakage from Gradients. In *NIPS*, 2019.

[266] X. Zhu, C. Vondrick, C. C. Fowlkes, and D. Ramanan. Do We Need More Training Data? *International Journal of Computer Vision*, 2016.

David Froelicher

Ph.D. Student

PERSONAL DATA

Birth: April 26th, 1992
Nationality: Swiss
Languages:
- English | C2
- French | native
- German | B1

(+41) 79 704 16 28
david@froelicher.net

www.davidfroelicher.com
people.epfl.ch/david.froelicher
github.com/froelich
Google Scholar
Linkedin

EPFL IC IINFCOM LDS
BC 254, Station 14
CH-1015 Lausanne

RESEARCH INTERESTS

Decentralized systems
Applied cryptography
Security and Privacy for Data Sharing
Privacy-enhancing technologies
Genomic Privacy

CODING SKILLS (main)

Golang, Java, Latex

EDUCATION

Master of engineering in communication systems specialized in IT security
Ecole Polytechnique Fédérale de Lausanne | 2016

Bachelor of engineering in communication systems
Ecole Polytechnique Fédérale de Lausanne | 2014

PROFILE

I am a 5th year PhD student under the supervision of Prof. Jean-Pierre Hubaux at the Laboratory for Data Security (LDS) and Bryan Ford at the Decentralized and Distributed Systems Laboratory (DeDiS), at the Ecole Polytechnique Fédérale de Lausanne (EPFL). I earned my MSc and BSc in Computer Science with a specialisation in IT Security from EPFL in 2016. In 2015, I did a master thesis internship in the NEC research laboratory in Heidelberg, Germany, where I have been involved in the design and implementation of a system enabling proofs of retrievability on deduplicated data.

I am currently working on privacy-preserving federated analytics by relying on homomorphic encryption, secure multiparty computation, distributed systems and differential privacy.

EXPERIENCE

Ph.D. Student

EPFL | Switzerland | 2016 - present
Laboratory for data security (LDS) and Decentralized and Distributed Systems Lab (DeDiS)

Research Assistant

EPFL | Switzerland | 2016
Laboratory for data security (LDS)

Master Thesis

NEC Laboratories Europe | Heidelberg, Germany | 2015 - 2016
Analysis of Security Primitives for Public Clouds. Implementing Proofs of Retrievability in Deduplicated Storage Systems.

Master Projects

EPFL | Switzerland | 2013- 2014
- Implement a zero-configuration peer-to-peer network for Map Reduce.
- Dynamically display historical data on Google Earth and enable users to navigate through the use of a Kinect.

PUBLICATIONS (main)

- **D. Froelicher**, J. R. Troncoso-Pastoriza, J. L. Raisaro, M. Cuendet, J. S. Sousa, H. Cho, B. Berger, J. Fellay, and J.-P. Hubaux. ***"Truly Privacy-Preserving Federated Analytics for Precision Medicine with Multiparty Homomorphic Encryption".*** Accepted in Nature Communications, 2021. [biorXiv]
- S. Sav, A. Pyrgelis, J. R. Troncoso-Pastoriza, **D. Froelicher**, J.-P. Bossuat, J. S. Sousa and J.-P. Hubaux. ***"POSEIDON: Privacy-Preserving Federated Neural Network Learning".*** Network and Distributed Systems Security (NDSS) Symposium 2021.

Talks & Awards

7th International Workshop on Genome Privacy and Security (GenoPri'20)
Online | 2020
Presentation on Privacy-Preserving Multi-centric Medical Research with Multi-party Homomorphic Encryption.
[website][talk (at 1h34)][slides]

Microsoft Private AI Bootcamp
Redmond, Washington, USA | 2019
30 selected Ph.D. students invited to a bootcamp with Microsoft Research.
[website][talk][tech report]

IDash Privacy & Security Workshop – Secure Genome Analysis Competition 2019
Indianapolis, Indiana, USA | 2019
Presentation of runner-up solution in Track II: Secure Genotype Imputation using Homomorphic Encryption.
[website] [blog] [talk]

Short Presentation of research interests
Lausanne, Switzerland | 2019
[talk]

Presentation of *UnLynx* at the Privacy Enhancing Technologies Conference
Minneapolis, USA | 2017
[website][talk][slides]

Reviewer Activities

Privacy Enhancing Technologies Symposium | 2019 & 2021

Digital Signal Processing Journal | 2018-present

EURASIP Journal on Information Security | 2018 - present

- **D. Froelicher**, J. R. Troncoso-Pastoriza, A. Pyrgelis, S. Sav, J. S. Sousa, J.-P. Bossuat, and J.-P. Hubaux. ***"Scalable Privacy-Preserving Distributed Learning."*** Privacy Enhancing Technologies Symposium (PETS), volume 3, 2021. (PETS 2021).

- M. Kim, A.Harmanci, J.-P. Bossuat, S. Carpov, J. H. Cheon, I. Chillotti, W. Cho, **D. Froelicher**, N. Gama, M. Georgieva, S. Hong, J.-P. Hubaux, D. Kim, K. Lauter, Y. Ma, L. Ohno-Machado, H. Sofia, Y. Son, Y. Song, J. Troncoso-Pastoriza and X. Jiang. ***"Ultra-Fast Homomorphic Encryption Models enable Secure Outsourcing of Genotype Imputation".*** Cell Systems, 2021. [bioXiv]

- **D. Froelicher**, M. Misbach, J. R. Troncoso-Pastoriza, J.L. Raisaro, J.-P. Hubaux. ***"MedCo^2: Privacy-Preserving Cohort Exploration and Analysis".*** Studies in Health Technology and Informatics, 2020.

- **D. Froelicher**, J.R. Troncoso-Pastoriza, J.S. Sousa and J.P. Hubaux, ***"Drynx: Decentralized, Secure, Verifiable System for Statistical Queries and Machine Learning on Distributed Datasets.",*** IEEE Transactions on Information Forensics and Security , Vol. 15 , Issue. 1, pp. 3035-3050, 2020.

- **D. Froelicher**, P. Egger, J. S. Sousa, J. L. Raisaro, Z. Huang, C. Mouchet, B. Ford, and J.-P. Hubaux: ***"UnLynx: A Decentralized System for Privacy-Conscious Data Sharing."*** Privacy Enhancing Technologies Symposium (PETS), volume 4, pages 152–170, Minneapolis, USA, 2017.

- F. Armknecht, J.-M. Bohli, **D. Froelicher** and G. Karame. ***"SPORT: Sharing Proofs of Retrievability across Tenants."*** Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security, pages 275-287, 2017.

Main Projects

- **DPPH: Data Protection in Personalized Health funded by the Strategic Focus Area Personalized Health and Related Technologies (PHRT) of the ETH Board.** 2018-2021 | Budget: CHF 3M
  This project aims at providing a secure and privacy-conscious framework to enable clinical and genomic data sharing and exploitation across a federation of medical institutions, hospitals and research labs.
  Academic partners: Fellay Group, DeDiS, LDS, GR-JET (EPFL) and Health Ethics and Policy (ETH). Industrial partners: SDSC.

- **MedCo: Enabling the Secure and Privacy-Preserving Exploration of Distributed Clinical and *Omics Cohorts in the Swiss Personalized Health Network (SPHN) funded by the PHRT and the SPHN.**
  2019-2021 | Budget: CHF 0,5 M
  This project aims at testing and deploying in operational environments secure and privacy-conscious cohort explorers dealing with distributed clinical and *omics data.

Journal of Visual Communication and Image Representation | 2018 - present

International Conference on Information Systems Security and Privacy | 2016

Teaching

Master Thesis Supervision
EPFL | 2019
- ***“Privacy-Preserving Statistics on Medical Data Using Homomorphic Encryption”***, John Stephan at Swisscom, Switzerland.
- ***“Efficient Privacy-Preserving Neural Network Inference for Heart Arrhythmia Detection”***, Philipp Chervet at CSEM, Switzerland.

Semester Projects Supervision
EPFL | 2017-present
- 1 Bachelor project
- 12 Master projects
- 2 Summer at EPFL projects

Teaching Assistant
EPFL | 2017-present
- Mobile Network, Master
- Information Security & Privacy, Master
- Advanced Topics on Privacy Enhancing Technologies, Master
- Introduction to Object-oriented Programming, Bachelor

Recreation

Cycling, tennis, badminton, football, squash, ski, guitar, travel

Software Projects (main)

Spindle
https://github.com/ldsec/spindle (private) | 2020 - present
Spindle is a distributed system for the secure and federated training and evaluation of machine learning models (linear/logistic regression, neural networks) on data from multiple sources. It makes use of lattice-based cryptography (***lattigo***). Developed in Golang at the LDS group at EPFL.

iDash solution 2019
https://github.com/ldsec/idash2020 (private) | 2019
Homomorphic encryption-based realization of a client-server privacy-preserving solution for genotype imputation based on the lattice-based homomorphic encryption scheme CKKS. Solution presented in the Homomorphic Encryption track of the iDash Secure Genome Processing Challenge in its 2019 edition (third place). Developed in Golang at the LDS group, EPFL.

Lattigo
https://github.com/ldsec/lattigo
Lattigo is a Go package implementing centralized and multiparty lattice-based cryptographic primitives. Developed in Golang at the LDS group, EPFL.

MedCo
https://medco.epfl.ch
MedCo is the first operational system that makes sensitive medical-data available for research in a simple, privacy-conscious and secure way. It enables hundreds of clinical sites to collectively protect their data and to securely share them with investigators, without single points of failure. The core module is developed in Golang, with additional modules and connectors in Javascript, Java and Scala.

Drynx
https://github.com/ldsec/drynx
Drynx is a library implementing secure multiparty protocols, homomorphic encryption, zero-knowledge proofs and blockchains in order to support a decentralized system that enables privacy-preserving statistical queries and the training and evaluation of machine-learning regression models on distributed datasets. It provides data confidentiality and individuals’ privacy, and ensures the correctness of the computations, protects data providers’ privacy and guarantees robustness of query results. Developed in Golang at the LDS group, EPFL.

UnLynx
https://github.com/ldsec/unlynx
Unlynx is a library implementing interactive protocols to perform distributed cryptographic operations such as key switching and Neff shuffle. The developed prototype is at the core of the operational software, MedCo, that is being deployed at the Swiss University Hospitals. Developed in Golang at the LDS group, EPFL.