



# GLACIATION

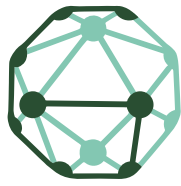
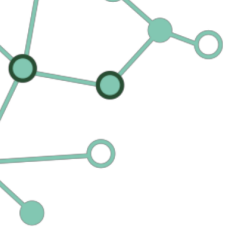
Green responsible privacy  
preserving data operations

## Deliverable D4.1 – Policies and Techniques for Data Protection in Modern Distributed Environments

GRANT AGREEMENT NUMBER: 101070141



This project has received funding from the European Union's HE research and innovation programme under grant agreement No 101070141.



# GLACIATION

**Project Acronym:** GLACIATION  
**Project Full Title:** Green responsible privACy preserving dAta operaTIONs  
**Call Identifier:** HORIZON-CL4-2021-DATA-01-01  
**Type of Action:** RIA  
**Start Date:** 01/10/2022  
**End Date:** 30/09/2025  
**Grant Agreement No:** 101070141

## D4.1 – Policies and Techniques for Data Protection in Modern Distributed Environments

**Executive Summary:** This document reports on the work done in Work Package 4 “Protection Techniques” in the first year of the project

**WP:** WP4

**Author(s):** Stefano Paraboschi, Marco Abbadini, Jonas Böhler, Francesco Capano, Sabrina De Capitani di Vimercati, Dario Facchinetti, Sara Foresti, Giovanni Livraga, Gianluca Oldani, Matthew Rossi, Pierangela Samarati

**Editor:** Stefano Paraboschi

**Leading Partner:** Università degli Studi di Bergamo

**Participating Partners:** SAP SE, UNIMI

**Version:** 1.0

**Status:** Submitted

**Deliverable Type:** Report (R)

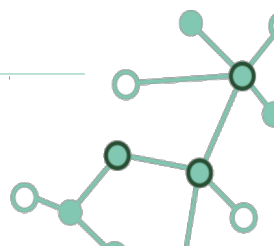
**Dissemination Level:** Public (PU)

**Official Submission Date:** 30/09/2023

**Actual Submission Date:** 30/09/2023

**Data:**

**Date:**



## Disclaimer

This document contains material, which is the copyright of certain GLACIATION contractors, and may not be reproduced or copied without permission. All GLACIATION consortium partners have agreed to the full publication of this document if not declared “Confidential”. The commercial use of any information contained in this document may require a license from the proprietor of that information. The reproduction of this document or of parts of it requires an agreement with the proprietor of that information.

The GLACIATION consortium consists of the following partners:

| No. | Partner Organisation Name                                      | Partner Organisation Short Name | Country |
|-----|--|---------------------------------|---------|
| 1   | MINISTRO DELL'ECONOMIA E DELLE FINANZE                         | MEF                             | IT      |
| 2   | EMC INFORMATION SYSTEMS INTERNATIONAL UNLIMITED COMPANY        | EISI                            | IE      |
| 3   | HIRO MICRODATACENTERS B.V.                                     | HIRO                            | NL      |
| 4   | GOTTFRIED WILHELM LEIBNIZ UNIVERSITÄT HANNOVER                 | LUH                             | DE      |
| 5   | THE LISBON COUNCIL FOR ECONOMIC COMPETITIVENESS ASBL           | LC                              | BE      |
| 6   | UNIVERSITÀ DEGLI STUDI DI MILANO                               | UNIMI                           | IT      |
| 7   | UNIVERSITÀ DEGLI STUDI DI BERGAMO                              | UNIBG                           | IT      |
| 8   | GEIE ERCIM   | ERCIM                           | FR      |
| 9   | EURECOM  | EURECOM                         | FR      |
| 10  | SAP SE   | SAP SE                          | DE      |
| 11  | UNIVERSITY COLLEGE CORK - NATIONAL UNIVERSITY OF IRELAND, CORK | UCC                             | IE      |
| 12  | SOGEI - SOCIETÀ GENERALE D'INFORMATICA S.P.A.                  | SOGEI                           | IT      |
| 13  | LAKESIDE LABS GMBH   | LAKE                            | AT      |
| 14  | ENGINEERING - INGEGNERIA INFORMATICA S.P.A.                    | ENG                             | IT      |
| 15  | EIDGENÖSSISCHE TECHNISCHE HOCHSCHULE ZÜRICH                    | ETH                             | CH      |

## Document Revision History

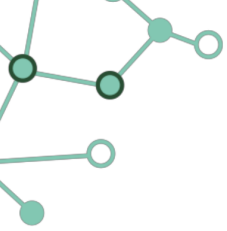
| Version | Description               | Contributions        |
|---------|---------------------------|----------------------|
| 0.0     | Initialize template       | UNIBG                |
| 0.1     | Table of Contents         | UNIBG                |
| 0.9     | Ready for internal review | UNIBG, SAP SE, UNIMI |
| 1.0     | Ready for submission      | UNIBG                |

### Authors

| Author                           | Partner |
|----------------------------------|---------|
| Stefano Paraboschi               | UNIBG   |
| Marco Abbadini                   | UNIBG   |
| Jonas Böhler                     | SAP SE  |
| Francesco Capano                 | SAP SE  |
| Sabrina De Capitani di Vimercati | UNIMI   |
| Dario Facchinetti                | UNIBG   |
| Sara Foresti                     | UNIMI   |
| Giovanni Livraga                 | UNIMI   |
| Gianluca Oldani                  | UNIBG   |
| Matthew Rossi                    | UNIBG   |
| Pierangela Samarati              | UNIMI   |

### Reviewers

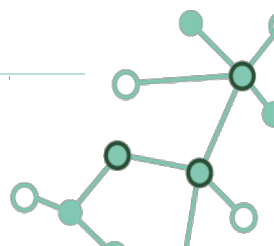
| Name                | Organisation |
|---------------------|--------------|
| Jonas Böhler        | SAP SE       |
| Alessio Chellini    | MEF          |
| Pierangela Samarati | UNIMI        |

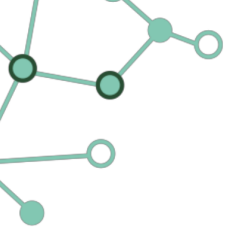


# Table of Contents

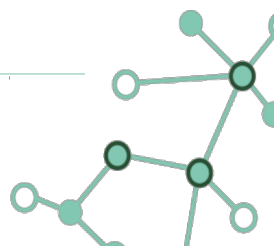
---

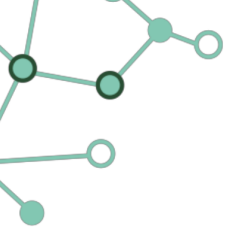
|  |           |
|--|-----------|
| <b>Executive Summary</b>                           | <b>13</b> |
| <b>1 Introduction</b>                              | <b>14</b> |
| <b>2 Policy Model and Language</b>                 | <b>16</b> |
| 2.1 Requirements                                   | 16        |
| 2.2 State of the art                               | 19        |
| 2.2.1 Apache Ranger                                | 20        |
| 2.2.2 Casbin                                       | 21        |
| 2.2.3 Cerbos                                       | 23        |
| 2.2.4 Hasura GraphQL Engine                        | 24        |
| 2.2.5 Keycloak                                     | 25        |
| 2.2.6 Open Policy Agent                            | 28        |
| 2.2.7 Zanzibar-inspired authorization systems      | 30        |
| 2.3 Analysis of authorization frameworks           | 31        |
| 2.4 Open Policy Agent                              | 32        |
| 2.4.1 The OPA document model                       | 32        |
| 2.4.2 The Rego policy language                     | 33        |
| 2.4.3 Examples                                     | 36        |
| 2.5 Integration of Open Policy Agent               | 37        |
| 2.5.1 At application level                         | 37        |
| 2.5.2 At platform level                            | 41        |
| <b>3 Data Wrapping</b>                             | <b>44</b> |
| 3.1 Integrity support for large-scale computations | 44        |
| 3.2 Access revocation for encrypted resources      | 46        |
| 3.3 State of the art                               | 47        |
| 3.4 Mix & Slice                                    | 48        |
| 3.4.1 Blocks, mini-blocks, and macro-blocks        | 48        |
| 3.4.2 Mixing                                       | 49        |
| 3.4.3 Slicing                                      | 51        |
| 3.5 Mixing   | 52        |
| 3.5.1 AES Mixing                                   | 52        |
| 3.5.2 OAEP Mixing                                  | 55        |
| 3.6 Access management                              | 58        |
| 3.7 Analysis                                       | 60        |
| 3.8 Implementation and experiments                 | 63        |
| 3.8.1 Mixing throughput                            | 63        |
| 3.8.2 Access and update throughput                 | 64        |





|  |            |
|--|------------|
| <b>4 Data Sanitization</b>                         | <b>67</b>  |
| 4.1 State of the art.....                          | 68         |
| 4.2 Fundamentals .....                             | 69         |
| 4.3 Anonymization process.....                     | 72         |
| 4.4 Data pre-processing .....                      | 73         |
| 4.4.1 Partitioning techniques.....                 | 73         |
| 4.4.2 Fragments retrieval .....                    | 76         |
| 4.4.3 Partitioning attributes.....                 | 78         |
| 4.5 Data anonymization .....                       | 78         |
| 4.6 Summary and information loss analysis .....    | 80         |
| 4.7 Implementation .....                           | 81         |
| 4.7.1 Architecture.....                            | 82         |
| 4.7.2 Deployment.....                              | 83         |
| 4.8 Experimental results .....                     | 84         |
| 4.8.1 Experimental settings .....                  | 84         |
| 4.8.2 Results.....                                 | 86         |
| <b>5 Secure Collaborative Computation</b>          | <b>89</b>  |
| 5.1 Scope & Methodology .....                      | 90         |
| 5.2 Preliminaries .....                            | 91         |
| 5.2.1 Differential Privacy .....                   | 91         |
| 5.2.2 Differentially Private Machine Learning..... | 92         |
| 5.2.3 Cryptography .....                           | 95         |
| 5.3 Secure and Private Distributed Learning .....  | 97         |
| 5.3.1 Architectures .....                          | 98         |
| 5.3.2 Cryptographic approaches.....                | 99         |
| 5.3.3 Distributed Noise Generation.....            | 101        |
| 5.4 Analysis .....                                 | 104        |
| 5.4.1 Security Features .....                      | 104        |
| 5.4.2 Noise Generation Techniques.....             | 107        |
| 5.5 Related Works.....                             | 108        |
| 5.6 Observation & Future research directions ..... | 109        |
| <b>6 Conclusions</b>                               | <b>111</b> |
| <b>References</b>                                  | <b>112</b> |

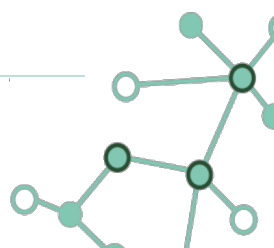


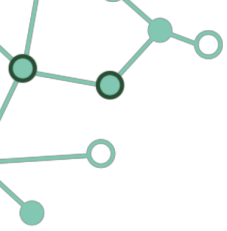


## List of Figures

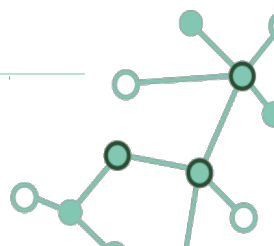
---

|    |   |    |
|----|---|----|
| 1  | Regulation and enforcement of data protection laws around the world [Pip23] .....   | 17 |
| 2  | How phases of the data lifecycle match with data protection use cases [Sha23] .....   | 18 |
| 3  | Architecture of the Apache Ranger framework [TP17] .....  | 21 |
| 4  | High level architecture of Cerbos [Cer23c] .....  | 23 |
| 5  | Architecture of the Hasura GraphQL Engine [Has23a] .....  | 25 |
| 6  | Overview of the Keycloak authorization architecture [Red23a] .....  | 27 |
| 7  | Open Policy Agent architecture overview [Sty23c] .....  | 28 |
| 8  | Example of relationships [OS23b] .....  | 30 |
| 9  | Architecture of the petprofiles1 service [Sty23f] .....   | 36 |
| 10 | Updated architecture of the petprofiles1 service when using OPA to enforce Role-based Access Control [Sty23f] .....   | 36 |
| 11 | Architecture of the petprofiles1 service [Sty23d] .....   | 42 |
| 12 | Visualization of mixing for a macro-block with 16 mini-blocks [0] ... [15] .....  | 50 |
| 13 | Mix&Slice: transforming a resource into fragments .....   | 51 |
| 14 | Mix&Slice: from resource R to fragments .....   | 52 |
| 15 | Visualization of AES mixing, 4 blocks and 16 mini-blocks .....  | 53 |
| 16 | AES mixing strategy .....   | 53 |
| 17 | Propagation of the content of mini-block [0] after consecutive encryption rounds .....  | 54 |
| 18 | Decryption of a resource with 16 mini-blocks. The absence of the mini-block [5] <sub>2</sub> prevents the reconstruction of the whole resource .....  | 54 |
| 19 | OAEP structures .....   | 55 |
| 20 | OAEP mixing with internal layered structure .....   | 56 |
| 21 | OAEP mixing of a macro-block M .....  | 56 |
| 22 | Recursive OAEP mixing .....   | 57 |
| 23 | Recursive OAEP mixing of a macro-block M .....  | 58 |
| 24 | An example of fragments evolution .....   | 59 |
| 25 | Revoke on resource R .....  | 60 |
| 26 | Access to resource R .....  | 60 |
| 27 | Percentage of resource recovered when the revoked user has kept an erasure code whose size is <i>a%</i> of the resource size .....  | 62 |
| 28 | Experimental throughput for several mini-block sizes and varying number of client threads .....   | 64 |
| 29 | Time for the execution of get requests .....  | 65 |
| 30 | Throughput for a workload combining get and put_fragment requests .....   | 65 |
| 31 | An example of a dataset (a), its spatial representation and partitioning (b), and a 3-anonymous and 2-diverse version (c), considering quasi-identifier QI={Age,Country} and sensitive attribute TopSpeed ..... | 70 |
| 32 | Generalization hierarchy for attribute Country .....  | 71 |

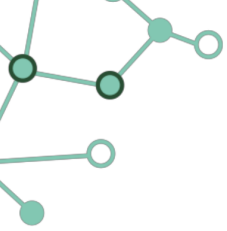




|    |   |     |
|----|---|-----|
| 33 | Overall view of the distributed anonymization process .....   | 72  |
| 34 | Quantile-based partitioning.....  | 74  |
| 35 | Multi-dimensional partitioning.....   | 75  |
| 36 | Quantile-based partitioning.....  | 77  |
| 37 | Multi-dimensional partitioning.....   | 77  |
| 38 | Parallelized multi-dimensional partitioning.....  | 77  |
| 39 | Anonymization algorithm for a fragment $F$ .....  | 79  |
| 40 | Spark-based distributed anonymization system.....   | 82  |
| 41 | Container deployment in a cloud environment.....  | 84  |
| 42 | Execution times of centralized Mondrian and distributed Mondrian varying the number of workers, $k$ , and $\ell$ .....  | 86  |
| 43 | DP and NCP information loss varying the number of workers and $k$ .....   | 88  |
| 44 | Architectures for secure and private collaborative learning, optional communication channels have dashed lines. ....  | 98  |
| 45 | Typical distributed noise generation scenarios where dotted orange arrows indicate noise sampling, and the sampling of central/partial noise is shown as $\zeta/\rho$ , respectively. Optional interactions have dashed lines. .... | 101 |



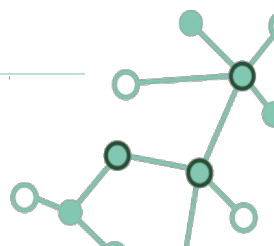




## List of Tables

---

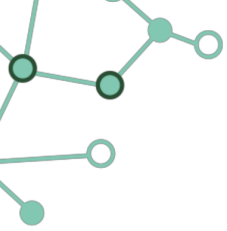
|   |   |     |
|---|---|-----|
| 1 | Comparison of the state-of-the-art authorization frameworks ..... | 32  |
| 2 | Comparison of the different integration options of OPA .....      | 40  |
| 3 | Key features of analyzed works .....                              | 105 |



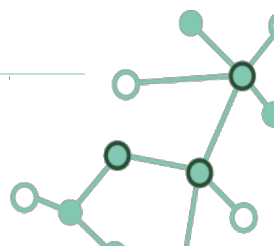
## List of Terms and Abbreviations

| Abbreviation | Description  |
|--------------|--|
| ABAC         | Attribute-based Access Control                       |
| ABE          | Attribute-based Encryption                           |
| ACL          | Access-control List                                  |
| ACS          | American Community Survey                            |
| AD           | Access Directory                                     |
| AES          | Advanced Encryption Standard                         |
| AES-NI       | Advanced Encryption Standard New Instruction         |
| AGEP         | Age of the Person                                    |
| AMD          | Advanced Micro Devices, Inc.                         |
| AONT         | All-Or-Nothing Transform                             |
| AWS          | Amazon Web Services                                  |
| CBC          | Cipher Block Chaining                                |
| CPU          | Central Processing Unit                              |
| CTR          | Counter  |
| DLO          | Dynamic Large Objects                                |
| DP           | Discernability Penalty                               |
| LGPD         | Lei Geral de Proteção de Dados                       |
| LTS          | Long Term Support                                    |
| CCPA         | California Consumer Privacy Act                      |
| EC2          | Elastic Compute Cloud                                |
| EEA          | European Economic Area                               |
| EU           | European Union                                       |
| GB           | Gigabyte   |
| Gbps         | Gigabyte per second                                  |
| GDPR         | General Data Protection Regulation                   |
| GLACIATION   | Green responsibLe privACy preservIng dAta operaTIONs |
| HDFS         | Hadoop Distributed File System                       |
| HE           | Horizon Europe                                       |

| Abbreviation | Description   |
|--------------|---|
| IV           | Initialization Vector   |
| k8s          | Kubernetes  |
| KiB          | Kibibyte  |
| LDAP         | Lightweight Directory Access Protocol   |
| MB           | Megabyte  |
| Mbps         | Megabyte per second   |
| MOSAICrOWN   | Multi-Owner data Sharing for Analytics and Integration respecting Confidentiality and OWNeR control |
| NCP          | Normalized Certainty Penalty  |
| NJ           | New Jersey  |
| NY           | New York  |
| OAEP         | Optimal Asymmetric Encryption Padding   |
| OCCP         | Occupation of the Person  |
| OPA          | Open Policy Agent   |
| OS           | Operating System  |
| PA           | Pennsylvania  |
| PAP          | Policy Administration Point   |
| PDP          | Policy Decision Point   |
| PEP          | Policy Enforcement Point  |
| PERM         | Policy, Effect, Request, Matchers   |
| PIP          | Policy Information Point  |
| POPIA        | Protection of Personal Information Act  |
| PUMS         | Public Use Microdata Sample   |
| QI           | Quais-Identifier  |
| RAM          | Random Access Memory  |
| RBAC         | Role-based Access Control   |
| RDF          | Resource Description Framework  |
| RSA          | Rivest-Shamir-Adleman   |
| SAML         | Security Assertion Markup Language  |
| SPECIAL      | Scalable Policy-aware Linked Data Architecture For Privacy, Transparency and Compliance             |



| Abbreviation | Description              |
|--------------|--------------------------|
| SSD          | Solid-State Disk         |
| ST           | State                    |
| TB           | Terabyte                 |
| URL          | Uniform Resource Locator |
| USA          | United States of America |
| WAGP         | Wage of the Person       |
| WP           | Work Package             |
| XOR          | Exclusive or             |





## Executive Summary

---

The goal of Workpackage 4 “Protection Techniques” is to investigate the opportunities offered by technology for the protection of data in the scenarios considered by GLACIATION. The general vision of GLACIATION is that modern large scale data management systems have to consider at the same time the requirements associated on one hand with privacy and security, on the other hand with the energy consumption and environmental impact deriving from the execution of data processing activities. The correct consideration of privacy and security is then one of the major pillars of the project, looking at how the respect of security can be obtained when working on data that are processed by a distributed system.

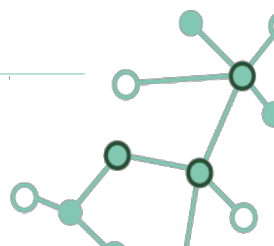
This document reports on the work done in WP4 in the first year of the project. Workpackage 4 is organized in 4 tasks and there is a chapter dedicated to each of the tasks, after a short introduction in Chapter 1.

Chapter 2 describes the work done within Task 4.1 “Policy model and language”. A decision was made in the project to focus on the Kubernetes system, which has found significant adoption in many large-scale systems. In order to maximize the impact of the results of the work in Task 4.1, an extensive analysis was carried out on the current policy management systems for Kubernetes. The Open Policy Agent (OPA) framework was identified as the most interesting solution. The design of the GLACIATION policy model and language will continue targeting its eventual realization within OPA, with the use of the Rego language that is offered by OPA for the representation of policies. Rego is a flexible language and is associated with an engine that supports an expressive declarative model.

Chapter 3 describes the work done within Task 4.2 “Data wrapping”. There are two research contributions that are presented in the chapter. First, the chapter briefly reports on the analysis of probabilistic integrity control for large scale workloads. Then, the chapter focuses on the realization of data protection based on the use of a novel All-Or-Nothing-Transform approach, with the analysis of how this cryptographic technique can support the efficient revocation of access privileges when encryption is used to regulate access to resources, as it is often the case in large distributed scenarios.

Chapter 4 describes the work done within Task 4.3 “Data sanitization”. The chapter describes the realization of a scalable architecture based on Apache Spark for the anonymization of large data collections. The technical choices that have been devised to provide scalability are investigated, together with the consideration of alternatives. Experimental results demonstrate the scalability of the approach and the benefit gained by the use of a distributed architecture.

Chapter 5 describes the work done within Task 4.4 “Secure collaborative computation”. The chapter reports the results of an extensive investigation on the state of the art for the realization of cooperative processing of large data collections where the confidentiality of data is protected. The proposed architecture integrates the application of differential privacy with cryptographic protocols. The goal is the construction of a system offering the same strong privacy protection offered by Differential Privacy in the Local model (LDP), with the same utility and limited precision loss guaranteed by the use of Differential Privacy in the Central model (CDP).





## 1 Introduction

---

The support of privacy and security requirements has a central role in GLACIATION. The goal of Work Package 4 “Protection Techniques” is to perform research in this area, with the goal of advancing the state of the art in the management of security and privacy in distributed and large-scale scenarios. The support for the representation and management of privacy and security requirements will be combined with the representation and evaluation of the energy consumption profile of computations and data transfer, to identify configurations for the execution of workloads able to improve efficiency and at the same time comply with existing restrictions on the processing and transfer of data.

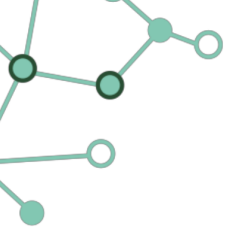
The goals of the project are ambitious and significant challenges lie ahead. The deliverable is produced at the end of the first year of the project, in the initial phases of the definition of the GLACIATION platform, but already there are significant results produced by the research activity on several directions.

The structure of the deliverable follows the organization of Work Package 4, which is structured into 4 tasks.

**Task 4.1 “Policy model and language” (Chapter 2)** The goal of this task is the design of a model and corresponding language for the representation of protection requirements. Significant attention is dedicated to the consideration of the compatibility with the GLACIATION platform. Chapter 2 reports on the careful investigation of several modern frameworks for the management of authorizations in distributed systems. The idea is to identify a concrete framework as a basis for the realization of the GLACIATION policy model. The goal is to design a policy model and language that is non-ambiguous, system-parsable, human-readable and expressive enough to effectively represent privacy regulations and security requirements.

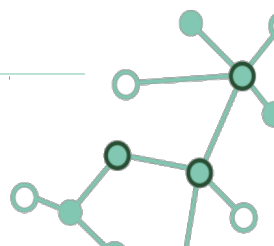
**Task 4.2 “Data wrapping” (Chapter 3)** This task focuses on protection techniques that apply transformations to the data in order to protect it from unauthorized parties. The classical approach relies on efficient symmetric encryption, to be applied when the data has to be stored or transferred through a party that does not have the right to see its content. More advanced forms of protection can be used if computations must be applied on the data. The research in this task has considered several goals. Chapter 3 reports on the adoption of sentinels and twins to realize efficient integrity protection for large computations over data. The chapter also illustrates novel techniques for the application of encryption using an All-Or-Nothing-Transform, which can support the efficient revocation of access privileges.

**Task 4.3 “Data sanitization” (Chapter 4)** This task investigates the protection of data that can be obtained by the application of techniques that sanitize or obfuscate identifying, privacy-sensitive or company-confidential data. The crucial tradeoff in this scenario is to balance the level of protection provided by the application of sanitization with the loss in precision and utility when the sanitized data is used for data analysis, aiming at limiting the perturbation on the outcome of the analysis compared to its execution over unprotected data. Several approaches are available and Chapter 4 reports on the design of a solution for the efficient and scalable application of  $k$ -anonymity over large data collections.



**Task 4.4 “Secure collaborative computation” (Chapter 5)** The opportunity to share data collaboratively is impacted by concerns about privacy, regulations, as well as the need to protect business-critical data assets, in a scenario where there is an increasing urgency to improve analytic and data models with external data sources. The goal of this task is to combine distributed data sources from multiple parties containing sensitive and/or business-critical data, supporting the extraction of useful information, like the discovery of process inefficiencies, optimization potentials, and options for businesses to build more resilient and sustainable processes. To enable data-centric collaboration across companies, the sensitive data will be protected with privacy-enhancing technologies (e.g., secure computation based on cryptographic protocols, differential privacy) which enables learning insights without sharing production or personal data in plaintext. Chapter 5 reports on an extensive analysis of the technological landscape in this domain and presents an architecture where differential privacy is combined with cryptographic protocols in order to support the realization of an architecture supporting the privacy- and security-compliant execution of analysis tasks on data contributed by multiple actors.

The work and research contributions reported in the deliverable are aligned with the goals of GLACIATION and the activity in the other work packages has been crucial in driving the investigation. For instance, the analysis of the existing authorization and policy management frameworks has considered as a central requirement the identification of Kubernetes as a central component of the GLACIATION platform. The work in Work Package 4 will continue in the next phases of the project and there will certainly be many important research results that will be produced by Work Package 4 in the near future.



## 2 Policy Model and Language

---

The edge-core-cloud continuum is a complex and ever-changing landscape with data being processed and stored in a variety of locations, including edge devices, core networks, and cloud computing platforms. This makes it difficult to enforce data regulations, which are often complex and specific to different jurisdictions.

The need of a policy model and language to uniformly represent data regulations requirements was very evident from the early stages of the GLACIATION project, as it facilitates compliance with the regulations, security, and transparency, while maintaining under control the complexity of operations. The goal of Task 4.1 is to contribute to the identification of a policy model and language to enforce data regulations. Specifically, the aim of the framework is to represent rules and constraints that apply to the internal organization operations due to the access of sensitive data. This includes rules about who can access data, how data can be used, and where data can be stored. The definition of these rules is the key to operate on the data according to data protection regulations and guarantee privacy of all users.

In this chapter, we first introduce the requirements of the GLACIATION policy model and language. Then, we discuss the open source solutions the state of the art has to offer with regard to authorization systems and see how they align with the requirements. Thereafter, we describe how Open Policy Agent (OPA), an open source, general-purpose policy engine that unifies policy enforcement across the software stack can be used to manage data protection rules. Finally, we showcase the use of OPA with the goal of securing the GLACIATION platform from deployments that could negatively affect its availability, security, and data protection posture.

The work described in this chapter reports on a portion of the activity performed within Task 4.1 and specifically looks at the concrete authorization and policy management solutions that have the greatest potential to support the policy model in GLACIATION. A currently ongoing line of work, which will contribute to WP6, focuses on the definition of a high-level structure for the GLACIATION policy model, with the identification of the components that are needed for the structured representation of the security and privacy requirements of GLACIATION use cases. The results of this activity are expected to be presented in Deliverable D6.1 “Reference metadata model” and will also contribute to advance the policy model and language solutions that will be presented in Deliverable D4.2 “Tools, models and techniques for data protection” due at the end of the project.

### 2.1 Requirements

---

In order to achieve the desired behavior of representing data regulations with actionable rules the organization’s processes and procedures can use to guide their data management, it is necessary to identify the requirements of the GLACIATION policy model and language.

Considering the ever-growing landscape of data regulations shown in Figure 1, one of the key requirements of a data protection framework is to represent the nuances of different data regulations. Notwithstanding common concepts, data regulations differ in several aspects. Some regulations are more complex than others, and some regulations present requirements depending on the data



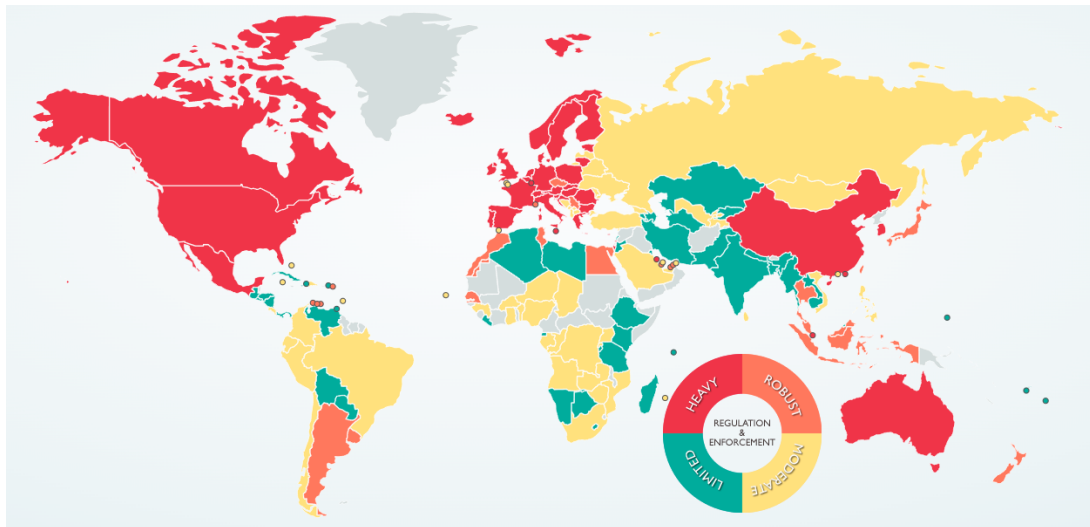
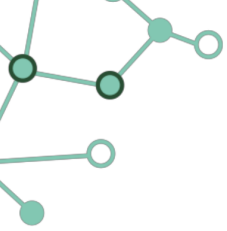
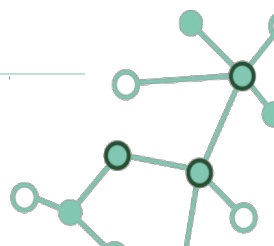


Figure 1: Regulation and enforcement of data protection laws around the world [Pip23]

types and the different data processing, and storage environments. Hence, the model and language should be able to represent data types, such as personal data, financial data, and medical data, but should also be able to represent different data processing and storage environments, such as cloud computing, edge computing, and on-premises computing. In order to support this broad spectrum of options, the policy model and language must be flexible enough to represent the nuances of different data regulations, so to ensure that organizations can comply with all applicable regulations. This is even more important considering that data regulations are constantly evolving, and a data governance framework that is not able to adapt will quickly become obsolete.

The policy model and language should represent confidentiality and integrity requirements of data, so that systems evaluating those rules are able to prevent unauthorized access to data, protect data from unauthorized modifications, and ensure the integrity of data both in transit and at rest. Generally speaking the data protection policy should be able to regulate:

- *Data collection*: how the organization collects personal data, including the methods used, the types of data collected, and the purposes for which the data is collected.
- *Data storage*: how the organization stores personal data, including the security measures in place to protect the data from unauthorized access, use, or disclosure.
- *Data processing*: how the organization processes personal data, including the methods used, the purposes for which the data is processed, and the safeguards in place to ensure that the data is processed fairly and lawfully.
- *Data sharing*: how the organization shares personal data with third parties, including the types of third parties with whom the data is shared, the purposes for which the data is shared, and the safeguards in place to ensure that the data is shared securely.
- *Data deletion*: how the organization deletes personal data, including the methods used, the purposes for which the data is deleted, and the safeguards in place to ensure that the data is



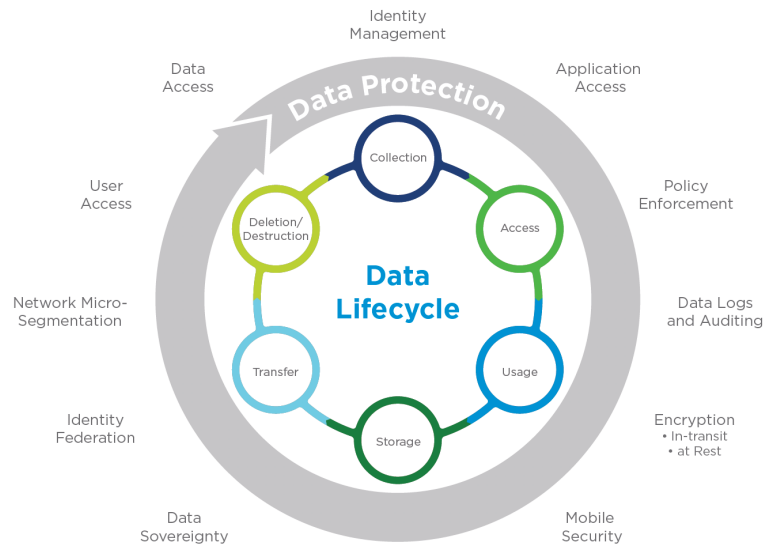
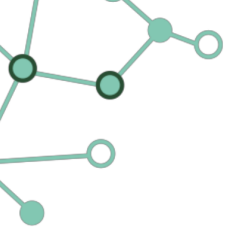


Figure 2: How phases of the data lifecycle match with data protection use cases [Sha23]

deleted securely.

Another key point necessary for the success of a data protection policy is its accessibility to both humans and machines. While data regulations, as every law, are written in natural language, this is not an option when the data protection policy is expected to be consumed by computer systems as much as humans. Indeed, nowadays most of the processes and procedures of an organization dealing with user data rely to some extent on information systems, so it is important that a policy model and language represents its rules in a format that is understandable both by a technical audience (e.g., data protection officers and data engineers) and software systems. First and foremost the language should be easy to use to create and audit data protection policies, with the goal to reduce friction in the adoption of data protection best practices within the organization, but it should also be easy to parse and enforce for software systems to avoid slowing down operations due to slow evaluations of the policy at runtime. Indeed, one of the main challenges in designing a policy model and language is to strike a balance between simplicity and ease of use on one hand, and expressiveness and flexibility, on the other. Simplicity and ease of use are important for making the policy model and language appealing and acceptable for end users. However, they can come at the cost of expressiveness and flexibility, which are necessary for capturing different requirements. Logic-based languages are a good example of this trade-off. They are very expressive and flexible, but they can be complex to use and enforce. This can make them unsuitable for practical settings where simplicity and ease of use are more important. The GLACIATION project aims to provide a policy model and language that is both simple and easy to use, while still being expressive and flexible enough to capture a wide range of requirements.

Finally, as the amount of data that is being collected and processed continues to grow, the model and language should be able to keep up with this increase. Hence the framework should be scalable to handle the increasing volume and complexity of data. This is even more important in the



the edge-core-cloud continuum environment, which has the potential to handle massive amounts of data within a heterogeneous environment, with different types of devices, networks, and applications.

The identification of the policy model and language satisfying the challenges above will provide a valuable framework to the GLACIATION project, as it will guide the data management of the GLACIATION platform with information about the regulations in place and the derived constraints on the treatment of data.

## 2.2 State of the art

---

The research community has investigated a variety of solutions for empowering users with control over their data. These solutions include: modeling GDPR and verifying compliance to it, so to prove the data processing activities align with the regulation (e.g., [ABD19]); enhancing security of users in digital interactions to protect users as they interact with services (e.g., [RFB<sup>+</sup>21, AFO<sup>+</sup>23a, ABF<sup>+</sup>23, AFO<sup>+</sup>23b]); leveraging encryption for enforcing access control (e.g., [ZDX<sup>+</sup>20, BFG<sup>+</sup>21, DFF<sup>+</sup>21b]); enriching authorization specifications (e.g., [BK19]); and supporting privacy-enhanced data flows in IoT processing systems by using techniques such as anonymization, encryption, and differential privacy (e.g., [GPW<sup>+</sup>20, DFF<sup>+</sup>23]).

In addition to these academic works, there have also been a number of European projects that have investigated solutions for the definition of policy models and languages (e.g., PrimeLife<sup>1</sup>, SPECIAL<sup>2</sup>, and MOSAICrOWN<sup>3</sup>). However, these projects have a different focus than GLACIATION. PrimeLife was concerned with the privacy of users accessing a system, SPECIAL focused on the support of GDPR specifications, hence considering a limited set of requirements, and MOSAICrOWN was concerned with the protection of data shared in digital data markets.

Overall, while inspiring, these efforts are complementary to the work of Task 4.1, which aims to provide an easy-to-use, comprehensive, and flexible policy that can be used to address a wide range of data protection requirements for real-world applications deployed on the GLACIATION platform. In order to provide a practical policy model and language applicable and compatible with the GLACIATION platform and its applications, we looked at successful open-source authorization solutions. These represent access control solutions that the industry is already relying on to secure their application and services, thus they represent a natural candidate for the introduction of privacy-related policy requirements. Specifically, after some research, we identified the following candidates: Apache Ranger, Casbin, Cerbos, Hasura GraphQL Engine, Keycloak, Open Policy Agent, Ory, Oso, and Permify.

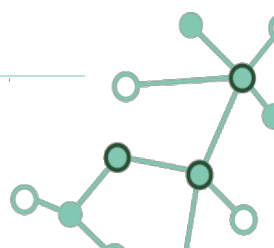
Before diving into the details of each one of these authorization frameworks, it is important to provide further details on the GLACIATION platform, which is a key differentiating factor that significantly influenced our choice. While this is still a work in progress, not yet documented in an official project deliverable, all partners are collaborating on the efforts to design the GLACIATION platform from the very beginning of the project. At the time of writing, a clear consensus has been reached among the partners with regards to the underlying technology that will be the foundation of the platform, and

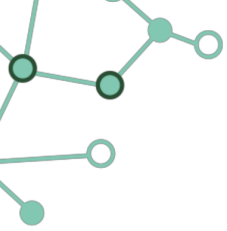
---

<sup>1</sup><https://cordis.europa.eu/project/id/216483>

<sup>2</sup><https://cordis.europa.eu/project/id/731601>

<sup>3</sup><https://cordis.europa.eu/project/id/825333>





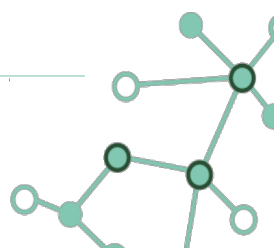
this technology is Kubernetes. Kubernetes, also known as K8s, is an open-source system for automating deployment, scaling, and management of containerized applications. With the widespread adoption of containers among organizations, Kubernetes, the container-centric management software, has become the de facto standard to deploy and operate containerized applications. Inspired by Google's internal cluster management system, Kubernetes makes everything associated with deploying and managing applications easier. Providing automated container orchestration, Kubernetes improves reliability and reduces the time and resources attributed to daily operations. These are just a few of the motivations that brought our consortium to choose Kubernetes as the GLACIATION platform. Considering that the purpose of this deliverable is not to motivate the architectural choices done in Work Package 2 "Kernel Architecture and Development", here we will not dwell further on the description of the merits of this choice, but rather show its implications with respect to the support of the policy model and language.

### 2.2.1 Apache Ranger

---

Apache Ranger is a framework to enable, monitor and manage comprehensive data security across the Hadoop platform [Com23]. It offers a centralized security framework to manage fine-grained access control over Hadoop and related components (Apache Hive, HBase etc.). At its core, Apache Ranger is a web application, which provides policy administration, audit and reporting features. Using its web interface and REST APIs, authorized users are able to manage and enforce security policies regarding access to a resource (e.g., file, folder, database, table, column) for a particular set of users and/or groups within Hadoop. The users of Apache Ranger can also enable auditing of user access and administrative security-related actions, and gather policy analytics for deeper control of the Hadoop environment. With the goal of decentralizing data ownership within an organization, Apache Ranger also provides the ability to delegate administration of certain data to other group owners.

Overall, Apache Ranger standardizes authorization across major Hadoop components by providing a centralized security administration tool that enables managing fine-grained authorization to do a specific action. With respect to the support of a policy model and language, it supports different authorization methods, including Role-based Access Control (RBAC) and Attribute-based Access Control (ABAC). On a more technical note, as shown in Figure 3, it implements these features using lightweight Ranger Java plugins, which run as part of the same process as the namenode (HDFS), Hive2Server (Hive), HBase server (Hbase), Nimbus server (Storm) and Knox server (Knox) respectively. This, on one hand, proves to be very resource efficient, since there is no additional OS level process to manage; on the other hand, it means that the support of Apache Ranger is limited to the set of Hadoop applications it is designed to be used for and no clear path to extend the use of Apache Ranger outside the Hadoop ecosystem is outlined. This may not be a significant limitation in Hadoop environments, but it is a significant one when we consider using Apache Ranger in the GLACIATION platform. Indeed, choosing Apache Ranger would mean imposing our platform users the adoption of the Hadoop data lake infrastructure, which negatively impacts the size of the audience that might be interested in adopting the GLACIATION solutions.



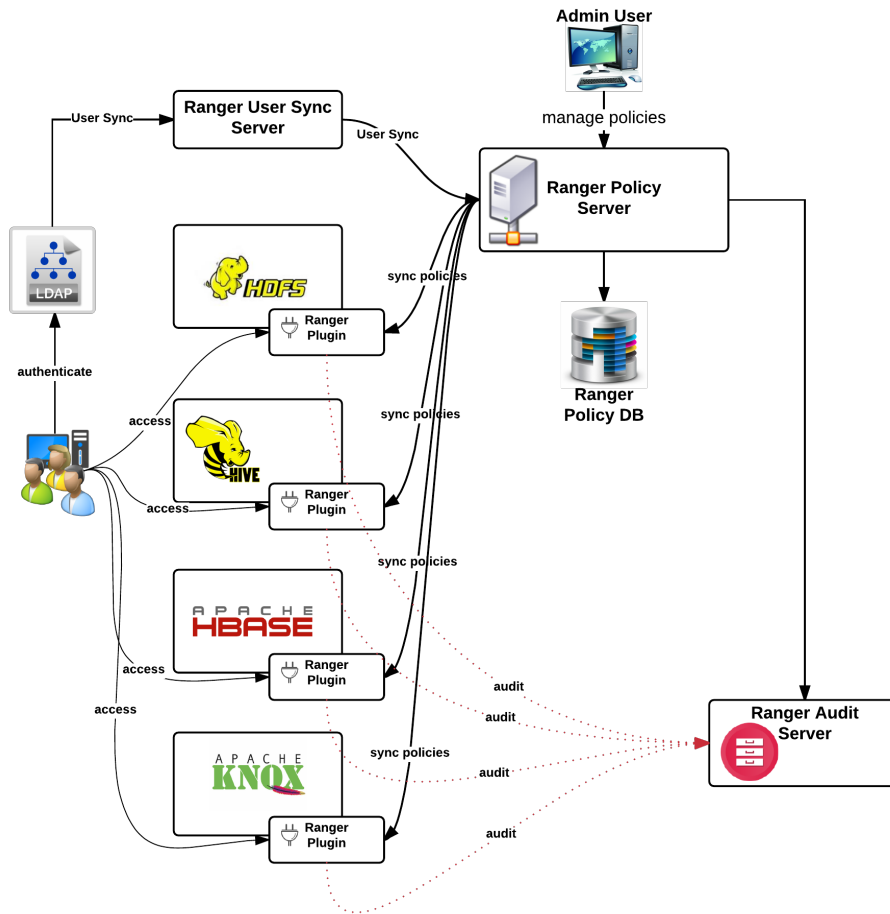
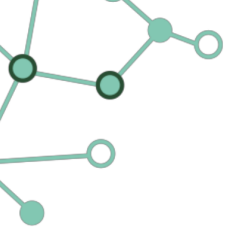


Figure 3: Architecture of the Apache Ranger framework [TP17]

## 2.2.2 Casbin

Casbin [Org23a] is an authorization library that supports several access control models (e.g., Role-Based Access Control, Attribute-Based Access Control) for a plethora of programming languages (i.e., Golang, Java, C/C++, Node.js, Javascript, PHP, Python, C#, Delphi, Rust, Ruby, Swift, Lua, Dart and Elixir). In Casbin, an access control model is abstracted with the PERM metamodel (Policy, Effect, Request, Matchers). So it is possible to change the authorization mechanism for a project by modifying a single configuration file.

Besides the access control model, Casbin embraces flexibility also from an architectural point of view. The role manager used to map the RBAC role hierarchy (user-role mapping) can retrieve data from Casbin policy rules, but also external sources (e.g., LDAP, Okta, Auth0, Azure AD). Policies can be stored into lots of places. Besides memory and file, at the time of writing dozens of databases are supported, from MySQL, Oracle, and Postgres to AWS S3, Cassandra, MongoDB, and Redis. Policy storage is implemented through adapters, and some of them support loading only a subset of the policy stored to allow for efficient policy enforcement in large, multi-tenant environments when



Listing 1: Basic Access-control list model

```
# Request definition
[request_definition]
r = sub, obj, act

# Policy definition
[policy_definition]
p = sub, obj, act

# Policy effect
[policy_effect]
e = some(where (p.eft == allow))

# Matchers
[matchers]
m = r.sub == p.sub && r.obj == p.obj \
    && r.act == p.act
```

Listing 2: Example Casbin rules

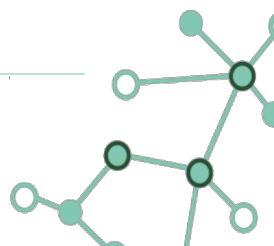
```
p, alice, data1, read
p, alice, data2, read
p, bob, data1, read
p, bob, data1, write
p, bob, data2, read
p, bob, data2, write
p, charley, data1, read
p, charley, data1, write
p, dom, data2, read
p, dom, data2, write
```

parsing the entire policy becomes a performance bottleneck. To keep the size of the library under control, these role data sources and policy storage options are supported with different implementations not part of the main library (with the exception of the default role manager and the default file adapter, respectively).

Thanks to these properties, Casbin has become a very successful project with over 15.5k stars on GitHub [Org23c], and hundreds of projects are using it, including established Fortune 500 companies (e.g., Alibaba Group, Cisco, HP, IBM, Intel, Microsoft, RedHat, Siemens, SpaceX, T-Mobile, Tencent, Verizon, VMware).

Casbin goals and properties align very well also to the goals of the GLACIATION model and language, but, similarly to what previously done with Apache Ranger in Section 2.2.1, we need to consider how it would integrate with the GLACIATION platform. Unsurprisingly, Casbin proves to be a viable choice from this point of view as well, since there are implementations of authorizations middleware for Kubernetes [Org23b]. For example, k8s-authz, the most recently maintained one, is a Kubernetes RBAC & ABAC authorization middleware based on Casbin. This middleware implements a K8s validation admission controller to proxy the requests for any type of K8s resource and perform policy verification on it, so that the user is allowed to perform the operations only if the Casbin enforcer authorizes it.

Although Casbin fulfills most of the foreseen requirements, unfortunately it raises substantial usability concerns. Indeed, while abstracting the access control model with the PERM metamodel allows for extra flexibility, it also imposes significant readability limitations. This can be easily grasped looking at Listings 1 and 2. The PERM metamodel declaratively defines the request parameters, the policy rule structure, their effect, and how to match a request with a rule. This configuration file is hard to understand by the average user even for basic Access Control Lists, and the language gets quite more complicated when it is necessary to introduce relationship concepts and compute functions over existing elements to perform rule matching. Moreover, while a comma-separated list



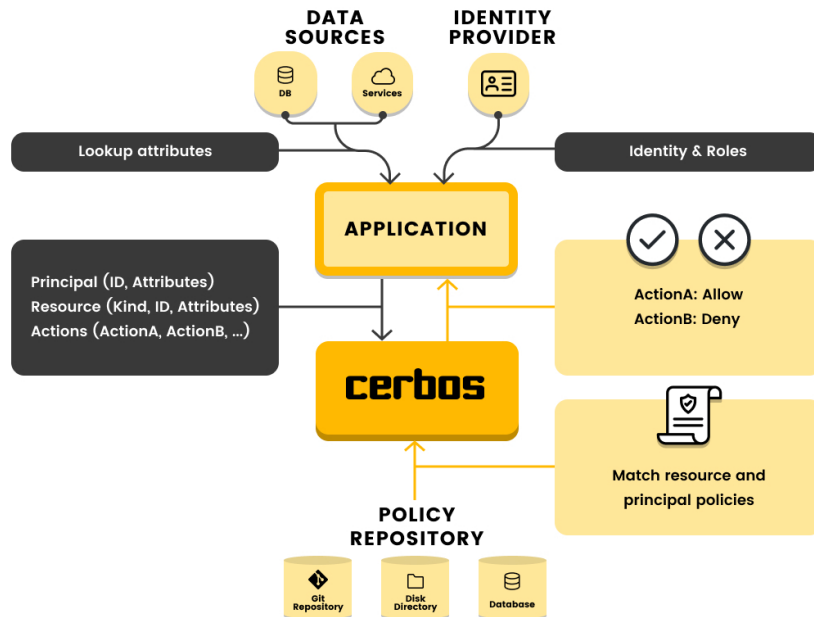


Figure 4: High level architecture of Cerbos [Cer23c]

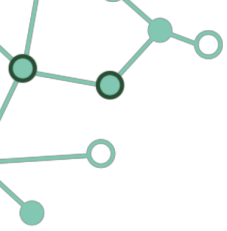
of values may be the most efficient way to store rules, users do not have any information regarding the specific meaning of each column unless they understand the definition of the access control model. This means the same exact rules can have multiple meanings depending on the specific definition of the access control model in use, which again proves confusing and significantly hinders the capability of a user to author, and audit policy rules.

### 2.2.3 Cerbos

Cerbos is a scalable, open-source authorization layer for implementing roles and permissions [Cer23a]. Figure 4 shows how it decouples and centrally manages the authorization logic across all applications and services to gain visibility and deploy access changes across the fleet. Cerbos exposes a simple, language-agnostic API that can be used from any part of the software stack to evaluate policies, make access decisions, and audit access controls with real-time change logs. To facilitate Cerbos integration with existing applications, it comes with Client SDKs for Go, Java, JavaScript, .NET, PHP, Python, Ruby, and Rust. Moreover, it supports all major identity providers (e.g., Auth0, Okta), but also custom directory systems.

From a policy model and language perspective, Cerbos enables users to define powerful, context-aware access control rules for their application resources in simple and intuitive policies expressed in a YAML format. This human-readable access configuration enables collaboration for enforcing security policies and auditing compliance requirements. The extensible roles and conditions allow the model to go beyond Role-based Access Control, and implement Attribute-based rules flexible enough to serve a broad range of use cases.

Cerbos's Policy Decision Point (PDP), the stateless service where policies are executed and de-



cisions are made, runs as a separate process. In a Kubernetes environment, Cerbos can either be deployed as a service or a sidecar, while in other environments it could also be deployed directly as a systemd service or even as an AWS Lambda function. However, at the time of writing, no integration able to regulate access to Kubernetes resources is available open source, which significantly undermines the compatibility of Cerbos with the GLACIATION platform. Moreover, despite its versatile and human-readable policy representation, based on the number of GitHub stars [Cer23b] and the list of companies adopting it, Cerbos is not as popular as alternative solutions. So, while it would be a great option to support the policy model and language of GLACIATION, Cerbos is still a niche authorization system most companies are not familiar with and adopting it would hinder the accessibility of the GLACIATION platform.

## 2.2.4 Hasura GraphQL Engine

---

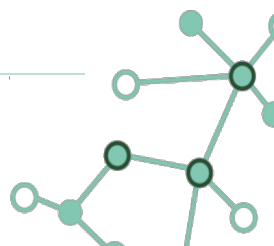
The Hasura GraphQL Engine [Has23b] is a fast open-source GraphQL server that provides a unified, connected, real-time, secured GraphQL API over many popular databases and other data sources (e.g., REST APIs, GraphQL endpoints). Specifically, the Hasura GraphQL Engine connects to a wide range of databases in order to automatically generate a full featured data API, without needing to write handlers, schemas or resolvers. The list of databases is quite extensive and includes Postgres, Microsoft SQL Server, Google BigQuery, Amazon Athena, Snowflake, MySQL, Oracle, MariaDB, and MongoDB.

The definition of a single unified GraphQL API permits to connect data sources together and work with them easily and efficiently. To this end, Hasura provides an optimized syntax to leverage the power of the underlying SQL. Indeed, the Hasura GraphQL Engine does not have any resolvers, but operates as a compiler that transforms GraphQL queries into efficient SQL queries.

With the goal to be a fully-featured backend engine, Hasura also comprises other features and services, including flexible authentication and role-based access control (RBAC) authorization models as shown in Figure 5.

Actual authentication is handled outside Hasura, meaning the responsibility for generating session variables is delegated to an external authentication service. Once a user is authenticated with the external auth service, Hasura authentication can be configured either via JSON web tokens (JWT) or a webhook service, including third-party services like Auth0, Firebase Auth, AWS Cognito, and even custom solutions in order to verify the user and set session variables that then control access to data.

For authorizations, by defining roles and permissions the fine-grained access control system of Hasura permits to specify which users can access which data, ensuring that the application respects the data protection requirements. Specifically, Hasura helps define granular, role and session variable based permission rules to control data access. These permissions utilize the session variables returned by the authentication service and are granular enough to control access to every row or column in the database. With this information Hasura is able to validate the specific GraphQL request against the authorization permission rules. If the operation is allowed, Hasura generates an optimized SQL query, which includes the constraints from the permission rules, and sends it to the database to perform the required operation; fetching the required rows for queries or inserting,





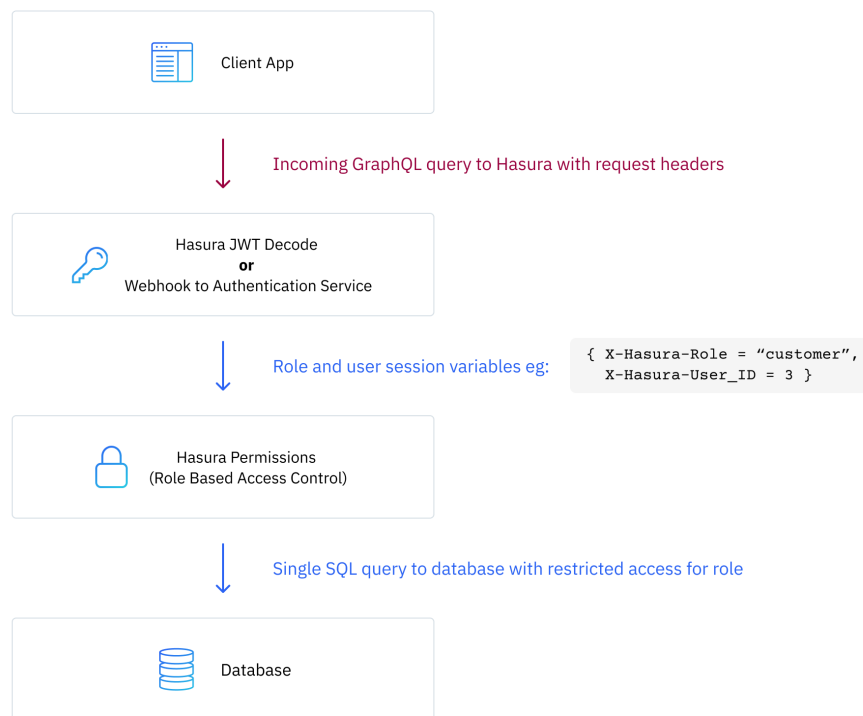
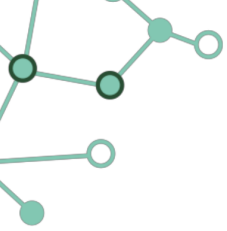


Figure 5: Architecture of the Hasura GraphQL Engine [Has23a]

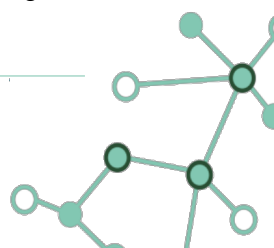
editing or deleting rows for mutations.

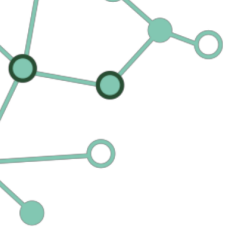
Although Hasura proves to be a very popular framework with over 30k stars on GitHub [Has23c] and multiple enterprises adopting it (e.g., Airbus, Alphabet, Microsoft, Netlify, Philips, Verizon), its authorization system is tightly coupled with the idea to serve all the organization data through a single unified GraphQL API, which unfortunately does not align with the GLACIATION objectives. Indeed, the GLACIATION platform aims to provide a general-purpose policy model and language usable without imposing substantial changes to the application structure and lifecycle. This is key to usability and adoption, since the lower the effort to integrate existing application with the GLACIATION platform, the broader will be the audience interested in the project outcomes.

### 2.2.5 Keycloak

Keycloak [Red23b] is an open source identity and access management framework for applications and services. It primarily provides authentication services, so applications can completely delegate and centralize user management (e.g., users storage and authentication). Specifically, it also offers:

- *Single-sign on*: users authenticate with Keycloak rather than with individual applications. So applications do not have to deal with login forms, authenticating users, and storing users. Once logged-in, the user session is valid for every application. This also applies to logout,





which means users only have to logout once to be logged-out of all applications that use Keycloak.

- *Identity brokering and social login*: the admin console permits to easily enable login with social networks and authenticate users with existing identity providers using standard protocols (e.g., OpenID Connect, Kerberos, SAML 2.0).
- *Admin console*: administrators can centrally manage all aspects of the Keycloak server through the admin console (e.g., enable features, configure identity brokering and user federation, define fine-grained authorization policies, manage users).
- *Account management console*: users can manage their own accounts from the account management console. They can update their profile, change passwords, and setup two-factor authentication. Users can also manage sessions as well as view history for their account.
- *Authorization services*: Keycloak provides fine-grained authorization services; this allows developers to manage permissions for all their services from the Keycloak admin console and gives the developer the power to define exactly the policies they need.

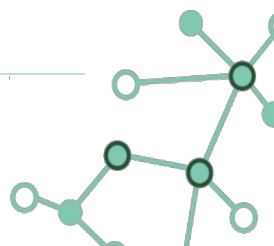
Keycloak supports fine-grained authorization policies and is able to combine different access control mechanisms, including: Attribute-based access control (ABAC), Role-based access control (RBAC), and Context-based access control.

Keycloak provides the necessary means to create permissions for protected resources and scopes, associate those permissions with authorization policies, and enforce authorization decisions in applications and services.

Considering the heterogeneous environments that distinguish current applications where users are distributed across different regions, with different local policies, using different devices, and with a high demand for information sharing, Keycloak authorization services can help organizations improve the authorization capabilities of their applications and services by providing the infrastructure to help avoid code replication across projects and quickly adapt to changes in security requirements.

From a design perspective, as shown in Figure 6, Authorization Services is based on a well-defined set of authorization patterns providing these capabilities:

- *Policy Administration Point (PAP)*: A set of UIs based on the Keycloak Administration Console to manage resource servers, resources, scopes, permissions, and policies.
- *Policy Decision Point (PDP)*: A distributable policy decision point to where authorization requests are sent and policies are evaluated accordingly with the permissions being requested.
- *Policy Enforcement Point (PEP)*: Provides implementations for different environments to actually enforce authorization decisions at the resource server side. Keycloak provides some built-in Policy Enforcers.
- *Policy Information Point (PIP)*: Being based on Keycloak Authentication Server, it provides attributes from identities and runtime environment during the evaluation of authorization policies.



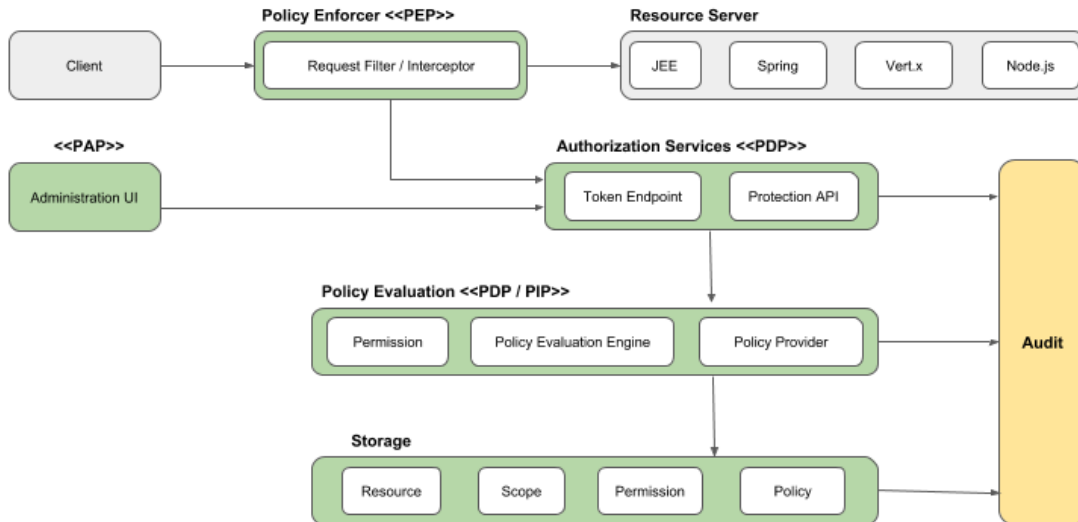


Figure 6: Overview of the Keycloak authorization architecture [Red23a]

Policy Enforcement involves the necessary steps to actually enforce authorization decisions for a resource server. This is achieved by enabling a Policy Enforcement Point at the resource server that is capable of communicating with the authorization server, ask for authorization data and control access to protected resources based on the decisions and permissions returned by the server. Keycloak provides built-in support for enabling the Keycloak Policy Enforcer to Java applications. When the developer enables the policy enforcer all requests sent to the application are intercepted and access to protected resources will be granted depending on the permissions granted by Keycloak to the identity making the request.

The broad use of Keycloak as an authentication solution (17.1k stars on GitHub) leads to consider it as a natural candidate to also manage authorization for applications and services. But, the Keycloak Administration Console requires quite some manual effort to set up a fine-grained policy. This is equally true for the policy enforcement, since built-in support for enabling Keycloak is officially available only for the Java programming language. Finally, considering the specifications of the GLACIATION platform, Keycloak does not provide any integration with Kubernetes, thus limiting its applicability as an authorization framework in our scenario.

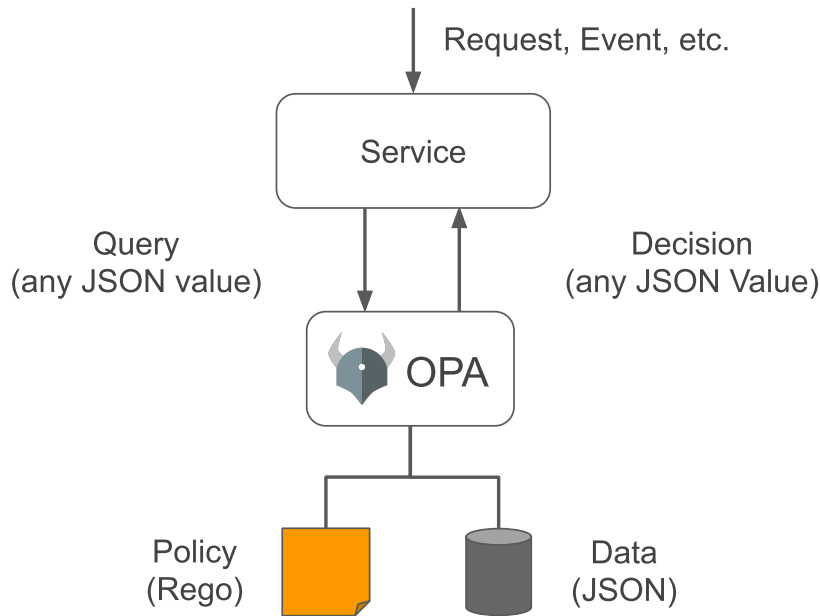


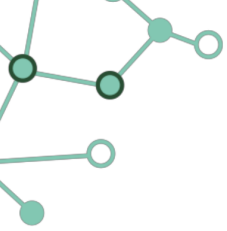
Figure 7: Open Policy Agent architecture overview [Sty23c]

## 2.2.6 Open Policy Agent

The Open Policy Agent (OPA) is an open source, general-purpose policy engine that decouples policy decision-making from policy enforcement [Sty23b]. OPA provides a high-level declarative language that permits to specify policy as code and simple APIs to offload policy decision-making from software. Whenever code needs to make a policy decision it queries OPA and supplies arbitrary structured JSON data as input. The query input is then evaluated against policies and data to produce a policy decision (as shown in Figure 7).

OPA policies are expressed in Rego, a high-level declarative language purpose-built for expressing policies over complex hierarchical data structures. OPA and Rego are domain-agnostic so it is possible to describe almost any kind of invariant in policies. Moreover, policy decisions are not limited to simple yes/no or allow/deny answers. Like query inputs, policies can generate arbitrary structured JSON data as output. These distinctive features enable the widespread use of OPA in multiple scenarios, including microservices, Kubernetes, CI/CD pipelines, API gateways, and more.

Rego was inspired by Datalog, which is a well understood, decades old query language, and extends it to support structured document models (e.g., JSON). Rego focuses on providing powerful support for referencing nested documents and ensuring that queries are correct and unambiguous. The declarative nature of the language lets policy authors focus on what queries should return rather than how queries should be executed. As a result, these queries are simpler and more concise than the equivalent in an imperative language. Moreover, like other applications that support declarative query languages, OPA is able to optimize queries to improve performance. This is key for use cases requiring very low-latency policy decisions. Finally, while OPA supports multiple built-in functions (e.g., string manipulation, arithmetic, JWT verification, and executing HTTP requests), the set can be extended with custom built-in functions and plugins that implement new functionality.



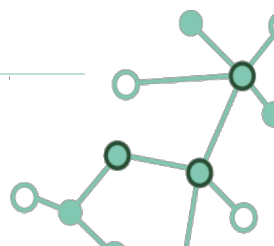
To help write, verify the correctness, and use policies, OPA embraces policy-as-code by complementing the policy engine with tools that help developers use and understand the policies they put in place (e.g., integrated development environments, testing, profiling, coverage, automated performance tuning, and hot reloading). This significantly speeds up the development process of new rules and reduces the amount of time needed to modify rules as requirements evolve.

From an architectural perspective, OPA offers great flexibility with different setups that permits to balance integration effort, availability, and consistency. It is possible to deploy OPA as a separate process on the same host as the application service, and interact with it either by changing the service's code (i.e., importing an OPA-enabled library), or transparently by using a network proxy integrated with OPA (e.g., Envoy). Furthermore, it is even possible to embed OPA policies into the application service and evaluate them from the service itself. For services implemented in the Go programming language a library is available, while for every other language the policy can be compiled to WebAssembly instructions and run by a WebAssembly runtime.

Transparent integration of OPA with existing applications is very important as it significantly reduces the friction in the introduction of the new security measures. To this end, OPA offers integration with the Envoy network proxy. Envoy is a L7 proxy designed for large modern service oriented architectures. As part of its feature set, it supports an external authorization filter which calls an authorization service to check if the incoming request is authorized or not. The OPA ecosystem comes with a plugin to enforce OPA policies with Envoy. Developers can use this version of OPA to enforce fine-grained, context-aware access control policies with Envoy without modifying microservice implementations.

With regard to off-the-shelf integrations, OPA's community provides first-class integration between OPA and Kubernetes with OPA Gatekeeper, a Kubernetes Admission Controller which enforces policies on objects during create, update, and delete operations. For example, by using OPA Gatekeeper it is possible to require the use of a specific image registry, the specification of resource requirements and limits, and prevent the creation of conflicting objects. Moreover, by deploying it as a mutating admission controller it is even possible to mutate incoming objects (e.g., inject sidecar containers into pods, set specific annotations on resources, and point container images at the corporate image registry).

Overall, OPA represents a prime example of a policy-as-code authorization system. It utilizes code to define authorization policies, facilitating their modification and offering a higher degree of flexibility. Rego, its declarative policy language, simplifies the definition of complex authorization policies, and makes OPA a popular solution across many organizations. Indeed, while originally created by Styra, OPA is now a graduated project in the Cloud Native Computing Foundation (CNCF) landscape, and multiple organization adopt it as their authorization system of choice (e.g., Atlassian, Capital One, Cisco, Cloudflare, Goldman Sachs, Google Cloud, Netflix, Pinterest, SAP/InfraBox, T-Mobile, Tripadvisor). These features, along with the broad ecosystem built around OPA, make Rego a great candidate to support the policy model and language of the GLACIATION platform.



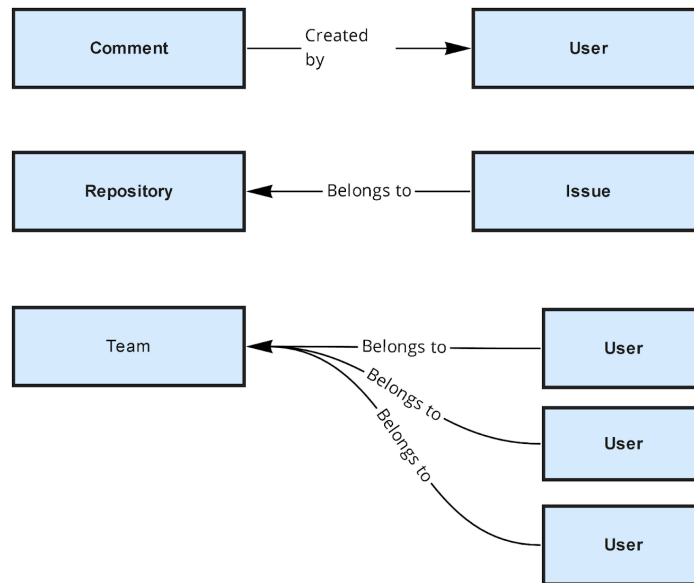


Figure 8: Example of relationships [OS23b]

## 2.2.7 Zanzibar-inspired authorization systems

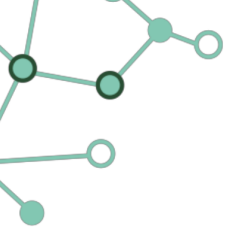
Zanzibar [PCB<sup>+</sup>19] is Google’s purpose-built authorization system. It is a centralized authorization database built to take authorization queries from high-traffic apps and return authorization decisions. An instance of Zanzibar hosts a list of permissions and responds to queries from many apps.

Google Zanzibar is a system to manage Relationship-Based Authorization (ReBAC). Relationship-based authorization means organizing permissions based on relationships between resources (see Figure 8). For instance, allowing only the user who created a post to edit it. Relationships include data ownership, parent-child relationships, groups, and hierarchies.

In general, authorization in ReBAC is performed by traversing the directed graph of relationships. The nodes and edges of this graph are very similar to triples in the Resource Description Framework (RDF) data format. ReBAC systems support hierarchies of relationships, and some support more complex definitions that include algebraic operators on relationships such as union, intersection, and difference. Graph-based systems shine in scenarios requiring ReBAC and can describe hierarchical and nested relationships. They ensure data consistency even in high-volume scenarios and offer reverse indices, enabling reverse queries (not only does x have access to y, but also who has access to y).

Engineering teams are increasingly adopting services for core infrastructure components, and this applies to authorization too. There are a number of authorization-as-a-service options available to those who want something like what Google made available to its internal engineers via Zanzibar (e.g., Ory [Ory23], Oso [OS23a], and Permify [Per23]).

Despite their advantages, deploying a Zanzibar-based graph invariably introduces a sizable and complex system into the cloud environment, often necessitating reliance on a hosted service. This



dependency can instigate latency concerns and further scaling challenges. Specifically, compared to policy-as-code, graph-based systems exhibit lower performance and are practically unfeasible to run at the edge due to their size. They also come with higher latency, owing to their non-local nature, and are limited to simple ReBAC policies. Due to these reasons we considered Zanzibar-based frameworks not to fit with the general requirements of the GLACIATION platform.

## 2.3 Analysis of authorization frameworks

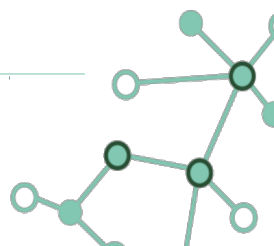
---

With the research conducted in the previous section regarding the open-source, state-of-the-art authorization frameworks available, now we have enough information about the pros and cons of each option to make an informed decision about what framework better suits the GLACIATION requirements (Section 2.1).

To summarize in a very compact and somewhat oversimplified representation, in Table 1 we highlight how the authorization frameworks compare on a few dimensions:

- *Generality*: it represents the ability of the framework to be applied to a broad and diverse spectrum of use cases. As we do not want to limit the scope of use cases our platform can work with, we are seeking for the most generally applicable framework.
- *Flexibility*: this feature goes hand in hand with the previous one, and it is associated with how much the policy system can be tuned to the specific organization needs. The more expressive the framework is, the better it can capture policy nuances.
- *Usability*: it represents how easy it is to audit and specify policies. Some of the frameworks offer simple UIs to declare and observe policies, others require learning a policy language. While the latter are inevitably harder to understand, they come with a lot of advanced, but necessary, features, like versioning, testing, and profiling.
- *Broad support of programming languages*: it depends on the number of libraries available to facilitate the interaction of different languages with the authorization framework. It is very limited when the support is tight to a single language, and limited when there is still missing support to widely used back-end languages.
- *Popularity*: we consider the number of stars the open-source project has on its GitHub repository as a good estimator of the number of businesses that are using it in practice. The higher the popularity of a tool, the higher the possibility developers already know about it, and the lower is the cost of entry to use the policy model and language.
- *Existing integration with Kubernetes environments*: the foundational technology used to support the GLACIATION platform is Kubernetes, as such any already existing integration with K8s helps with the rapid development of the GLACIATION platform.

Based on our evaluation, the best candidate framework to support the GLACIATION policy model and language is Open Policy Agent (OPA). Indeed, it satisfies all the requirements our policy system needs to meet, and it comes with a thriving ecosystem of integrations that will significantly help with adoption of the GLACIATION platform.












| Authorization framework  | General purpose | Flexible model | Usability | Language support | Popularity (#stars) | K8s integration |
|--|-----------------|----------------|-----------|------------------|---------------------|-----------------|
|  <b>Apache Ranger</b>     | No              | No             | Yes       | Very limited     | 770                 | No              |
|  <b>casbin</b>            | Yes             | Yes            | Limited   | Yes              | 15.6k               | Yes             |
|  <b>cerbos</b>            | Yes             | Yes            | Yes       | Limited          | 1.6k                | No              |
|  <b>HASURA</b>            | No              | Limited        | Yes       | Yes              | 30.1k               | No              |
|  <b>KEYCLOAK</b>          | Limited         | No             | Yes       | Very limited     | 17.1k               | No              |
|  <b>Open Policy Agent</b> | Yes             | Yes            | Yes       | Yes              | 8.4k                | Yes             |
|  <b>ORY / keto</b>        | Limited         | Limited        | Yes       | Yes              | 4.2k                | No              |
|  <b>oso</b>               | Limited         | Limited        | Yes       | Yes              | 3.2k                | No              |
|  <b>PERMIFY</b>           | Limited         | Limited        | Yes       | Yes              | 1.6k                | No              |

Table 1: Comparison of the state-of-the-art authorization frameworks

## 2.4 Open Policy Agent

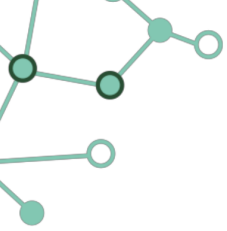
OPA is a lightweight general-purpose policy engine. Services offload policy decisions to OPA by executing queries. OPA evaluates policies and data to produce query results (which are sent back to the client). Policies are written in the Rego high-level declarative language and can be loaded dynamically into OPA.

### 2.4.1 The OPA document model

OPA is not tied to any particular domain model. OPA policies (written in Rego) make decisions based on arbitrary hierarchical structured data, and can represent these decisions as arbitrary structured data (e.g., booleans, strings, maps, maps of lists of maps, etc.). Data can be loaded into OPA from the outside world using push or pull interfaces that operate synchronously or asynchronously with respect to policy evaluation. These documents almost always contribute to the policy decision-making logic. However, policies can also make decisions based on other rules (possibly authored by different groups). With Rego, it is possible to refer to all this input through a global variable called `data` using the exact same dot/bracket-style reference syntax. Policy authors need to learn only one way of modeling and referring to information that drives policy decision-making, due to consistency in the types of values represented and the way they are referenced.

Note that for external data (i.e., data uploaded from outside of OPA) their location under `data` is controlled by the software doing the loading. On the other hand, the location of intermediary rules





results under `data` is controlled by policies themselves using the `package` directive in the language.

External data loaded asynchronously are always accessed under the `data` global variable. On the other hand, documents can also be pushed or pulled into OPA synchronously when software queries OPA for policy decisions. We refer to external data pushed synchronously as *input*. Policies can access these inputs under the `input` global variable. To pull base documents during policy evaluation, OPA exposes (and can be extended with custom) built-in functions. Built-in functions return values can be assigned to local variables and sent back as intermediary rule result. To avoid naming conflicts, parameters loaded synchronously are kept outside of the `data` global variable.

## 2.4.2 The Rego policy language

---

Rego was inspired by Datalog, a well understood query language, and provides extensions to it in the form of support for structured document models. Rego queries are assertions on data stored in OPA. These queries can be used to define policies that enumerate instances of data that violate the expected state of the system. This subsection introduces the main aspects of Rego. However, to have a complete overview of the Rego syntax and semantics, we suggest to look at the documentation [Sty23e].

### Scalar values

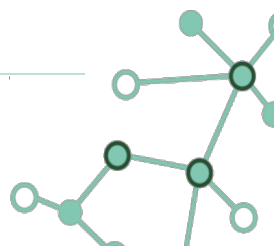
Scalar values are the simplest type of term in Rego. Scalar values can be strings, numbers, booleans, or null. This is useful for defining constants that are referenced in multiple places (e.g.,  $\pi := 3.14159$ ).

Rego supports two different types of syntax for declaring strings. The first is likely to be the most familiar: characters surrounded by double quotes. In such strings, certain characters must be escaped to appear, such as double quotes themselves, backslashes, etc. The other type of string declaration is a raw string declaration. These are made of characters surrounded by backticks (i.e., ```), with the exception that raw strings may not contain backticks themselves. Raw strings are what they sound like: escape sequences are not interpreted, but instead taken as the literal text inside the backticks. Raw strings are particularly useful when constructing regular expressions for matching, as it eliminates the need to double escape special characters.

### Composite values

Composite values define collections. There are multiple types of collection:

- Arrays are an ordered collection of any kind of values.
- Objects represent unordered key-value collections. In Rego, any value type can be used as an object key. For example, the assignment in Listing 3 maps port numbers to a list of IP addresses (represented as strings). Because JSON does not support non-string object keys, when Rego values are converted to JSON every object key is marshalled as strings.
- Sets are unordered collections of unique values. Just like other composite values, sets can be defined in terms of scalars, variables, references, and other composite values. OPA represents



Listing 3: Document describing a simplistic deployment environment

```
sites := [  
  {  
    "region": "east",  
    "name": "prod",  
    "servers": [  
      { "name": "web-0", "hostname": "hydrogen"},  
      { "name": "web-1", "hostname": "helium" },  
      { "name": "db-0", "hostname": "lithium" }  
    ]  
  },  
  {  
    "region": "west",  
    "name": "smoke",  
    "servers": [  
      { "name": "web-1000", "hostname": "beryllium" },  
      { "name": "web-1001", "hostname": "boron" },  
      { "name": "db-1000", "hostname": "carbon" }  
    ]  
  }  
]
```

set documents as arrays when serializing to JSON or other formats that do not support a set data type. The important distinction between sets and arrays or objects is that sets are unkeyed while arrays and objects are keyed, i.e., it is not possible to refer to the index of an element within a set.

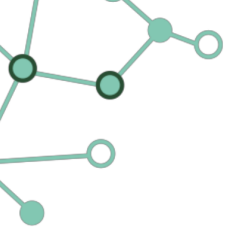
## References

References are used to access nested documents. The simplest reference contains no variables. For example, the reference `sites[0].servers[1].hostname` returns the hostname of the second server in the first site document from the example data in Listing 3 (i.e., “helium”). References are typically written using the *dot-access* style. The canonical form avoids the dot sign and closely resembles dictionary lookup in Python: `sites[0][“servers”][1][“hostname”]`. Both forms are valid, however, the dot-access style is typically more readable and so preferred when it is possible to use it.

References can include variables as keys. References written this way are used to select a value from every element in a collection (e.g., `sites[i].servers[j].hostname`). If the variables are unused outside the reference, it is preferable to replace them with an underscore ‘\_’ character.

## Rules

The `default` keyword allows policies to define a default value for documents produced by rules. The following piece of Rego code shows how it is possible to define the behavior of default deny for the allow rule.



```
default allow := false
```

When OPA evaluates a rule, OPA generates the content of the document that is defined by the rule. The rule itself can be understood intuitively as: *rule-name IS value IF body*. When evaluating rule bodies, OPA searches for variable bindings that make all of the expressions true. There may be multiple sets of bindings that make the rule body true. The rule body can be understood intuitively as: *expression-1 AND expression-2 AND ... AND expression-N*. In the following, we provide a more comprehensive example of what a Rego rule looks like:

```
allow := true if {  
  input.user == "bob"  
  input.method == "GET"  
}
```

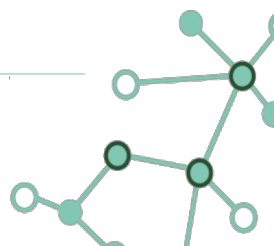
In the example above, `allow` will be true when user *bob* is accessing an endpoint with the *GET* method. Note, the same rule can be also represented in a more compact way, by omitting the optional future keyword `if` and the *value* (which defaults to `true`). So the same rule above can be rewritten as:

```
allow {  
  input.user == "bob"  
  input.method == "GET"  
}
```

A rule may be defined multiple times with the same name. When a rule is defined this way, we refer to the rule definition as incremental because each definition is additive. The document produced by incrementally defined rules is the union of the documents produced by each individual rule. More precisely, an incrementally defined rule can be intuitively understood as *<rule-1> OR <rule-2> OR ... OR <rule-N>*. The following policy provides an example of an incremental rule:

```
default allow := false  
  
allow if {  
  input.user == "bob"  
  input.method == "GET"  
}  
  
allow if input.user == "alice"
```

Specifically, in the example above, `allow` will be true when user *alice* accesses an endpoint with any method, and when user *bob* accesses an endpoint with the *GET* method. On the other hand, due to the default behavior declared with the same name, every other request would be denied (i.e., the value of `allow` will be false). Without the default definition, the `allow` document would simply be undefined, since all of the rules sharing the same name are undefined.



### 2.4.3 Examples

In this subsection, we will refer to a couple of examples provided by the authors of Open Policy Agent in order to showcase the power and integrability of their framework. The first one focuses on how an application service can be modified to ensure consistent protection of sensitive data. The second one identifies one of the security property the GLACIATION platform should ensure to guarantee data protection, i.e., making sure that the applications deployed in the Kubernetes cluster are trusted by the organization.

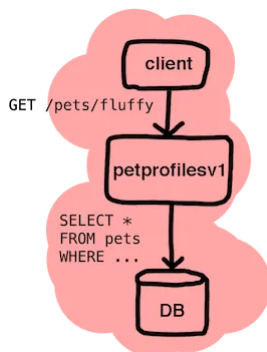


Figure 9: Architecture of the petprofilesv1 service [Sty23f]

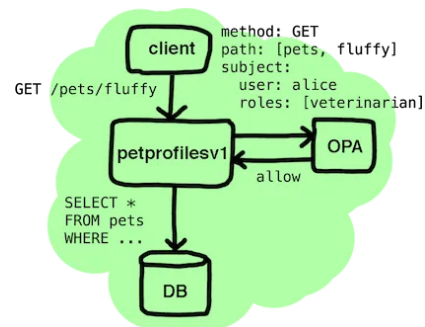


Figure 10: Updated architecture of the petprofilesv1 service when using OPA to enforce Role-based Access Control [Sty23f]

#### Protect access to sensitive data

Throughout this example [Sty23f] we will refer to a hypothetical service (petprofilesv1) used by a chain of veterinary clinics. The service exposes an HTTP API that serves profiles for pets at the clinics. As shown in Figure 9, the service implements the HTTP API by querying a SQL database and returning the results.

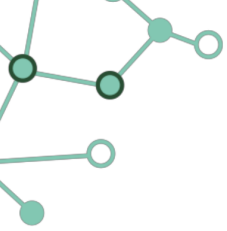
In this setting, we want to enforce a simple role-based authorization policy in our hypothetical service that says: “Only veterinarians are allowed to read pet profiles”. To make authorization decisions and enforce this policy, we need to give OPA input such as the identity of the caller, the operation the caller wants to perform, and the resource(s) the operation affects (see Figure 10).

To implement this policy, we could create a simple allow rule in OPA:

```
default allow = false

allow = true {
  input.method = "GET"
  input.path = ["pets", name] # name unused here
  input.subject.roles[_] = "veterinarian"
}
```

When the service queries OPA, it provides the identity of the caller (`input.subject`), the operation being performed (`input.method`), and the resource being operated on (`input.path`). The response from OPA indicates whether the request should be allowed (true) or denied (false).



## Ensure deployment of trusted images

Kubernetes allows decoupling policy decisions from the inner workings of the API Server by evaluating admission controller webhooks whenever a resource is created, updated or deleted.

Gatekeeper, i.e., the admission controller integrating OPA with Kubernetes, can validate resources in the cluster against Gatekeeper validation policies. The policies are defined as ConstraintTemplates and Constraints. ConstraintTemplates can be applied directly to a cluster and then Constraints can be applied to customize a policy to fit the specific platform needs.

In the following, we provide an example where we want to limit the set of allowed image repositories to the ones owned by the organization. This is a necessary step to ensure the service images deployed on the cluster do not come from untrusted sources, thus improving the protection of the organization operations and data. Most organization should enforce this constraint, as such, the OPA Gatekeeper library [opa23] already offers its corresponding ConstraintTemplate (see Listing 4).

To declare a constraint template it is necessary to declare a YAML document specifying: the Gatekeeper template API, the kind of the Gatekeeper resource being created (i.e., ConstraintTemplate), a few metadata information, and the custom resource definition together with its targets. With the custom resource definition the YAML document declares the schema for the parameters field of a K8sAllowedRepos constraint. Then, in the target section where the Gatekeeper admission controller is selected, the Rego policy is specified. First, it gathers container information from the Pod deployment, then it checks whether the image of the container starts with one of the repositories specified in the list of allowed repository. Finally, if there is no allowed repository matching the container image, a violation is raised with an error message.

With this constraint template applied to the Kubernetes cluster, the cluster is ready to receive the specific 'K8sAllowedRepos' constraint that specifies the list of container image repositories available for use by the operation team. Listing 5 shows an example 'K8sAllowedRepos' constraint where *Pods* belonging to the *default* namespace are restricted to use container images downloaded from the *openpolicyagent* registry.

As a result of applying the above constraint, Gatekeeper successfully permits the deployment of pods images from the *openpolicyagent* registry (Listing 6), but denies the deployment of pods from any other registries (Listing 7).

## 2.5 Integration of Open Policy Agent

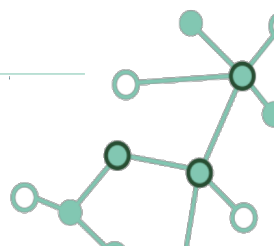
---

OPA exposes domain-agnostic APIs that services can call to manage and enforce policies. In this section we highlight the ways to integrate an application, service, or tool with OPA.

### 2.5.1 At application level

---

OPA supports different ways to evaluate policies. The REST API returns decisions as JSON over HTTP. The Go API (GoDoc) returns decisions as simple Go types (bool, string, map[string]interface, etc.). WebAssembly (also called *Wasm*) compiles Rego policies into Wasm instructions so they can be embedded and evaluated by any WebAssembly runtime. Custom compilers and evaluators may

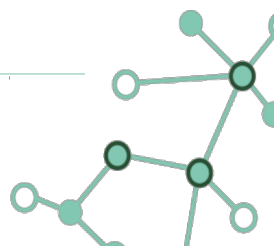


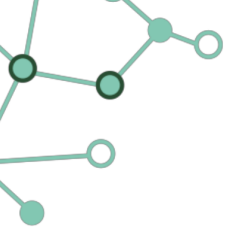


Listing 4: Snippet of the Gatekeeper ConstraintTemplate to restrict provenance of container images

```
apiVersion: templates.gatekeeper.sh/v1
kind: ConstraintTemplate
metadata:
  name: k8sallowedrepos
spec:
  crd:
    spec:
      names:
        kind: K8sAllowedRepos
      validation:
        # Schema for the 'parameters' field
        openAPIV3Schema:
          type: object
          properties:
            repos:
              description: The list of prefixes a container image is allowed to have.
              type: array
              items:
                type: string
  targets:
    - target: admission.k8s.gatekeeper.sh
      rego: |
        package k8sallowedrepos

        violation[{"msg": msg}] {
          container := input.review.object.spec.containers[_]
          satisfied := [good | repo = input.parameters.repos[_]
                       good = startswith(container.image, repo)]
          not any(satisfied)
          msg := sprintf("container <%v> has an invalid image repo <%v>",
                        [container.name, container.image])
        }
```





Listing 5: Gatekeeper Constraint restricting container images origin to the Open Policy Agent registry

```
apiVersion: constraints.gatekeeper.sh/v1beta1
kind: K8sAllowedRepos
metadata:
  name: repo-is-openpolicyagent
spec:
  match:
    kinds:
      - apiGroups: [""]
        kinds: ["Pod"]
    namespaces:
      - "default"
  parameters:
    repos:
      - "openpolicyagent/"
```

Listing 6: Example of allowed pod deployment

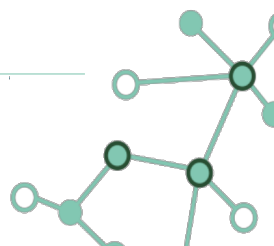
```
apiVersion: v1
kind: Pod
metadata:
  name: opa-allowed
spec:
  containers:
    - name: opa
      image: openpolicyagent/opa:0.9.2
      args:
        - "run"
        - "--server"
        - "--addr=localhost:8080"
      resources:
        limits:
          cpu: "100m"
          memory: "30Mi"
```

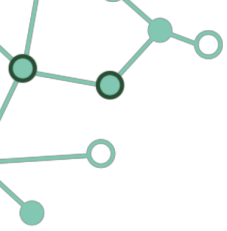
Listing 7: Example of denied pod deployment

```
apiVersion: v1
kind: Pod
metadata:
  name: nginx-disallowed
spec:
  containers:
    - name: nginx
      image: nginx
      resources:
        limits:
          cpu: "100m"
          memory: "30Mi"
```

be written to parse evaluation plans in the low-level Intermediate Representation format, which can be emitted by the OPA build command. The SDK provides high-level APIs for obtaining the output of query evaluation as simple Go types (bool, string, map[string]interface, etc.).

The different integration choices are summarized in Table 2, and a more detailed explanation follows. At the time of writing the most common way to integrate OPA with application and services is by using its REST API. OPA is most often deployed either as a sidecar or less commonly as an external service. Operationally, this makes it easy to upgrade OPA and to configure it to use its management services (bundles, status, decision logs, etc.). In Kubernetes, as well as any other sidecar-aware environment, deploying OPA this way automatically provides monitoring and logging.





| Dimension  | REST API        | Go Lib                     | Wasm                       |
|------------|-----------------|----------------------------|----------------------------|
| Evaluation | Fast            | Faster                     | Fastest                    |
| Language   | Any             | Only Go                    | Any with Wasm              |
| Operations | Update just OPA | Update entire service      | Update service rarely      |
| security   | Must secure API | Enable only what is needed | Enable only what is needed |

Table 2: Comparison of the different integration options of OPA

On the contrary, great care must be taken to secure OPA's configuration and APIs.

Integrating OPA via the Go API only works for software written in Go. Moreover, updates to OPA require re-vendoring and re-deploying the software. On the bright side, the evaluation has less overhead than the REST API because all the communication happens in the same operating system process. All of the management functionality (bundles, decision logs, etc.) must be either enabled or implemented. Security concerns are limited to those management features that are enabled or implemented.

Finally, Wasm policies are embeddable in any programming language that has a Wasm runtime. Evaluation has less overhead than the REST API (because it is evaluated in the same operating system process) and should outperform the Go API (because the policies have been compiled to a lower-level instruction set). Each programming language will need its own SDKs that implement the management functionality and the evaluation interface. Typically new OPA language features will not require updating the service since neither the Wasm runtime nor the SDKs will be impacted. However, updating the SDKs will require re-deploying the service. Security is analogous to the Go API integration: it is mainly the management functionality that presents security risks.

In this section, we highlight the main method to integrate OPA inside of application and services. For a deep-dive into every integration method see the Integrating OPA documentation [Sty23a].

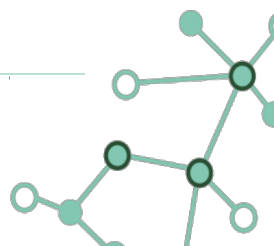
## REST API

To integrate with OPA outside of Go, the OPA authors recommend to deploy OPA as a host-level daemon or sidecar container. So, whenever applications or services need to make policy decisions they can query OPA locally via HTTP. The decision to run OPA locally on the same host as the application or service querying it helps ensure policy decisions are fast and highly-available.

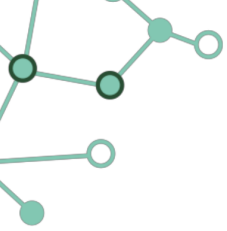
In order to query OPA for named policy decision, the Data API must be used. This API is exposed at `/v1/data/<path>`, and by sending HTTP requests with the POST method it is possible to specify the input JSON document (in case it is needed for policy evaluation). The `<path>` in the HTTP request identifies the rule we want to evaluate in order to get the policy decision. It is possible to request specific decisions by querying for `<package path>/<rule name>`. For example, in Listing 8 there is a single decision, `example/authz/allow`.

To request its evaluation and produce the related access decision the user should send an HTTP request as shown in Listing 9.

OPA returns an HTTP 200 response code if the policy was evaluated successfully. Non-HTTP 200 response codes when there are configuration or runtime errors. The policy decision is contained in the "result" key of the response message body. For example, the above request makes OPA return







Listing 8: Example Rego policy

```
package example.authz

import future.keywords.if
import future.keywords.in

default allow := false

allow if {
  input.method == "GET"
  input.path == ["salary", input.subject.user]
}
```

Listing 9: Example of evaluation request sent to the OPA Data API

```
POST /v1/data/example/authz/allow
Content-Type: application/json

{
  "input": {
    "method": "GET",
    "path": ["salary", "bob"],
    "subject": {
      "user": "bob"
    }
  }
}
```

the response in Listing 10.

Note that if the requested policy decision is undefined OPA returns an HTTP 200 response without the “result” key.

Listing 10: Example of OPA Data API response

```
200 OK
Content-Type: application/json

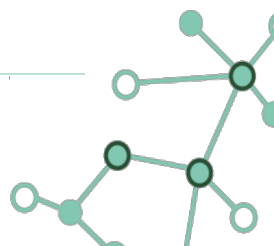
{
  "result": true
}
```

## 2.5.2 At platform level

---

### Envoy

Transparent integration of OPA with existing applications at the network layer is very important as it significantly reduces the friction in the introduction of the new security measure. To this end, OPA offers integration with the Envoy network proxy. Envoy is a L7 proxy designed for large modern service oriented architectures. As part of its feature set, it supports an external authorization filter which calls an authorization service to check if the incoming request is authorized or not (see Figure 11). This feature makes it possible to delegate authorization decisions to an external service and also makes the request context available to the service. The request context contains information such as the source of a network activity, the destination of a network activity, and the network request (e.g., http request). All this information can be used by the external service to make an informed decision about the fate of the incoming request received by Envoy. The OPA ecosystem comes with a plugin



to enforce OPA policies with Envoy. Developers can use this version of OPA to enforce fine-grained, context-aware access control policies with Envoy without modifying microservice implementations.

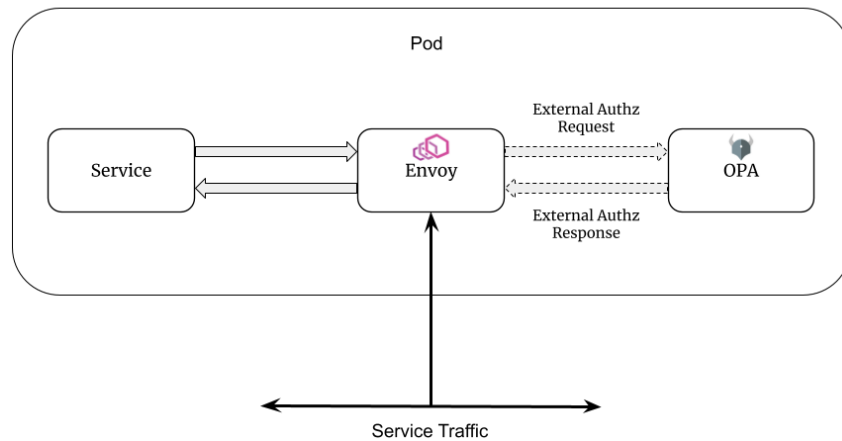


Figure 11: Architecture of the petprofilesv1 service [Sty23d]

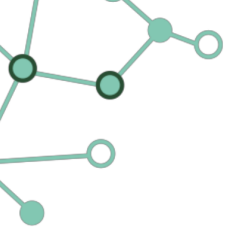
As shown in Figure 11, in addition to the Envoy sidecar, the application pods have to include a sidecar with the OPA plugin for Envoy. When Envoy receives API requests destined for the microservice, it checks with OPA to decide if the request should be allowed. Note that evaluating policies locally with Envoy is preferable because it avoids introducing a network hop (which has implications on performance and availability) in order to perform the authorization check.

## Kubernetes

Kubernetes allows decoupling policy decisions from the inner workings of the API Server by means of admission controller webhooks, which are executed whenever a resource is created, updated or deleted [Aut23].

Admission webhooks are HTTP callbacks that receive admission requests and either validate or mutate the request depending on whether the admission webhook is a validating admission webhook or a mutating admission webhook. Mutating admission webhooks are invoked first, and can modify objects sent to the API server to enforce custom defaults. After all object modifications are complete, and after the incoming object is validated by the API server, validating admission webhooks are invoked and can reject requests to enforce custom policies.

Gatekeeper, the admission controller that integrates Open Policy Agent, is a validating and mutating webhook that enforces policies using Kubernetes custom resource definitions. By default, it works as a validating webhook, however the webhook configuration can be tuned to meet specific needs if they differ from the defaults. Two particularly important configuration options deserve special mention: timeouts and failure policy. Timeouts permit to configure how long the API server will wait for a response from the admission webhook before it considers the request to have failed. Failure policy controls what happens when a webhook fails. Common failure scenarios include timeouts, a 5xx error from the server or the webhook being unavailable. There is the option to ignore errors, allowing the request through, or failing and rejecting the request. This results in a direct tradeoff

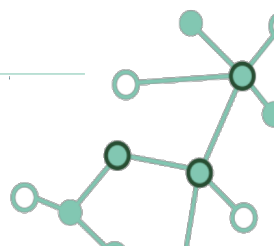


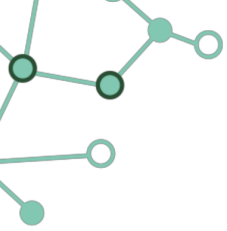
between availability and enforcement. Currently, Gatekeeper is defaulting to using *Ignore* for the constraint requests, which means constraints will not be enforced at admission time if the webhook is down or otherwise inaccessible.

In addition to the admission scenario, Gatekeeper's audit functionality allows administrators to see what resources are currently violating any given policy by periodically evaluating existing resources against constraints to detect pre-existing misconfigurations. There are three ways to gather audit results, depending on the level of detail needed. Prometheus metrics provide an aggregated look at the number of violations. Constraint status lists violations of a constraint from the most recent audit run with a limit on the maximum number of individual violations reported. Audit logs containing information on the violated constraint, violating resource, and violation message together with an `audit_id` to uniquely identify a given audit run, which allows indexing of historical audits.

From an architectural perspective, Gatekeeper is flexible in how it can be deployed. If desired, core pieces of functionality can be broken out to be run in different pods. This allows Gatekeeper to accommodate needs like running in a monolithic pod in order to avoid overhead, or running each operation in a separate pod to make scaling individual operations easier and to limit the impact of operational issues with any one operation (e.g. if audit is running in its own pod, audit running out of memory will not affect the validation webhook).

The use of Gatekeeper as an admission controller enables the use of OPA at different level in the software stack. Specifically, it secures the cluster to meet governance and legal requirements, and helps ensure adherence to best practices and institutional conventions in an automated way. Indeed, attempting to ensure compliance manually would be error-prone and frustrating. Automating policy enforcement ensures consistency, lowers development latency through immediate feedback, and helps with agility by allowing developers to operate independently without sacrificing compliance. These are all key motivations that led to the identification of Gatekeeper as a candidate for integration into the GLACIATION platform.





## 3 Data Wrapping

---

This chapter reports on the design of novel techniques to protect data and computations in the GLACIATION platform. Compared to data sanitization, where there is an irreversible loss of information in the protected data, data wrapping considers the use of techniques that do not reduce the information content.

An important role in data wrapping is assumed by cryptographic functions, which are available in common libraries and offer nowadays good performance. The application of a simple layer of encryption can then be obtained by directly following the prescriptions of the security policy. The work in Task 4.2 focuses on the identification of novel solutions for the protection of data. A first line of investigation considered in Task 4.2 is associated with the realization of efficient solutions for checking the integrity of large scale computations, which will be briefly presented in the next section. A number of options have been investigated to identify advanced approaches for data wrapping. The core of this chapter is dedicated to the presentation of the use of the All-Or-Nothing-Transform to obtain improved protection and the support for robust revocation.

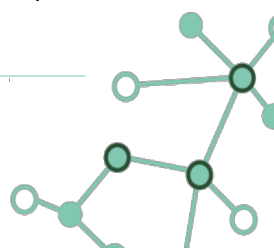
### 3.1 Integrity support for large-scale computations

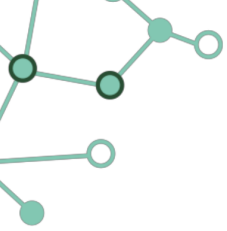
---

Distributed computing supports large scale and data-intensive computations with the cooperation of a multitude of parties, each responsible for a portion of the workload. Such parties are often not fully reliable and may return incorrect results. The proposal reported in [DFJ<sup>+</sup>23] addresses the problem of assessing the integrity of the computation results, with a comprehensive characterization of two techniques, sentinels and twins, evaluating their effectiveness and synergy.

Sentinels are pre-computed tasks whose result is known apriori, and enable checking returned results against a ground truth. Twins are replicated tasks assigned to different workers, and enable cross-checking returned results for a same task. Many questions arise in the design of a concrete integrity assessment strategy and several parameters have a critical impact on the overall protection. The model developed in [DFJ<sup>+</sup>23] enables to tune the integrity controls so to achieve best effectiveness. The model can be applied to a variety of scenarios, compatible with GLACIATION requirements, and can find extensive application in several domains.

The verification of the integrity of the results of computations outsourced to external parties is well known and recognized by the research and industrial communities. A promising approach to assess integrity in contexts where computations are not fixed and predefined (and therefore authenticated data structures providing deterministic integrity guarantees are difficult to use) relies on probabilistic techniques. These can always be applied when each portion of the problem assigned to a worker can be structured as a collection of jobs, each producing a result. It is then possible to inject jobs against which the behavior of workers is controlled. Most common probabilistic techniques either (a) insert jobs whose result is known a-priori, alerting for violations whenever results are different from the known one, or (b) replicate jobs to multiple workers, alerting for violations whenever results from different workers in response to the same replicated job differ. While the two techniques





are known and well recognized, the problem of their targeted generation and combination, so to provide best effectiveness for integrity guarantees, is still an open issue.

We addressed this problem and proposed a model to reason on the combined use of pre-computed and replicated jobs (sentinels and twins, respectively), so to provide best effectiveness. We frame our work in the context of a data classification problem, which allows us to capture a variety of scenarios, and investigate different issues that naturally arise in the application of such controls. Our investigation produces an improved characterization of each technique and provides a response to many questions:

- What are the aspects that have an impact on the effectiveness of sentinels and twins?
- How should sentinels be generated and twins be distributed among workers so to provide best integrity guarantees?
- How many replicas should be used to get the best effectiveness?
- When are twins more effective than sentinels (or vice versa)?
- Given an application domain, what is the combination of sentinels and twins able to provide the best protection?

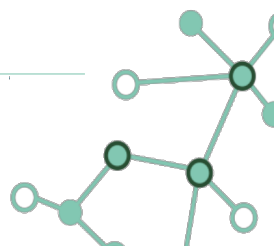
The results of the investigation show the effectiveness of the two techniques when carefully combined based on our findings. Our work represents then a reference for the application of probabilistic techniques, supporting the realization of efficient integrity assessment solutions for many domains.

An important result obtained in this investigation is the identification of a clear criterion that can be used to distinguish when the use of sentinels is preferable to the use of twins. The model starts from the recognition that a crucial parameter is  $P_{max}$ , which represents the probability of the most common result returned by the computations. It is then determined that sentinels are preferable to twins when, with  $c$  representing the number of possible computation results:

$$P_{max} > \frac{1}{2} \cdot \left(1 + \frac{1}{c}\right)$$

Our analysis tells us that, for each scenario, either sentinels or twins should be used as control jobs, depending on the value of  $P_{max}$  characterizing the scenario. When the number of possible results  $c$  is equal to 2, which is the lowest possible value, twins (sentinels, resp.) should be used if  $P_{max}$  is lower (higher, resp.) than or equal to 0.75. As the number of possible results grows, the value of  $P_{max}$  when sentinels are more effective than twins decreases, reaching 0.50 as the number of possible results becomes very large (and hence  $\frac{1}{c}$  negligible). The formula above also tells us that when  $P_{max}$  is not higher than 0.50, twins are always more effective than sentinels.

The analysis also shows how sentinels and twins exhibit a complementary behavior and respond to different threats. When twins prove to be the most efficient solution, it is still convenient to use a small number of sentinels to be protected against scenarios where coordinated misbehavior in the task processors can make twins ineffective.





### 3.2 Access revocation for encrypted resources

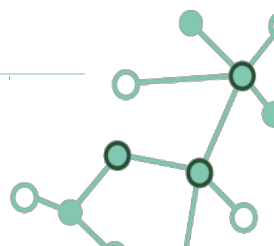
---

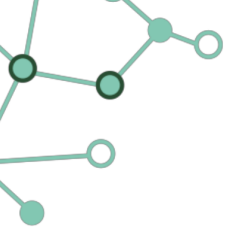
Cloud technologies provide companies a reliable and economic alternative for the implementation of core business services. Example of this are web, accounting, email and file sharing services. In all these cases a critical task to perform is the enforcement of access control rules. Briefly, every company is interested in restricting which entity has access to which resource. A viable approach to retain control is through encryption: resources are encrypted before they are uploaded to the cloud, and only legitimate users are given the key to access. This approach brings several advantages. First, it provides protection from the cloud provider itself. In the classical *honest-but-curious* scenario [DFLS22, XMW<sup>+</sup>21, SBM20] the provider is considered trustworthy, but not allowed to access resources in their plaintext representation, and not necessarily authorized to enforce access control. Second, the use of encryption is mostly computationally inexpensive, hence it can be applied or removed with little impact on performance even in environments with demanding requirements. Finally, encryption removes the need for a third party to enforce the policy, since only the holders of the key can recover a resource from its encrypted representation.

While granting access to an encrypted resource is a relatively easy task (i.e., the authorized user is simply given access to the decryption key), revoking access to a protected resource is a delicate operation. One may for example block access to the key, however, this approach cannot be considered valid, because it fails shortly when a user stores locally a copy of the key. Alternatively, the data owner (i.e., the owner of the protected resource) may re-upload on the cloud a new version of the resource encrypted with a fresh key, which is re-distributed to the list of users who still hold authorization. While this approach ensures protection, it still requires to download, re-open and re-encrypt the resource and redistribute the access key every time a policy update is performed. This process may be associated with excessive overhead, which may not be compatible with modern large scale applications.

In this chapter we present a novel approach to provide efficient access revocation for large resources uploaded to the cloud. The approach we propose only requires to encrypt a small portion of the resource every time a policy update is performed, and is resilient against the threat of a user storing local copies of the access keys previously distributed. The fundamental concept behind our approach is the *All-Or-Nothing Transform* (AONT) [Riv97], which prevents the reconstruction of a plaintext resource even when a small portion of it is unavailable (brute force attacks are still possible, but the success probability can be made negligible tuning the parameters). Unfortunately, classic AONT constructions are fragile in a scenario where the user knows the key, therefore we propose a new AONT construction that leverages *mixing*, which ensures that each bit of an encrypted resource depends on all the bits of a plaintext resource (and viceversa). This operation guarantees that, even when a user knows the decryption key, access to the plaintext resource (or any portion of it) is prevented unless a brute-force attack is performed.

The mixing operation ensures complete interdependence among the bits of a resource, however, it is associated with an overhead that directly depends on its size. To improve efficiency with large resources, while at the same time support fine-grained access (thus enabling access to portions of them), our solution first divides large resources into equally large chunks called *macro-blocks*, then performs mixing on each of them. The only consequence of this design is that, whenever an





access revocation is performed, a small portion of each macro-block must be updated. To facilitate this operation we apply a linear transformation, called *slicing*, which transforms macro-blocks into fragments, and ensures that: i) every fragment includes bits from each macro-block, and ii) the lack of single fragment is sufficient to prevent the decryption of the whole resource. As a matter of fact, after the resource has been encrypted and separated into fragments with our approach, only a single fragment must be re-encrypted to enforce access revocation. There is no need for the provider to implement any access control logic, as the sole storage functionality is required.

In this chapter, we extend the algorithm leveraging the Optimal Asymmetric Encryption Padding (OAEP). We also discuss and implement a variant of the approach that relies on a recursive OAEP construction. Moreover, the chapter includes a security analysis that discusses the protection given by our approach against an attacker leveraging erasure codes. Finally, the experimental evaluation analyzes the performance associated with our approach, highlighting the throughput when a sequence of revocation operations are carried out. All this work has also been reported in a paper which is going to be published in IEEE Transactions on Dependable and Secure Computing [BDF<sup>+</sup>23].

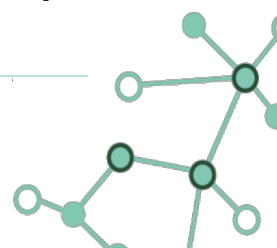
The remainder of the chapter is organized as follows. In Section 3.3 we discuss the state of the art. Section 3.4 introduces the basic concepts of mixing and slicing. Section 3.5 describes the Mix&Slice construction, discussing two mixing strategies (the first based on AES and the second on an extended 3-round OAEP). Section 3.6 explains how our construction can be leveraged to implement access revocation. Section 3.7 presents the security analysis; two scenarios are considered: i) the attacker maintains the copies of some resource fragments locally, and ii) the attacker leverages erasure codes. Lastly, Section 3.8 illustrates our implementation and the experimental evaluation, discussing the applicability of our approach.

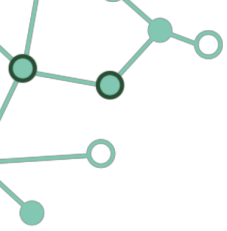
### 3.3 State of the art

---

The first research proposal that envisioned the reconstruction of a plaintext resource only when the full ciphertext is available was published by Rivest in 1997 [Riv97]. The idea of the author was that, the removal of  $n$  bits from a resource, should require an attacker to perform  $2^n$  attempts to be successful in the reconstruction. This is the origin of the *All-Or-Nothing Transform* (AONT) concept.

An AONT algorithm does not provide confidentiality by itself, but it can be complemented with the use of encryption to produce an *AONT scheme*. Rivest [Riv97] proposed to append to the ciphertext a block containing the random encryption key  $k$  XOR-ed with a hash of all the previous encrypted message blocks. The consequence is that the unavailability of a single block prevents the reconstruction of the decryption key, hence the ability to recover the plaintext resource. The problem associated with this approach is that it does not offer protection in the access revocation scenario. Indeed, a user that was previously authorized to access a resource, and that has maintained a local copy of the decryption key, can easily reconstruct the resource (or parts of it) without processing the whole ciphertext block sequence. The reason is that the key can be seen as a compact digest of the resource, and this information can be saved by an attacker using little storage. This limitation is also shared with other techniques that have been proposed subsequently [KSLC17, QKL<sup>+</sup>19], in





fact, these techniques assume that the decryption key is never shared with a previously authorized user.

Alternative approaches to enforce access control in the cloud leveraging encryption have been proposed. A first line of research is represented by *selective encryption* (e.g., [DFJ<sup>+</sup>07, DFJ<sup>+</sup>10a, HKD15]). In this setting, each resource is encrypted with a dedicated key that only an authorized user can derive. Policy updates are then managed by the data owner or the server through updatable encryption [BEKS20] or over-encryption [DFJ<sup>+</sup>07, DFJ<sup>+</sup>10a]. Updatable encryption schemes directly modify the key used for encryption by sending to the server a compact update token. Over-encryption instead, delegates to the server the enforcement of an additional encryption layer that is dedicated to supporting policy updates. Both these approaches can support policy updates without requiring to download, re-encrypt and re-upload a resource, however, they require to delegate the server part of the authorization procedure. The advantage of the technique we propose is that it can be used removing this requirement. This is shared with other recent works such as *Knob* [CRR20], which similarly to Mix&Slice, proposes to re-encrypt a small portion of a resource. Knob construction is more efficient compared to Mix&Slice (higher throughput), however, it does not provide protection against users previously authorized to access a resource.

An alternative line of research compared to selective encryption is *Attribute-Based Encryption (ABE)*. ABE approaches (e.g., [XNH<sup>+</sup>22, YWRL10, GSB<sup>+</sup>22, ZDX<sup>+</sup>21, GPSW06, HN11]) enable access control by imposing that only authorized users can perform the key derivation procedure based on a set of attributes (e.g., role, age). The shortcoming in this case is that the ABE schemes rely on asymmetric encryption, hence they can suffer from poor performance when many attributes are used.

Related to this research topic are secure deletion techniques [CHHS13, DW10] employed by commercial storage devices. These solutions (e.g., [TCG15]) leverage the fact that deleting a compact information from a disk page can effectively invalidate the derivation of the decryption key associated with large resources. However, these techniques are not applicable to our scenario, since they would require the server to store the decryption key (or intermediate representations).

Another related research line is the problem of protecting the confidentiality of encrypted data stored in a distributed environment in case of key exposure. The solutions presented in [KRM20, BDF<sup>+</sup>20, BFG<sup>+</sup>21] leverage novel secret sharing schemes and data fragmentation to prevent resource reconstruction by making fragments unavailable.

## 3.4 Mix & Slice

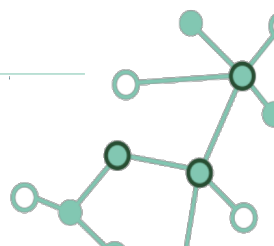
---

In the first part of this section we illustrate the concepts of block, mini-block, and macro-block. Then, we describe in detail the mixing (Section 3.4.2) and slicing (Section 3.4.3) procedures.

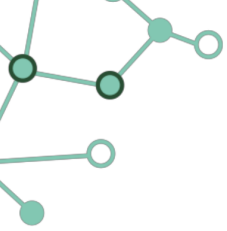
### 3.4.1 Blocks, mini-blocks, and macro-blocks

---

The basic construction that allows us to build the AONT encryption scheme is mixing, which transforms input data into output data so that all the bits in the output depend on every single bit in the input. There are two ways to implement this construction: i) leveraging a symmetric block cipher, and







ii) using a cryptographic hash function. Modern symmetric block ciphers guarantee the dependency of every bit in the output from every bit in the input. Moreover, they guarantee that it is not possible for an attacker to revert their application without performing a brute-force attack. For instance, functions like AES guarantee that the absence of  $o$  bits from the ciphertext (i.e., the output), and of  $i$  bits from the plaintext (i.e., input), requires an attacker to perform a brute-force attack to generate and verify the  $2^{\min(i,o)}$  possible configurations [ABM14], even when the encryption key  $k$  is known. On the other hand, cryptographic hash functions are non invertible transformations that guarantee that every bit in the input has an effect on every bit in the output.

Naturally, the protection against a brute-force attack depends on the number of combinations to be verified, hence the number of missing bits (e.g., when  $x$  bits are missing,  $2^x$  combinations must be tested). The number of missing bits is the *security parameter* in our proposal. To further understand it, we introduce the concepts of *block*, *mini-block* and *macro-block*.

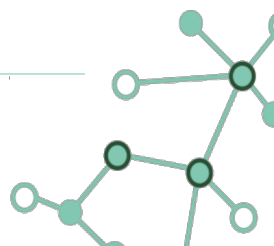
- *Block*: the sequence of bits input to a block or a hash function (the classic concept of block).
- *Mini-block*: a contiguous portion of bits, of a given length, in a block. It represents the *atomic unit* of protection (i.e., deletion operates at the level of mini-block).
- *Macro-block*: The sequence of blocks. Resources can have arbitrary length, macro-blocks allow extending the protection to sequences of bits longer than the block. In our construction *mixing* is applied at the macro-block level.

With our approach, arbitrary sizes can be set for the length of mini-block, block and macro-block. We only require that the length of the mini-block must be a divisor of the length of the block, and the length of the macro-block to be a product of the size of a mini-block. In the examples we will illustrate our approach assuming 128-bit blocks (that are used by AES) and 32-bit mini-blocks (hence 4 mini-blocks per block). We will use  $m_{size}$ ,  $b_{size}$ ,  $M_{size}$  to represent the size (in bits) of mini-blocks, blocks, and macro-blocks, respectively. We will use  $b_j[i]$  ( $M_j[i]$ , resp.) to represent the  $i$ -th mini-block in a block  $b_j$  (macro-block  $M_j$ , resp.). Lastly, the notation  $[i]$  to denote the  $i$ -th mini-block in a generic bit sequence, and  $[[j]]$  to denote the  $j$ -th block. The additional subscript associated with a mini-block/block denotes the mixing round that produced it (e.g.,  $[i]_1$  for mini-block  $i$  at round 1).

### 3.4.2 Mixing

---

As explained in the previous section, mixing makes sure that each bit in the output is dependent on every bit in the input, for each macro-block. To illustrate why this property is important, we consider the protection applied by a modern block cipher like AES on a resource composed of 16 32-bit mini-blocks  $[0], \dots, [15]$ . Clearly, since AES operates on 128-bit blocks (or 4 mini-blocks), AES must be applied four times to encrypt the resource. This means mixing is applied to the four 128-bit sequences  $[0] \dots [3]$ ,  $[4] \dots [7]$ ,  $[8] \dots [11]$ , and  $[12] \dots [15]$ , or rather that the deletion of a single mini-block will prevent the reconstruction of only the block that includes it. The other three will be directly available to an attacker that knows the encryption key  $k$ . This level of protection is not adequate to perform access revocation in the cloud scenario: first, to securely revoke access to a large resource at least a mini-block from each block must be deleted, and second, removing a single mini-block with



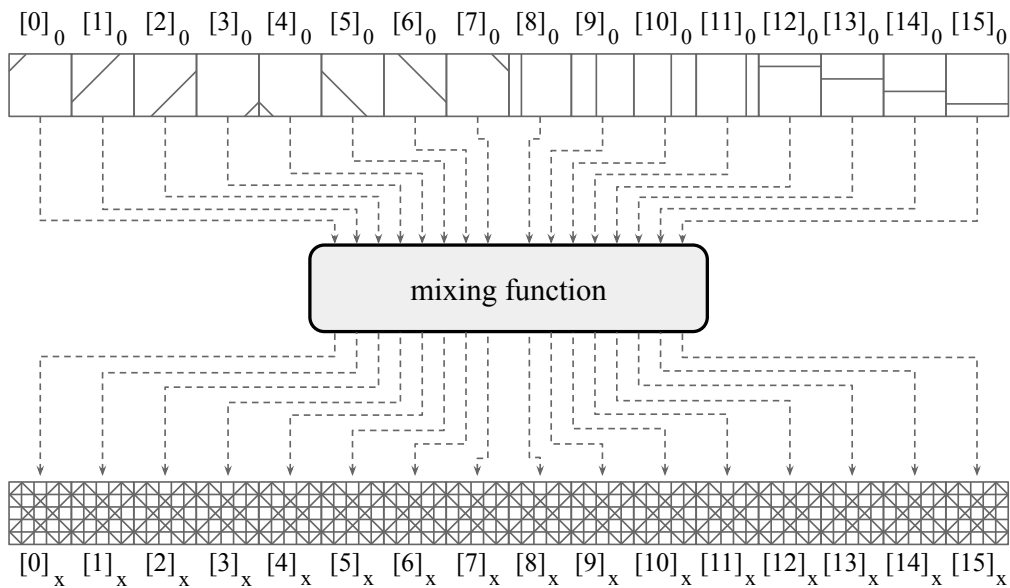


Figure 12: Visualization of mixing for a macro-block with 16 mini-blocks  $[0] \dots [15]$

32-bit length does not provide enough protection against the brute-force attack. In order to protect large resources we must extend mixing to sequences of bits larger than the traditional block, as shown in Figure 12. In detail, the design of an adequate algorithm must satisfy the following three properties.

- *Complete mixing*: every bit in the input macro-block must cryptographically affect every bit in the output.
- *Arbitrary macro-block size*: mixing must be applied on arbitrarily large macro-blocks.
- *No-shrinking effect*: mixing must not reduce the size of a macro-block.

As previously mentioned, complete mixing ensures no reconstruction of the whole macro-block when a small portion of it is unavailable, excluding the brute-force scenario. The second property permits to protect resources of arbitrary size, implementing an effective access revocation policy, as it will be demonstrated in the experimental evaluation (Section 3.8). The third property ensures that malicious users cannot store locally a small portion of the protected resource to easily perform a reconstruction attack when their access capability is revoked. This is an important aspect, and it is achieved through multiple encryption rounds (more details in the following).

Due to the previous considerations, large macro-blocks ensure strong security guarantees. However, there are also cases where smaller macro-blocks may be desired. The first case is represented by huge resources, which may take considerable time to download. Having a single macro-block with size equal to the one of the resource implies downloading must be completed before decryption is performed. Also, an update to a small portion of the resource entails applying mixing entirely from scratch, resulting in longer update operations. Taking into consideration that the performance aspect is critical for many real applications, it is convenient to partition huge resources into macro-blocks.

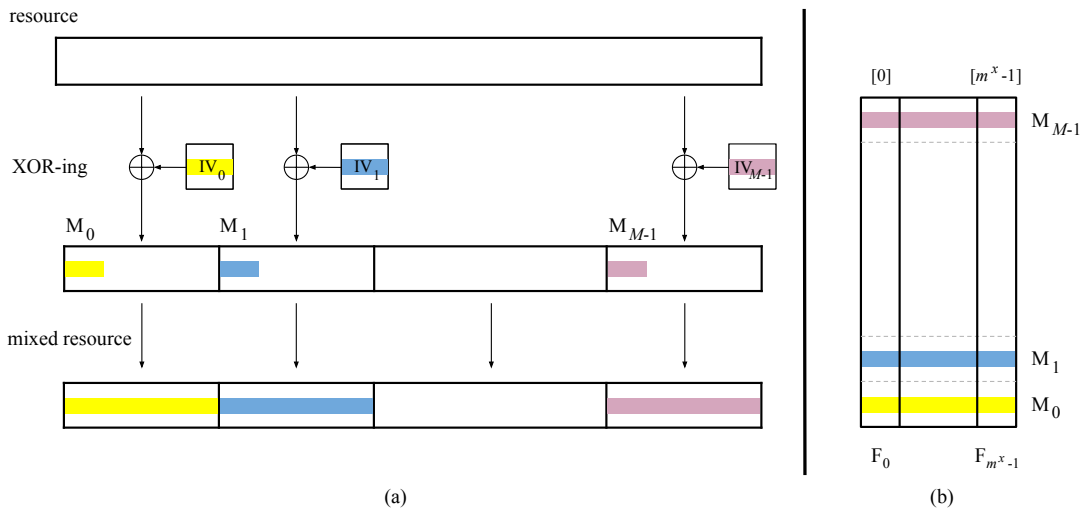


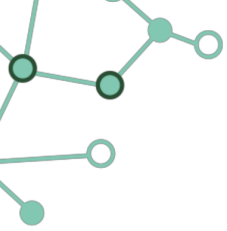
Figure 13: Mix&Slice: transforming a resource into fragments

Not only this choice permits to serve fine-grained retrieval requests (i.e., enabling authorized access to portions of the resource), it also permits to parallelize the AONT construction, greatly reducing the time taken for upload, download, and update (this aspect will be investigated in Section 3.8).

The realization of an efficient AONT encryption scheme would then entail a preliminary step to partition a huge resource into equally large macro-blocks, on which mixing is applied. To make sure the encrypted representation of each macro-block differs even in the case of sequences with equal partitions, it is sufficient to XOR every macro-block with a unique *initialization vector* (*IV*) before mixing is applied. Because of the complete mixing property, there is no need for the IV to be as long as the macro-block (every bit in the input affects all the bits in the output). To achieve indistinguishability between mixed macro-blocks, it is sufficient to generate a random IV for the first block, and then increment it by 1 for each of the subsequent macro-blocks in the resource, similarly to the CTR encryption mode [Dwo01]. The process is shown in Figure 13(a).

### 3.4.3 Slicing

As explained in the previous subsection, complete mixing operates at macro-block level. Hence, when a resource has been fragmented either to support fine-grained access or simply because it is too large, the removal of a single mini-block implies that only the macro-block it belongs to is protected. To revoke access to the whole resource, at least a mini-block from each macro-block must be removed, otherwise the attacker would be able to perform partial reconstruction. This observation leads to the introduction of *slicing*. Slicing is a process that transforms mixed macro-blocks into fragments so that: i) every fragment stores at least one mini-block for each macro-block, ii) every mini-block belongs to a single fragment (complete resource coverage and no repetitions). A simple linear transformation can be used to perform this operation (e.g., storing in the same fragment the mini-blocks that occur at the same position for different macro-blocks). This transformation is illustrated in Figure 13(b). The following definition formalizes the concepts of slicing and fragments.



**Mix&Slice**

```

1: cut R in M macro-blocks M0, ..., MM-1
2: apply padding to the last macro-block MM-1
3: IV := randomly choose an initialization vector
4: for i = 0, ..., M - 1 do
5:   Mi[[1]] := Mi[[1]] ⊕ IV
6:   Mix(Mi)
7:   IV := IV + 1
8: for j = 0, ..., f - 1 do
9:   Fj[i] := Mi[j]

```

/\* mix macro-blocks \*/  
/\* XOR the first block with the IV \*/  
/\* Mix M<sub>i</sub> with a chosen procedure \*/  
/\* update the IV for the next macro-block \*/  
/\* slicing \*/

Figure 14: Mix&Slice: from resource R to fragments

**Definition 3.4.1 (Slicing and fragments)** Let R be a resource and M<sub>0</sub>, ..., M<sub>M-1</sub> be its (individually mixed) macro-blocks, each composed of f mini-blocks. Slicing produces f fragments for R where F<sub>j</sub> = ⟨M<sub>0</sub>[j], ..., M<sub>M-1</sub>[j]⟩, with i = 0, ..., f - 1.

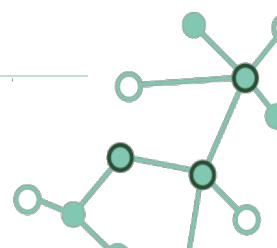
Figure 14 overviews the procedure used by Mix&Slice to encrypt a resource R. R is: i) cut into M macro-blocks, ii) padding is applied to the last macro-block, iii) a random initialization vector is selected, iv) the first block of each macro-block is XOR-ed with the updated IV, v) the macro-block is mixed with a chosen method (Section 3.5), and vi) the mixed macro-blocks are sliced into fragments.

### 3.5 Mixing

We present two strategies to mix macro-blocks. The first one (Section 3.5.1) is based on the AES block cipher and leverages the wide availability of its hardware implementation. The second one (Section 3.5.2) is based on the Optimal Asymmetric Encryption Padding (OAEP). The difference between the first and the second approach lies in the dimension of the mini-blocks. Indeed, with AES this can only be a divisor of 128 bits (i.e., the size of the AES block), while with OAEP larger mini-blocks are possible.

#### 3.5.1 AES Mixing

Our first proposal originates from the ability of AES to perform mixing at the block level. The idea is to propagate mixing to the macro-block level by encrypting with several rounds, and in between each round, permute the position of mini-blocks in the macro-block sequence following a specific pattern. Before describing in detail this process, let us briefly comment the simple example shown in Figure 15. The first row in the Figure shows a resource with 4 128-bit blocks, and 16 mini-blocks ([0], ..., [15]). At the first round AES is applied to each of the 4 blocks. This implies that each [i]<sub>1</sub> depends on every [j]<sub>0</sub> with (i div 4) = (j div 4), as illustrated with the pattern-coding. Before starting the second encryption round, we permute the mini-blocks produced at round 1 so to group mini-blocks at distance 4. In other words, encryption is applied to ⟨[0]<sub>1</sub>[4]<sub>1</sub>[8]<sub>1</sub>[12]<sub>1</sub>⟩, ⟨[1]<sub>1</sub>[5]<sub>1</sub>[9]<sub>1</sub>[13]<sub>1</sub>⟩, ⟨[2]<sub>1</sub>[6]<sub>1</sub>[10]<sub>1</sub>[14]<sub>1</sub>⟩, ⟨[3]<sub>1</sub>[7]<sub>1</sub>[11]<sub>1</sub>[15]<sub>1</sub>⟩. The result is a sequence of 16 mini-blocks



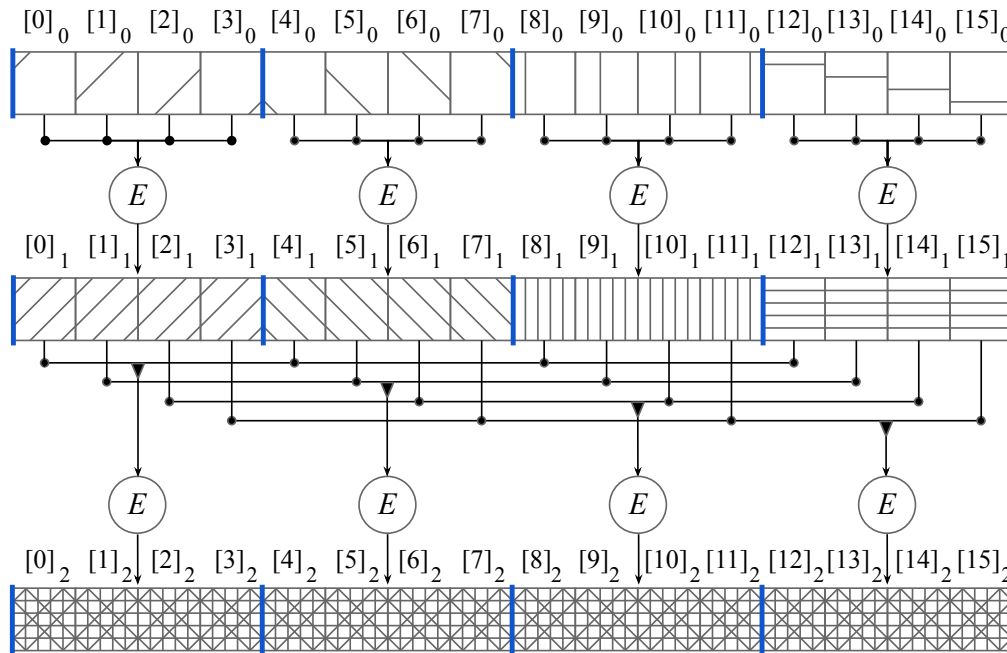


Figure 15: Visualization of AES mixing, 4 blocks and 16 mini-blocks

#### AES-Mix(M)

```

1: for  $i := 1, \dots, x$  do /* at each round  $i$  */
2:    $span := m^i$  /* compute the mixing span */
3:    $distance := m^{i-1}$  /* distance of the mini-blocks to be permuted */
4:   for  $j := 0, \dots, b-1$  do /* each  $j$  is an encryption */
/* form the  $j$ -th block picking mini-blocks at  $distance$  */
5:     let  $block$  be the concatenation of all mini-blocks  $[l]$ 
6:     s.t.  $(l \bmod distance) = j$  and  $(j \cdot m) \div span = l \div span$ 
7:      $[[j]]_i := E(k, block)$  /* encrypt the  $j$ -th block */

```

Figure 16: AES mixing strategy

in which every bit is cryptographically affected by each bit in the original resource. This aspect is highlighted with the pattern-coding of the Figure, showing a uniform texture. Clearly, the larger the resource the higher the number of encryption rounds to be performed.

Extending the approach to a general case,  $x$  encryption rounds are required to mix a macro-block with  $b = m^{x-1}$  blocks, where  $m$  is the number of mini-blocks in a block. This because mixing is applied to a number  $span$  of mini-blocks that is multiplied by  $m$  at every round. As a consequence, we set the macro-block to be composed of a number of mini-blocks that is the power of  $m$ . Note that  $m = 1$  is not a valid configuration. With reference to the running example, we used 32-bit mini-blocks, hence  $m = 4$ , and  $b = 4^{x-1}$ . The mixing procedure is also illustrated with the pseudocode reported in Figure 16. The progression of  $span$  is instead visualized in Figure 17, which shows the mixing of 64 mini-blocks. This time three encryption rounds are required to mix the macro-block. The number comes from the straightforward formula  $\log_m(m \cdot b)$ .

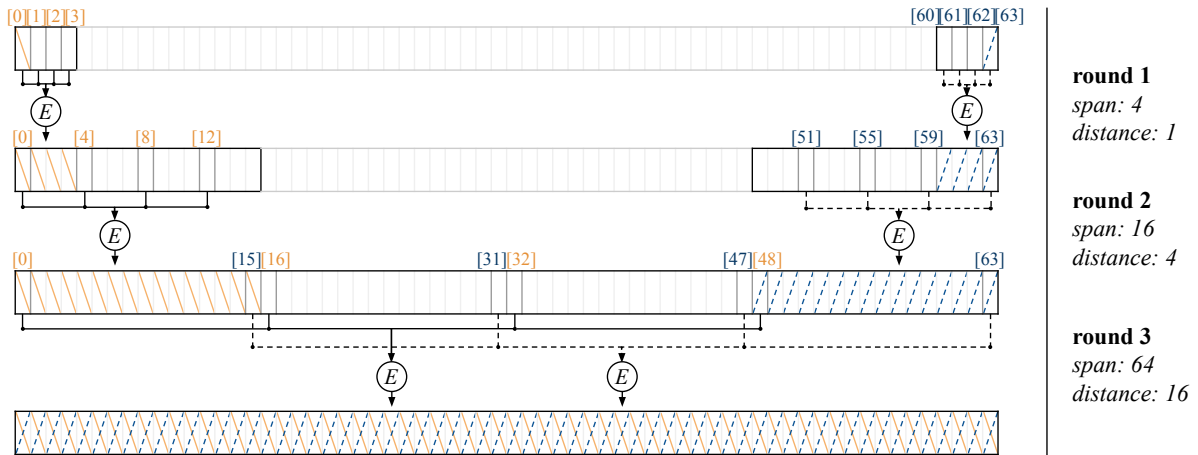


Figure 17: Propagation of the content of mini-block [0] after consecutive encryption rounds

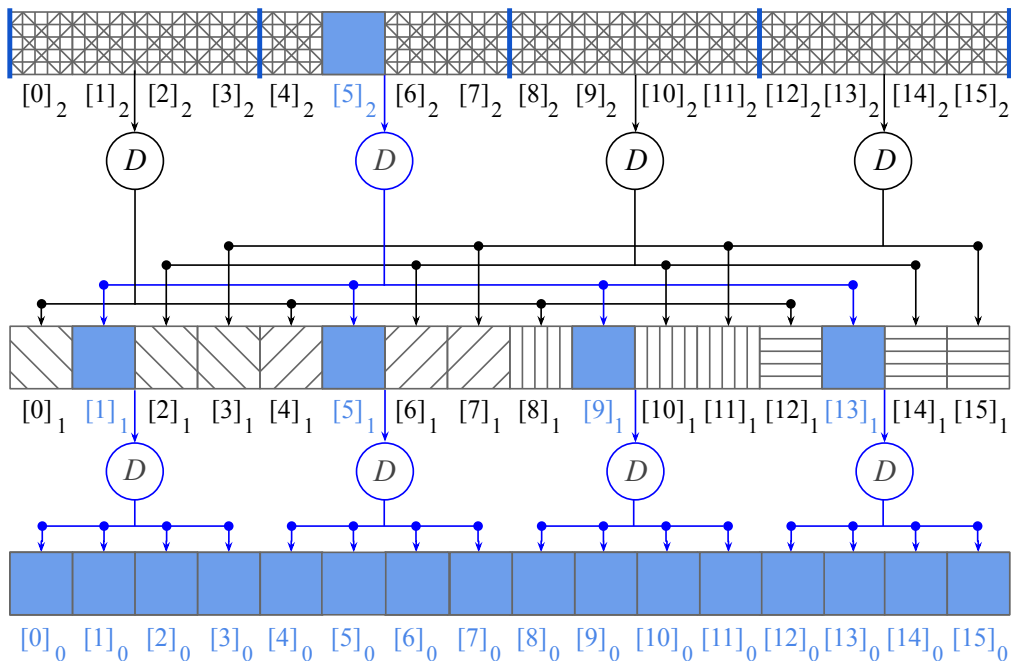


Figure 18: Decryption of a resource with 16 mini-blocks. The absence of the mini-block [5]<sub>2</sub> prevents the reconstruction of the whole resource

The proposed AES mixing strategy satisfies the properties illustrated in Section 3.4.2. Briefly, i) complete mixing is guaranteed due to the propagation of the span, and at each round, the uncertainty that stems from the lack of a mini-block requires the attacker to verify  $2^{m \cdot size}$  cases. To clarify this aspect, consider the example shown in Figure 18, in which the mini-block [5]<sub>2</sub> is removed from a macro-block long 16 mini-blocks. Since  $m=4$ , from the first unmixing round it is clear that the attacker must fall back to brute force attack to reconstruct every single block in the resource (i.e., the grey color propagates to all the blocks). Then, ii) mixing naturally applies to macro-blocks of

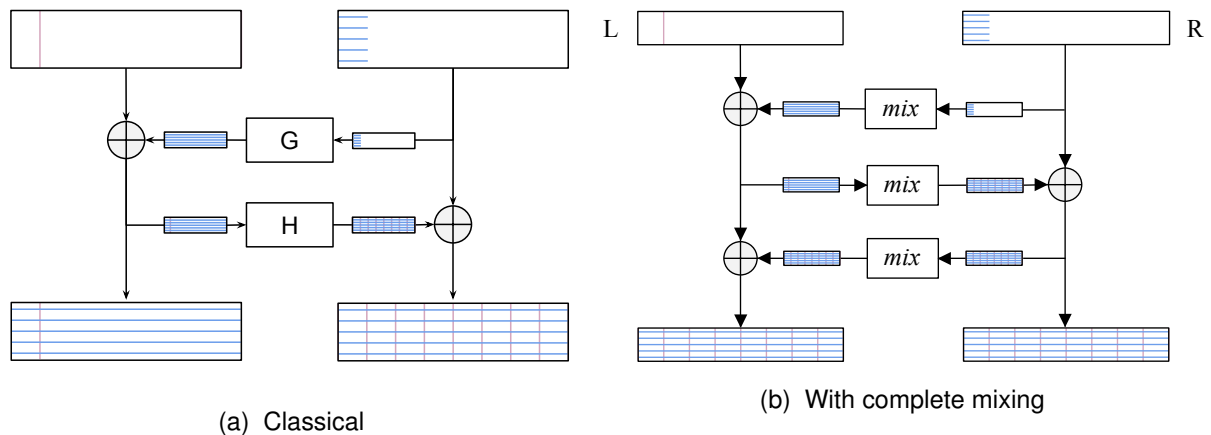


Figure 19: OAEP structures

*arbitrary size*, and in case the length of the macro-block is not a power of the number of mini-blocks in the block a standard padding technique can be achieved. Finally, iii) AES mixing also guarantees *no-shrinking effect*, as the size of the block remains invaried after encryption is performed.

### 3.5.2 OAEP Mixing

The second method we propose to enforce complete mixing on the macro-block leverages the Optimal Asymmetric Encryption Padding (OAEP) [BR94, Boy99]. As the name suggests, this technique is meant to strengthen the security of asymmetric encryption schemes against chosen ciphertext attacks. The structure of OAEP is illustrated in Figure 19a. Briefly, it uses two cryptographic hash functions  $G$  and  $H$  (which can also be the same function) to process and pad an input sequence split into  $L$  (i.e., left) and  $R$  (i.e., right). First, OAEP computes  $G(R) \oplus L$  to produce the left part of the output. Then, it uses the previous result to produce the right output sequence with  $H(G(R) \oplus L \oplus R)$ .

The basic OAEP construction we have briefly presented cannot be directly used to apply complete mixing to the resource. In fact, as shown in Figure 19a, it does not guarantee that any bit in  $L$  cryptographically affects all the bits in the left output sequence. This is because  $L$  is only XOR'ed to  $G(R)$ , hence the  $i$ -th bit in  $L$  only affects the  $i$ -th bit in the left output sequence. To avoid that this asymmetry and the limited mixing of the left output sequence, we observe that the output sequence on the right, on the contrary, is completely mixed. The explanation is straightforward, the intermediate sequences used to build the right output are both produced applying a cryptographic hash function that, as anticipated previously, ensures complete mixing. We can extend this reasoning to remove the limitation, provided that a third hashing round is added to the basic 2-round OAEP construction. This is illustrated in Figure 19b, in which a uniform texture is used to color both left and right output sequences. For completeness we report here the fact that a 4-round OAEP can be used to produce a super pseudorandom permutation [LR88], however, the fourth round is not needed to create an AONT scheme, and its presence would only have an impact on performance.

With 3-round OAEP ensuring *complete mixing*, we need then to check the support to macro-blocks of arbitrary size and verify the *no-shrinking effect* property. Unfortunately, ordinary hash func-

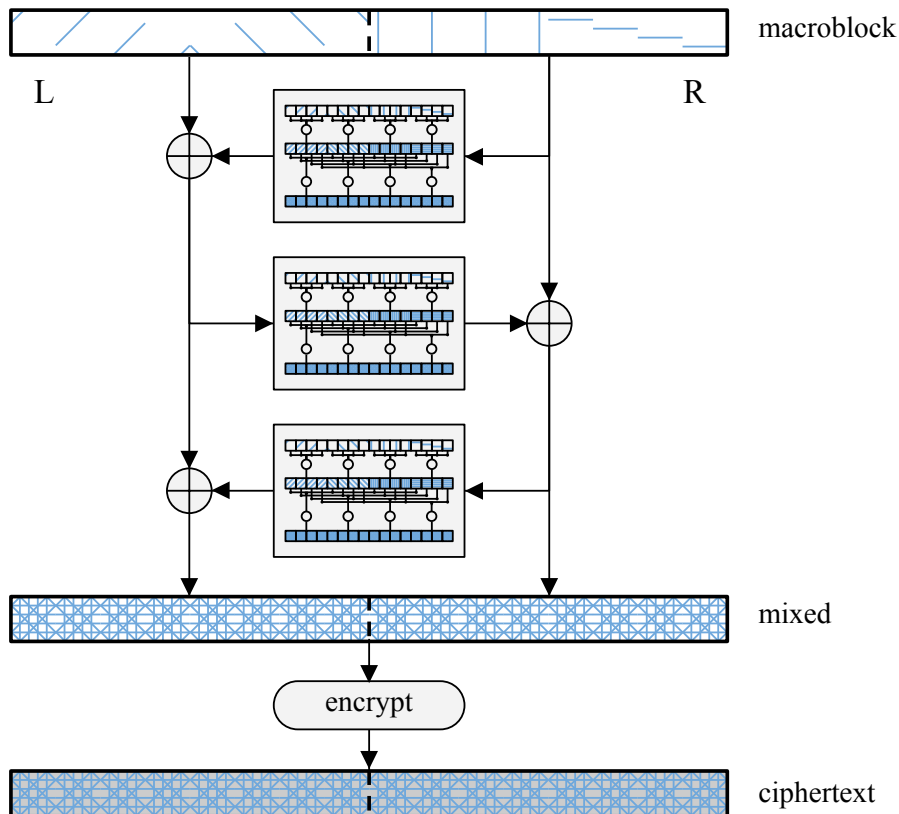


Figure 20: OAEP mixing with internal layered structure

**OAEP-Mix(M)**

- 1: Let  $M=LR$  s.t.  $size(L)=size(R)$  /\* left (L) and right (R) half of M \*/
- 2:  $L_1 := L \oplus \text{AES-Mix}(R)$  /\* first round \*/
- 3:  $R_1 := \text{AES-Mix}(L_1) \oplus R$  /\* second round \*/
- 4:  $L_2 := L_1 \oplus \text{AES-Mix}(R_1)$  /\* third round \*/
- 5:  $M := L_2 || R_1$  /\* mixed macroblock \*/
- 6:  $E(k, M)$  /\* mixed and encrypted macroblock \*/

Figure 21: OAEP mixing of a macro-block M

tions produce a fixed-size output independently to the input size, hence violating the *no-shrinking effect* property. To support macro-blocks of arbitrary size, and at the same time satisfy the *no-shrinking effect* property, we propose to use a stream cipher, or the same AES-Mix function we previously explained, in place of the functions  $G$  and  $H$ . Before discussing this argument in detail, we want to highlight that another aspect must be noticed. Given the output and the functions  $G$  and  $H$  (alternatively the proposed stream cipher together with the decryption key  $k$ ), the entire OAEP construction can be immediately reversed disclosing the plaintext resource. In other words, OAEP does not provide confidentiality. To produce a valid AONT encryption scheme, an extra layer of encryption



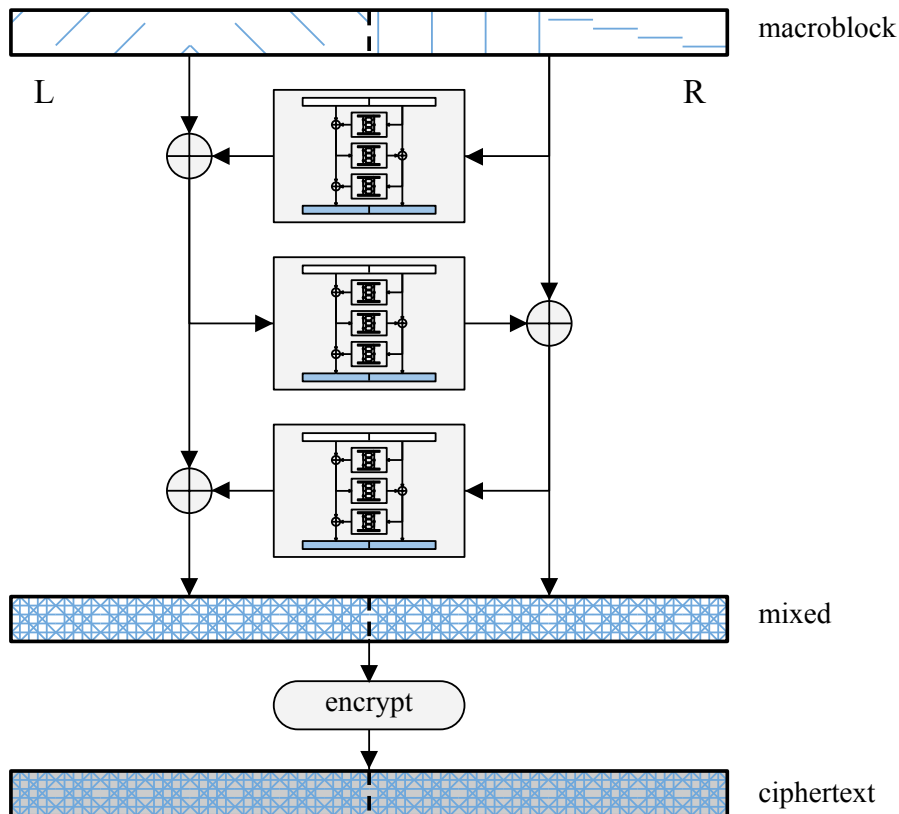
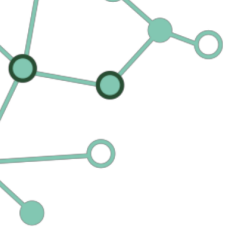


Figure 22: Recursive OAEP mixing

must be applied after the OAEP function has been evaluated. In our implementation we relied on the standard AES in counter mode [Dwo04].

**OAEP-Mix** The first approach we propose to build an OAEP-Mix function is to replace the hashing functions  $G$  and  $H$  with the AES-Mix function described in Section 3.5.1. The layered structure of AES-Mix guarantees complete mixing, supports arbitrarily large macro-blocks, and satisfies the no-shrinking effect. The only modification to AES-Mix is the use of a cryptographic hash function in place of AES (i.e., replacing AES with a cryptographic hash function in the circled E in Figure 15). The resulting scheme is shown in Figure 20, and its pseudocode is reported in Figure 21. Line 6 of the algorithm ensures that only authorized users can access the resource in plaintext. This detail is important and solves the problem associated with the use of invertible functions in the OAEP construction. The cryptographic hash algorithm selected in our implementation is BLAKE2 [SA15]. Compared to AES, it permits to use larger blocks and mini-blocks, possibly reducing the number of rounds required to achieve complete mixing.

**ROAEP-Mix** The second approach we propose is to use a recursive 3-round OAEP mix function, followed by the application of final layer of encryption to ensure confidentiality. This approach is shown in Figure 22, and its pseudocode is reported in Figure 23. The only requirement is that the input macro-block must include  $b = 2^x = (Msize/bsize)$  blocks, with  $bsize$  the size of the output



```

ROAEP-Mix(M)
1: Let M=LR s.t. size(L)=size(R)                                     /* left (L) and right (R) half of M */
2: if Msize/2 = bsize                                             /* L and R can be input of the mix function */
3: then L1 := L ⊕ mix(R)                                           /* first round */
4:   R1 := mix(L1) ⊕ R                                           /* second round */
5:   L2 := L1 ⊕ mix(R1)                                         /* third round */
6: else L1 := L ⊕ ROAEP-Mix(R)                                     /* first round */
7:   R1 := ROAEP-Mix(L1) ⊕ R                                       /* second round */
8:   L2 := L1 ⊕ ROAEP-Mix(R1)                                   /* third round */
9: if Msize = b · bsize
10: then return(E(k, L2 || R1))                                  /* mixed and encrypted macroblock */
11: else return(L2 || R1)                                       /* mixed macroblock */

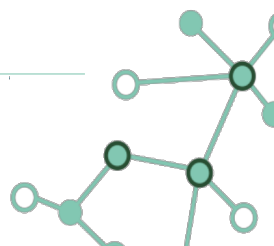
```

Figure 23: Recursive OAEP mixing of a macro-block M

of the selected cryptographic hash function. The main advantage associated with this approach is that it gives more flexibility in the choice of the mini-block size (*msize*). Indeed, the largest *msize* available is the size of the output of the selected cryptographic hash function (e.g., 512 bits in our implementation), and larger mini-blocks provide stronger protection against a brute-force attack. However, as confirmed by our experimental evaluation (Section 3.8), this approach does not produce a significant performance gain. This is due to the widely available hardware implementation of many block ciphers like AES in modern CPUs, which makes them highly performant compared to cryptographic hash functions that are usually implemented by software libraries. As CPU architectures evolve, we expect hardware implementations of cryptographic hash functions to be eventually available, making procedures like ROAEP-Mix more competitive.

### 3.6 Access management

As explained in the introduction of this chapter, an AONT encryption scheme requires access to the decryption key and to the whole resource to enable decryption. In our case, this equates to receiving the decryption key from the resource owner, along with all the mixed and encrypted resource fragments. Indeed, the inability to retrieve even only one fragment implies the lack of at least one mini-block for each resource macro-block, and the consequent inability to perform unmix. Thanks to this property, there is no need to upload to the cloud a re-encrypted version of the whole resource using a fresh key every time the access policy is updated. When an access revocation must be performed, the resource owner can randomly select a fragment, download it, re-encrypt it with a fresh key (that will be sent to users still authorized for the access), and finally re-upload it to the cloud, overwriting the previous fragment version. Notice that this procedure can be repeated many times, without causing significant degradation to the decryption time (see Section 3.8). Figure 24 illustrates a sequence of access revocations. Initially, the resource is mixed using 5 macro-blocks, then it is sliced into the 16 encrypted fragments uploaded to the cloud (quadrant (a) in Figure 24). Quadrants (b), (c), and (d) show the sequence of rewriting operations performed to enforce three access revo-



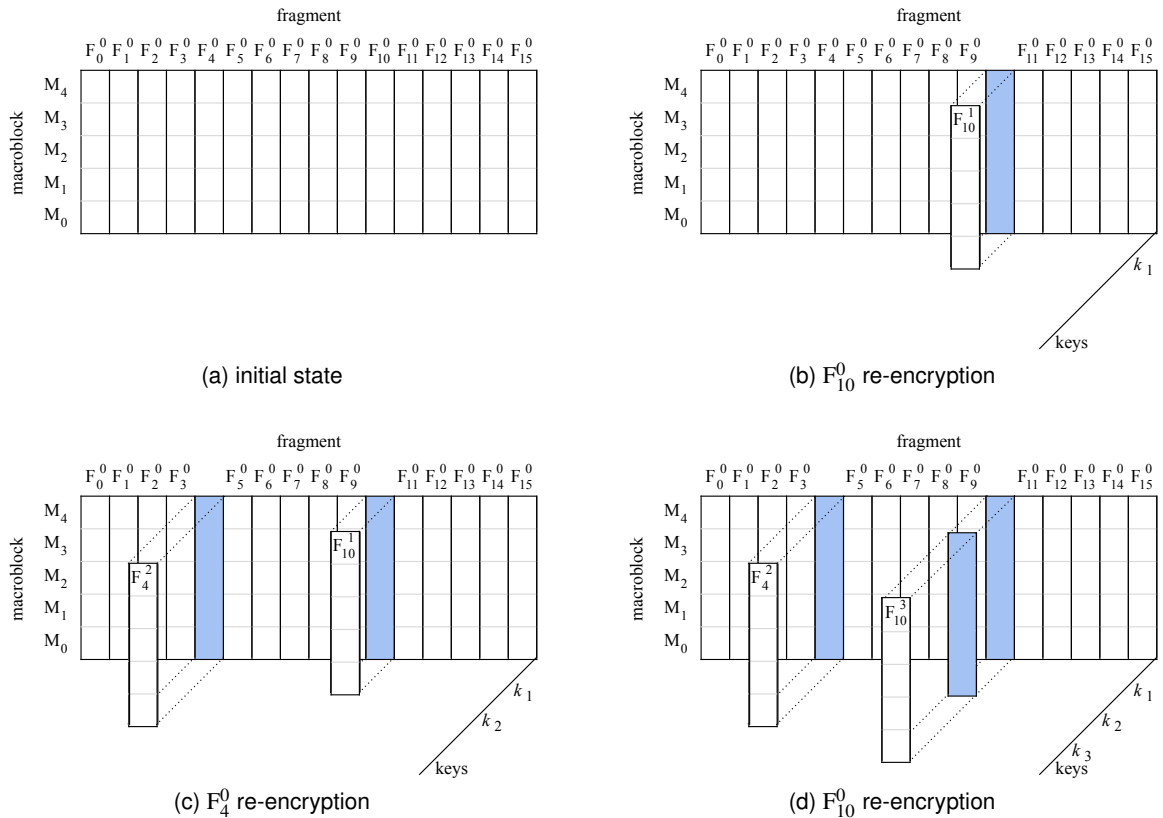


Figure 24: An example of fragments evolution

cations. When the first revocation is performed (quadrant (b)), the resource owner randomly picks fragment  $F_{10}^0$ , re-encrypts it with key  $k_1$  to produce  $F_{10}^1$ , and re-deploys it to the cloud, overwriting the previous version (e.g., a shadow is used in Figure 24 to denote this). When the second revocation is performed (quadrant (c)), the resource owner selects fragment  $F_4$  and key  $k_2$ , repeating the previous procedure. Finally, when the third access revocation is enforced, fragment  $F_{10}$  is selected again, but this time it is re-encrypted with key  $k_3$ , and fragment  $F_{10}^1$  is overwritten, not  $F_{10}^0$ .

Every time a policy update occurs, the resource owner is required to produce a new key. This operation does not introduce additional key management overhead, as a straightforward *key regression* technique [FKK06] can be used to generate an unlimited sequence of new, pseudorandom keys starting from a given key  $k_0$ . The sequence of keys  $k_1, k_2, \dots, k_n$  is produced encrypting  $k_0$  by the owner using an asymmetric encryption technique like RSA. Given  $i$  and  $j$ , with  $j > i$ , only the resource owner will be able to perform forward key derivation producing  $k_j$  from  $k_i$ . All other users will instead perform backward key derivation, or rather they will use their public key to retrieve  $k_z$  from  $k_i$ , when  $z < i$ . On modern architectures the cost to perform this operation is almost negligible, since several thousand key derivations per second can be processed.

Using key regression, every user authorized to access the resource only needs to know the last recently used encryption key, along with the backward derivation key. Notice that, information about the key to be used to decrypt a fragment can be directly stored in a descriptor uploaded to the server



**Revoke**

- 1: randomly select a fragment  $F_i$  of R /\* fragment to be rewritten \*/
- 2: download  $F_i^c$  and its descriptor from the server
- 3: **if**  $c > 0$  **then** /\*  $F_i^0$  has been overwritten in a revocation \*/
- 4:   derive key  $k_c$  /\* derive  $k_c$  using key regression \*/
- 5:    $F_i^0 := D(k_c, F_i^c)$  /\* retrieve the original version of the fragment \*/
- 6:   determine the last key  $k_{l-1}$  used /\* retrieve  $k_{l-1}$  from R's descriptor \*/
- 7:   generate new key  $k_l$
- 8:    $F_i^l := E(k_l, F_i^0)$  /\* encrypt  $F_i^0$  with key  $k_l$  \*/
- 9:   upload  $F_i^l$  overwriting  $F_i^c$  /\* overwrite previous version \*/
- 10: update  $F_i^l$ 's descriptor

Figure 25: Revoke on resource R

**Access**

- 1: download R's fragments and descriptors
- 2: retrieve last encryption key  $k_l$
- 3: compute keys  $k_0, \dots, k_l$
- 4: **for** each downloaded fragment  $F_i^x$  **do**
- 5:   **if**  $x > 0$  **then**
- 6:      $F_i^0 := D(k_x, F_i^x)$  /\* retrieve the original version of fragments \*/
- 7:   **for**  $j = 0, \dots, M - 1$  **do** /\* reconstruct and decrypt macro-blocks \*/
- 8:      $M_j :=$  concatenation of mini-blocks  $F_i^0[j], i = 0, \dots, f - 1$
- 9:     **UnMix**( $M_j$ ) /\* one of: AES-Mix, OAEP-Mix, ROAEP-Mix in decrypt mode \*/

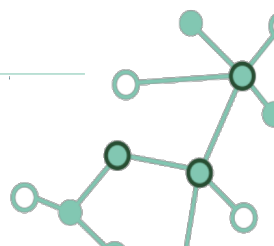
Figure 26: Access to resource R

along with the fragment itself. This permits to perform access revocations simply by updating the resource descriptor and by distributing the last recently used fragment decryption key to the users who still hold access authorization. The process is described in Figure 25.

To access the resource at a given time, users need first to retrieve the last recently used encryption key. Then, users must download all the fragments together with their descriptors. Using the time information stored in each descriptor and the backward derivation key, users can recover the key used to protect each fragment. After decryption is carried out, all mixed macro-blocks are reconstructed from the slices. Finally, unmixing is performed according to one of the three aforementioned techniques. The process is described in Figure 26.

### 3.7 Analysis

In this section we discuss the effectiveness of the approach, or rather we analyze the protection given by the algorithm when a user whom access has been revoked, maintains locally a portion of the resource. It should be noticed that, even after access revocation, our construction does not prevent unauthorized users to download a resource, hence protection is solely enforced by the application of the encryption and by re-writing the fragments. Another important aspect we highlight





is that we make no assumption on the availability of encryption keys; given their compactness, it is realistic that users can store them indefinitely.

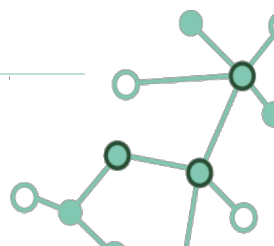
To analyze the effectiveness, we start from the trivial scenario in which a user has lost access to the resource due to the overwriting of a single fragment. In this setting, the user can only utilize the old encryption key (i.e., the key used to access the resource before authorization was revoked) and perform a brute force attack attempting all the possible combinations of missing bits. To perform this attack, it is necessary to test at least  $2^{msize}$  combinations to reconstruct a macro-block. When  $f_{miss}$  fragments have been overwritten due to  $f_{miss}$  update operations, the search space becomes larger, as the unauthorized user must attempt at least  $2^{msize \cdot f_{miss}}$  to reconstruct a macro-block. More sophisticated scenarios occur when the unauthorized user stores locally portions of the original resource. In the following we analyze two cases: i) the user stores locally entire fragments, and ii) the user relies on erasure codes.

**Local storage of fragments** Our approach ensures at least one fragment becomes unintelligible after a revocation. The user whom access is revoked will only be able to access the resource (or part of it) if a copy of that particular overwritten fragment is stored locally. To defend against this attack, the fragment to be overwritten is selected randomly. By doing so the previously authorized user has a probability of  $(f_{loc}/f)$  to succeed in the attack, where  $f_{loc}$  is the number of fragments the user stores locally and  $f$  the total number of fragments. This probability can quickly become negligible after a sequence of access revocations is performed. Indeed, after  $f_{miss}$  different fragments have been overwritten, the probability of success becomes  $P_A = (f_{loc}/f)^{f_{miss}}$ . This makes the attack unfeasible in a real scenario, since to be successful, it requires the unauthorized user to store locally a very high percentage of fragments, essentially paying a storage cost that approaches the maintenance of the whole resource.

A potential extension of this approach could consider, instead of overwriting entire fragments, a randomly chosen set of mini-blocks ensuring coverage of all macro-blocks. However, this approach would significantly increase the complexity in the reconstruction process.

**Local storage of portions of all mini-blocks** An alternative to storing entire fragments is storing a fixed-size portion of all the mini-blocks for each of the fragments. The advantage of this strategy is that, after a single revocation is performed, the number of brute force attack combinations that need to be verified is potentially lower. To give an example, when the user stores half of the bits of each mini-block, the effort to reconstruct a single macro-block involves the verification of  $2^{(msize/2)}$  combinations. However, after a sequence of  $f_{miss}$  different fragments are overwritten, the computational effort grows as  $2^{(msize/2) \cdot f_{miss}}$ , making this strategy ineffective. We highlight that, aside from the required computation effort, this strategy does not seem preferable compared to the previous (i.e., storing entire fragments), since it entails essentially paying a storage cost that approaches the maintenance of the whole resource.

**Local storage of erasure codes** Instead of storing fragments or portions of them, a user could leverage erasure codes [HSX<sup>+</sup>12], which enable the reconstruction of a resource when it is partly damaged (or portions of it are missing). The simplest example of erasure code is the parity bit,



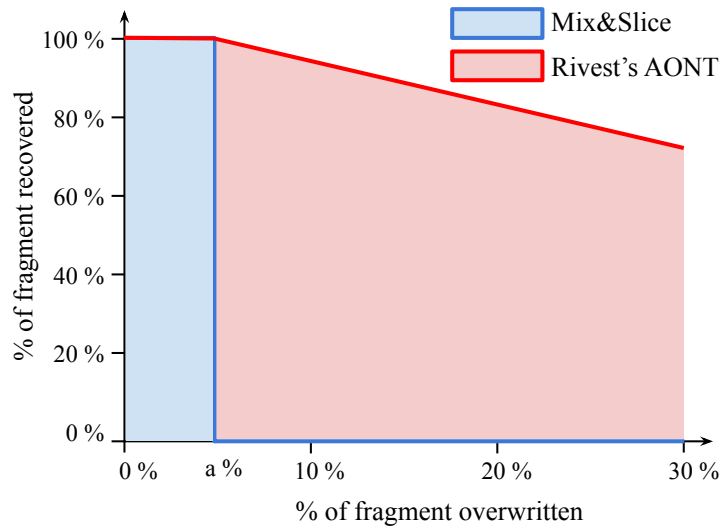
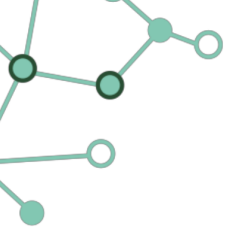


Figure 27: Percentage of resource recovered when the revoked user has kept an erasure code whose size is  $a\%$  of the resource size

which only permits to verify the integrity of a resource when an odd number of bits is corrupted. More advanced techniques like Reed-Solomon [RS60] permit also to reconstruct missing portions of a resource. To this end, these techniques store complementary bit sequences. In general, an erasure code of size  $n$  permits to recover up to  $n$  bit in the original resource. In our scenario the unauthorized user uses this capability to try to get access to the resource after revocation. In detail, the user stores locally a code of size equal to  $t$  fragments, with  $t < f$ , and  $f$  is the total number of fragments. This permits to recover the plaintext resource when no more than  $t$  different fragments have been overwritten.

Erasure codes represent a more efficient attack strategy compared to the previous alternatives, since they permit to store more compact sequences instead of entire fragments (or portions of them). It is interesting to discuss the impact of erasure codes in relation to AONT transformations like the one proposed by Rivest [Riv97] and our proposal. Rivest's transformation relies on the use of a compact key and, as it produces observable states with smaller size compared to the input, it violates the *no-shrinking property*. A direct consequence is that a user could store the AONT key and be able to revert the transformation even after a fragment is updated. The missing fragment in this case would prevent the reconstruction of some portions of the resource, but large parts of it would still be accessible. To give an example, assume that the user maintains an erasure code as big as  $a\%$  of the resource size, and that the owner has overwritten  $r\%$  of the resource size. When  $r \leq a$ , the user would succeed in reconstructing the plaintext resource in both Rivest's scheme and our approach. However, when  $r > a$ , the Rivest's scheme and our approach differ significantly. With Rivest's AONT, the user can apply the erasure code and then use the AONT key to invert the function, regaining access to a portion of the resource potentially as big as  $100 - (r - a)\%$  of the initial size. Conversely, with our approach, even after applying the erasure code, no key can be used to invert the AONT transformation, hence no part of the resource can be recovered. This is



graphically shown in Figure 27. To defend against this attack scenario, more than a single fragment can be overwritten during a policy update.

**A note on collusion** In our analysis we do not consider collusion with the server, which is assumed to be trustworthy to enforce access revocation through fragment overwriting. Collusion hence occurs when two users join their effort to gain access to a resource that neither of them can access. Indeed, two (or more) users can reduce the cost of local storage by maintaining separate and complementary portions (or erasure codes) of the plaintext resource. This scenario does not add any complication to the previous analysis, simply the group of colluding users can be considered as a single entity. Ultimately, this resorts to the group of colluding parties storing the plaintext resource itself.

## 3.8 Implementation and experiments

---

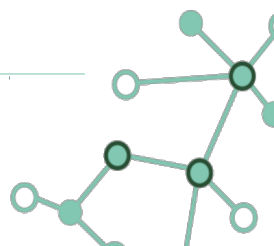
To assess the performance and applicability of our proposal we implemented a client application (Section 3.8.1) responsible for encrypting/decrypting resources, and a server (Section 3.8.2) enabling the storage, retrieval and policy update of resources. The client application is written in Python and relies on a C library that implements the encrypt/decrypt operations. The server is instead implemented as an extension of Swift, a modern, open-source object storage service for the cloud. The server exposes two interfaces: i) `get`, to retrieve a resource, and ii) `put_fragments`, to store/update a fragment at the server. All the tests have been performed on: i) a Linux Ubuntu 22.04 LTS workstation with Intel Xeon CPU E5-2620 v4, 8 cores, for the client; and ii) an Amazon EC2 m4.xlarge instance, with 4 CPUs and 16 GB of RAM, for the server. Network connection is a symmetric 100 Mbps link. In the first part of the experiments we evaluate the throughput for reconstructing the plaintext version of a resource (i.e., decrypt mode). Then, we test the time required to perform policy updates (i.e., fragment overwriting).

### 3.8.1 Mixing throughput

---

The first experiment measures the throughput of a client application reconstructing the plaintext resource, or rather a client performing the unmix operation (i.e., pure decryption time). Two decryption strategies are considered: i) 256KiB macro-blocks and AES-Mix variant of our construction (with  $b_{size}=128$  bits) with both software implementation of AES and its hardware implementation AES-NI (which is supported by many of the Intel x86 CPUs), ii) 256KiB macro-blocks and the OAEP-Mix variant of our construction (with  $b_{size}=512$  bits) leveraging BLAKE2 as the cryptographic hash function for the internal OAEP layer.

The results of our analysis are reported in Figure 28. Clearly, performance varies based on the particular configuration, or rather with the size of the mini-block (32 bits, 64 bits, or 128 bits) and the number of physical threads (1, 2, 4, 8, or 16) available to client. With 32-bit mini-blocks AES-Mix requires 8 rounds (i.e.,  $256\text{KiB}=32 \cdot 4^8$ ), and 15 rounds (i.e.,  $256\text{KiB}=64 \cdot 2^{15}$ ) with 64-bit mini-blocks. Conversely, OAEP-Mix requires 4 rounds (i.e.,  $256\text{KiB}=32 \cdot 16^4$ ) with 32-bit mini-blocks, and 5 rounds (i.e.,  $256\text{KiB}=64 \cdot 8^5$ ) with 64-bit mini-blocks. Moreover, AES-based strategies do not permit to use 128-bit long mini-blocks. In that case only OAEP-Mix can be leveraged, and 7 rounds are required.



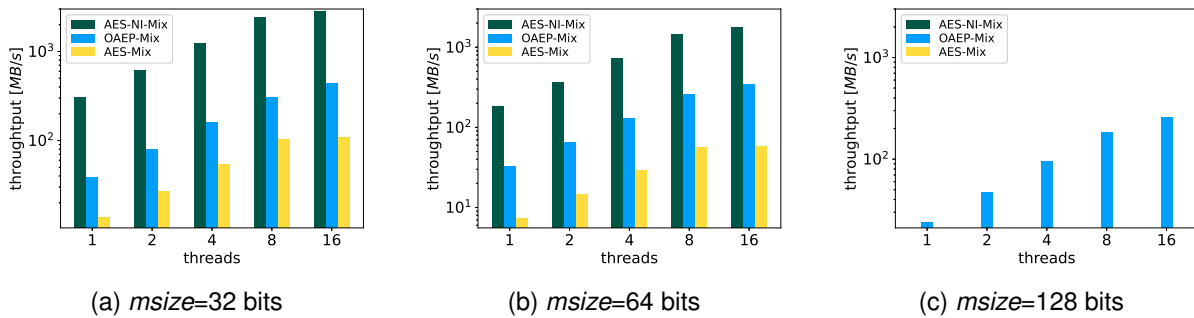


Figure 28: Experimental throughput for several mini-block sizes and varying number of client threads

The data show that the OAEP-Mix strategy outperforms the AES-Mix software implementation. This is due to the bigger block size, which permits to reduce the number of rounds. However, the AES-NI-Mix implementation exhibits the best performance, with a throughput peak of 2.5 GB/s when 32-bit mini-blocks are used. The data also confirm that, for each execution strategy, the throughput grows linearly up to 8 threads, which is the number of physical threads available.

An aspect worth mentioning is that, even if our AONT encryption scheme is associated with a lower throughput compared to the use of AES with a traditional encryption mode (e.g., CBC or CTR), the throughput is still much higher compared to bandwidth of current connections. Again, Figure 28 reports a peak throughput of 2.5 GB/s (roughly 20 Gbps) when 16 threads are used. To give a comparison, this throughput is approximately 1000 bigger than the bandwidth required to stream a 4K video (20 Mbps), and around 20 times bigger than ordinary home broadband connections (1 Gbps). The AWS instance used in our experiment provided a 750 Mbps network bandwidth, hence this was the performance bottleneck every time a client-server interaction was required.

### 3.8.2 Access and update throughput

The second experiment measures the time and the cost, in terms of throughput, of get requests and policy updates. Our implementation leverages Swift on the server side, hence from now on the term *object* will be used in place of resource to align with the official terminology. With regard to the storage of objects, we considered two alternatives: i) separate objects are used by the server to store different fragments, and ii) fragments are managed as sub-objects through the Dynamic Large Objects (DLO)<sup>1</sup> Swift service. From a high level perspective, the first configuration has the advantage that a policy update can be managed simply by overwriting a single object on the server, however, compared to the second configuration, the client must open many concurrent connections with the server (i.e., one for each fragment) to guarantee high throughput. With the second configuration instead (i.e., DLO), the client submits to the server a single get request for the object, delegating to the server the retrieval of all the related sub-objects. The tests consider objects of several sizes (1MB, 4MB, 16MB, 64MB, 256MB, or 1GB), and varying number of fragments (1, 4, 16, 64, 256,

<sup>1</sup>[https://docs.openstack.org/swift/latest/overview\\_large\\_objects](https://docs.openstack.org/swift/latest/overview_large_objects)



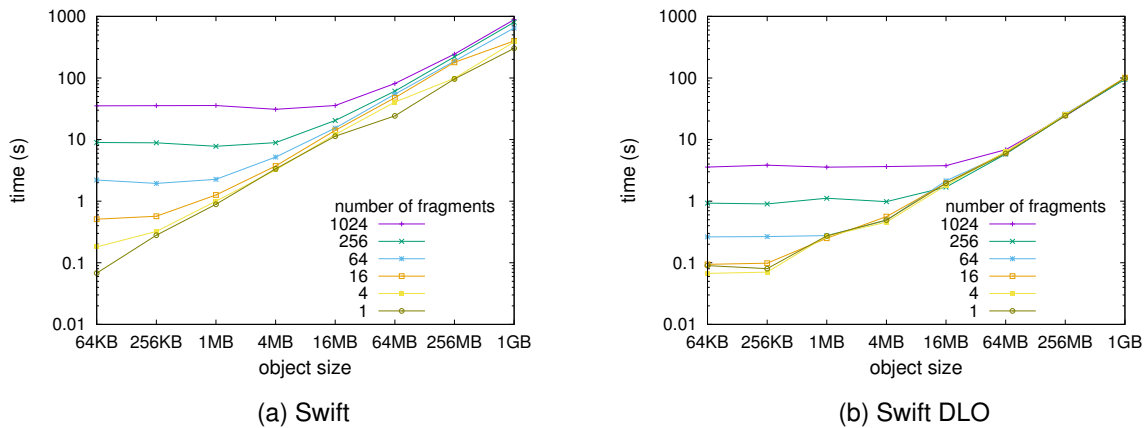


Figure 29: Time for the execution of get requests

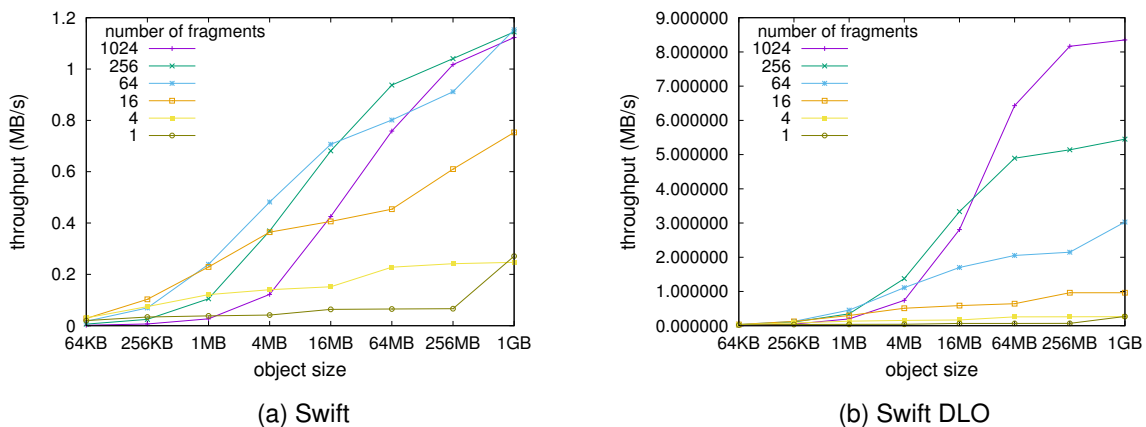
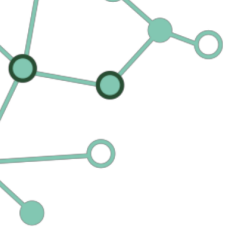


Figure 30: Throughput for a workload combining get and put\_fragment requests

and 1024). The configuration with one fragment per object represents the baseline, since it relates to the case where the object is stored in encrypted form (i.e., no use of AONT encryption).

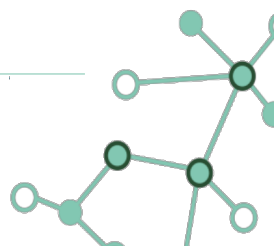
The time required to process a get request is shown in Figure 29. As explained previously, different object sizes and number of fragments are used. Quadrant (a) is associated with the first option (i.e., one object per fragment), while quadrant (b) is associated with the DLO configuration. In both cases, the data clearly show that our approach provides performance similar to the baseline for objects larger than 4MB, especially in the case of the DLO configuration, and as expected, a larger amount of fragments implies higher overhead for get operations. However, the parameter that confirms having the highest impact in the case of large objects is the network bandwidth.

To evaluate the performance of our approach in a scenario where policy updates are performed, we simulate a workload in which a put\_fragment request is issued after 50 get requests. The total number of objects stored by the server is instead fixed to 1000. Again, different object sizes and number of fragments are considered in the experiment. The results are shown in Figure 30. The data clearly show that our approach provides significant performance gain when medium to large-sized objects are stored. Indeed, for objects larger than 4MB the throughput of configurations



using fragments is consistently higher compared to the baseline. This is natural, since the baseline requires complete object overwriting every time a `put_fragment` request is issued, hence our approach proves to be more scalable when many policy updates are performed. Comparing quadrant (a) with (b), the DLO configuration provides higher throughput on average. This is mostly due to the lower time required to perform `get` (see Figure 29). Finally, we highlight that the best configuration of the number of fragments must be determined based on the particular workload of an application, based on the frequency of policy updates (i.e., `put_fragment`) with respect to `get` operations.

To summarize, the experimental evaluation demonstrates the effectiveness of our approach. The size of mini-blocks represent the primary security parameter. The number of fragments, hence the size of the macro-blocks, is instead directly associated with performance. The selection of the best parameter for the number of fragments and the macro-block size ultimately depends on the considered application scenario.





## 4 Data Sanitization

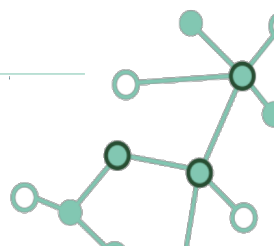
---

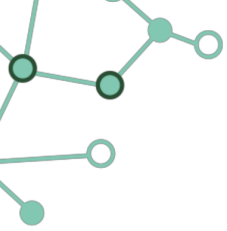
To protect the privacy of individuals in datasets that contain sensitive information, it is not sufficient to simply remove identifying information. We must also obfuscate data that could be used to link individuals to the dataset or to infer their sensitive attributes.  $k$ -anonymity and  $\ell$ -diversity are two techniques that can be used to achieve this.  $k$ -anonymity [Sam01] works by generalizing the values of quasi-identifier attributes. Quasi-identifier attributes are those that could be used to link individuals to the dataset or to infer their sensitive attributes. By generalizing the values of these attributes, we make it more difficult to identify individuals in the dataset.  $\ell$ -diversity [MKG07] builds on  $k$ -anonymity by requiring that each group of  $k$  records that have been generalized to the same values must also contain at least  $\ell$  different values of the sensitive attribute. This ensures that the sensitive attribute is not too concentrated in any one group of records, making it more difficult to infer the sensitive attributes of individuals in the dataset.

$k$ -anonymity and  $\ell$ -diversity are two privacy metrics that are easy to understand, but difficult to implement in practice. This is because they require balancing the privacy of individuals in the dataset with the utility of the data, meaning that the data should still be useful for research or analysis after the dataset has been anonymized. Additionally, the computation of an optimal  $k$ -anonymous solution requires knowing the entire dataset, which is not feasible for large datasets. This means that existing solutions for  $k$ -anonymity and  $\ell$ -diversity implicitly assume that the data is stored in a centralized location. However, this assumption is not realistic for large-scale systems, where data is often distributed across multiple servers. Scalable distributed architectures can help addressing the challenges of anonymizing large datasets. However, the design of these architectures is critical. A simple approach, such as distributing the anonymization load among workers of a distributed system, would not be effective. This is because it would either degrade the quality of the anonymized data or make the computation too slow. Requiring the entire dataset to be processed in a centralized location can be prohibitively expensive for large datasets, and it can also introduce privacy risks if the dataset is not properly secured. Distributed anonymization approaches offer a potential solution to these problems. These approaches partition the dataset into multiple chunks, which are then anonymized by different workers in parallel. This approach can significantly improve performance, and it can also help protecting the privacy by reducing the amount of data that each worker needs to access.

In this chapter, we present a distributed anonymization approach that extends Mondrian [LDR06], an existing algorithm for  $k$ -anonymity. Our approach also supports  $\ell$ -diversity. We support different strategies for generalizing data, including the use of generalization hierarchies. We evaluate our approach on a variety of datasets and show that it can effectively anonymize large datasets while preserving utility. Our results suggest that distributed anonymization is a promising approach for protecting privacy in the age of big data. In particular:

- We use a partitioning strategy that minimizes the need for workers to communicate with each other. This can improve performance and reduce privacy risks.
- We support different strategies for generalizing the data, including the use of generalization hierarchies. This allows data owners to maximize utility.





- We evaluate our approach on a variety of datasets and show that it can effectively anonymize large datasets while maximizing utility.

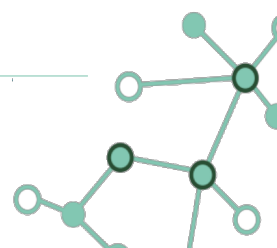
The remainder of this chapter is organized as follows. Section 4.1 discusses the state of the art. Section 4.2 discusses basic concepts over which our solution builds. Section 4.3 presents an overview of our approach for distributed anonymization, modeling the reference scenario and illustrating the actors involved, and a sketch of our anonymization approach. Section 4.4 discusses our approach for partitioning the dataset to be anonymized in fragments to be assigned to the workers for anonymization. Section 4.5 illustrates how workers can independently anonymize the fragments allocated to them. Section 4.6 describes how information loss is assessed. Section 4.7 presents the implementation of our distributed anonymization algorithm. Finally, Section 4.8 illustrates experimental results.

## 4.1 State of the art

---

Privacy protection in data publishing has been widely studied (e.g., [Sam01, Dwo06a, VC02, DFT08, DFT05]). In the literature both syntactic and semantic techniques for anonymizing data have been proposed. Syntactic techniques, such as  $k$ -anonymity [Sam01] and  $\ell$ -diversity [MKG07]), generalize or suppress sensitive attributes to make it difficult to identify individuals. Semantic techniques, such as differential privacy [Dwo06a] and its variations [DR14], add noise to the data to make it difficult to infer sensitive information about individuals. Traditional algorithms for enforcing  $k$ -anonymity and  $\ell$ -diversity operate in centralized scenarios [LDR06, LDR05, HJM07]. However, the problem of distributing and parallelizing anonymization has been recently studied to protect large datasets [ZYL13, ZLD<sup>+</sup>16]. The approach in [ZYL13] partitions the dataset and anonymizes the resulting fragments using the MapReduce paradigm [DG08]. This approach differs from ours in that it aims to maintain the same value distribution in each fragment as in the original dataset, while we aim to maintain homogeneous values for the quasi-identifier in each fragment. This reduces the amount of generalization needed to enforce  $k$ -anonymity, and hence the information loss due to generalization. The distributed anonymization approach in [ZLD<sup>+</sup>16] partitions data so that fragments contain records that are semantically similar. This is achieved using Locally Sensitive Hashing and  $k$ -member clustering. However, this approach does not use Mondrian for computing fragments.

There are a number of different distributed anonymization approaches that rely on distributed architectures for parallelization, similar to our proposal (e.g., [AKS21, AKS21, ABH<sup>+</sup>18, SA17, CWR14, ZQD<sup>+</sup>22, BJA21]). The approach in [AKS21] parallelizes Mondrian using Apache Spark, but it relies on data exchange among workers to coordinate the anonymization of different portions of the original dataset distributed to workers. Our approach, on the other hand, aims to limit data exchange among workers. The solution in [AKS21] differs from ours in that it uses hierarchical clustering and  $k$ -means to provide  $\ell$ -diversity instead of performing partitioning according to the Mondrian strategy. The approach in [ABH<sup>+</sup>18] considers Apache Spark for parallelizing different anonymization approaches, but it does not discuss the Spark-based adaptation of Mondrian. The solution proposed in [SA17] assigns random tuples to workers, our solution, on the other hand, studies a strategy for distributing tuples to workers to minimize information loss. The first approach aimed at parallelizing the Mondrian algorithm was proposed in [CWR14]. This approach uses the





MapReduce paradigm, which requires workers to exchange data with each other. Our approach, on the other hand, minimizes the need for workers to communicate in order to reduce the delays and costs associated with data transfer over a network. A more recent approach to parallelizing the Mondrian algorithm using MapReduce was proposed in [ZQD<sup>+</sup>22]. This approach differs from ours in a few ways. First, it operates on the whole dataset, rather than a sample. This can be more computationally expensive, but it can also provide better anonymization guarantees. Second, it uses a distributed algorithm for partitioning the dataset, which requires workers to communicate with each other. Our approach, on the other hand, minimizes the need for communication between workers. Third, the proposal in [ZQD<sup>+</sup>22] does not use quantiles for partitioning. Our approach uses quantiles to ensure that the anonymized data is consistent across all workers. The proposal in [BJJA21], which enforces Mondrian using Spark, is complementary to ours as it focuses on improving performances by optimizing data structures used by Spark. We, on the other hand, propose a novel distributed enforcement approach for Mondrian, which ensures scalability with limited information loss through a careful data partitioning design.

The idea of distributing the execution of the Mondrian algorithm for anonymizing a large dataset by partitioning a sample of the dataset and limiting data exchange among workers was first proposed in [DFF<sup>+</sup>21c, DFF<sup>+</sup>21a]. In this chapter, we significantly extend the previous work [DFF<sup>+</sup>21c, DFF<sup>+</sup>21a] with the definition and support of novel partitioning strategies, generalization approaches, and metrics for selecting the most suitable attributes for data partitioning and for computing information loss. These contributions have been presented in a paper published in the IEEE Transactions on Big Data [DFF<sup>+</sup>23].

Other data fragmentation-based anonymization solutions use vertical fragmentation of the private table. This type of fragmentation divides the table into multiple fragments, each of which contains a subset of the attributes in the table. The fragmentation is designed to enforce  $\ell$ -diversity (e.g., [XT06]), which ensures that each fragment contains at least  $\ell$  different values for each quasi-identifier attribute. Alternatively, the fragmentation can be designed to protect sensitive associations among attributes in the relation schema (e.g., [DFJ<sup>+</sup>10b, DFJ<sup>+</sup>15]).

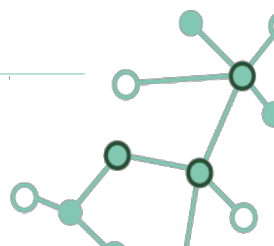
Other researchers have investigated the anonymization of distributed data and/or multiple datasets (e.g., [TG12, KPEK14, DYL<sup>+</sup>13, JX09]). While related, these solutions address a different problem than ours. Their problem is characterized by multiple sources of data (e.g., [TG12, KPEK14]), possibly with multiple privacy requirements (e.g., [DYL<sup>+</sup>13]), and with different owners, each with visibility on its own portion of data (e.g., [JX09]).

## 4.2 Fundamentals

---

Our proposal is based on three main concepts:  $k$ -anonymity,  $\ell$ -diversity, and Mondrian.

**$k$ -Anonymity.**  $k$ -anonymity [Sam01] is a privacy property that aims to protect the identities of respondents in data publication. It does this by ensuring that no tuple in a released dataset can be linked to less than a certain number  $k$  of respondents. This is done by generalizing the values of the quasi-identifier (QI) attributes, which are attributes that can be used to identify respondents, such as gender, date of birth, and living area. For example, an individual's age might be generalized to a range of values, such as [25,30]. This makes it more difficult for an attacker to link a tuple in the



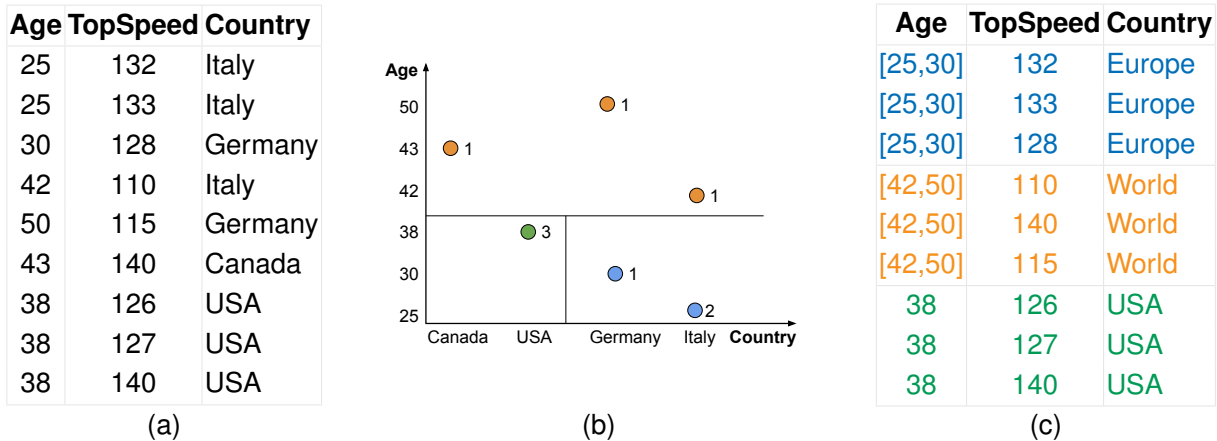


Figure 31: An example of a dataset (a), its spatial representation and partitioning (b), and a 3-anonymous and 2-diverse version (c), considering quasi-identifier  $QI=\{Age, Country\}$  and sensitive attribute *TopSpeed*

released dataset to a specific respondent. A brief example of 3-anonymous version of a dataset is shown in Figure 31(c). The unanonymized raw data reported in Figure 31(a) are generalized on the quasi-identifier attributes *Age* and *Country*. Concretely, the punctual information associated with *Age* has been converted into equally-sized ranges (e.g., [25,30]), while countries have been replaced with larger regions using a taxonomy, described in Figure 32. It is interesting to note that the 7-th, 8-th and 9-th records present the value *USA* for country (i.e., thus no generalization was applied in this case). The reason is that all these records share the same value on the *Country* quasi-identifiers, hence no generalization is necessary. The visualization of the original records into three distinct buckets, identified on a space defined by quasi-identifiers' values, is shown in Figure 31(b).

However,  $k$ -anonymity is not a perfect solution. It is possible for an attacker to still link a tuple to a specific respondent if they have access to additional information, such as the respondent's name or social security number. Additionally,  $k$ -anonymity does not protect against attacks that target sensitive attributes, such as medical information or political affiliation. Despite these limitations,  $k$ -anonymity is a valuable approach for protecting the privacy of respondents in data publication. It is a simple and effective way to make it more difficult for attackers to link data points to specific individuals.

**$\ell$ -Diversity.**  $\ell$ -Diversity [MGK06] is a privacy property that extends  $k$ -anonymity to prevent attribute disclosure. Attribute disclosure occurs when an attacker can infer the value of a sensitive attribute for a particular respondent based on the values of the quasi-identifier (QI) attributes. For example, suppose that a dataset contains *Age* as a quasi-identifier attribute and a sensitive attribute *TopSpeed*. If the dataset is 3-anonymous, then no tuple can be linked to less than 3 respondents. However, if all of the respondents who are 25 years old have the same top speed, then an attacker could still infer the top speed of a particular respondent by knowing their age. With reference to the data shown in Figure 31(a), assume that the *TopSpeed* value associated with the person aged 42 from *Italy*

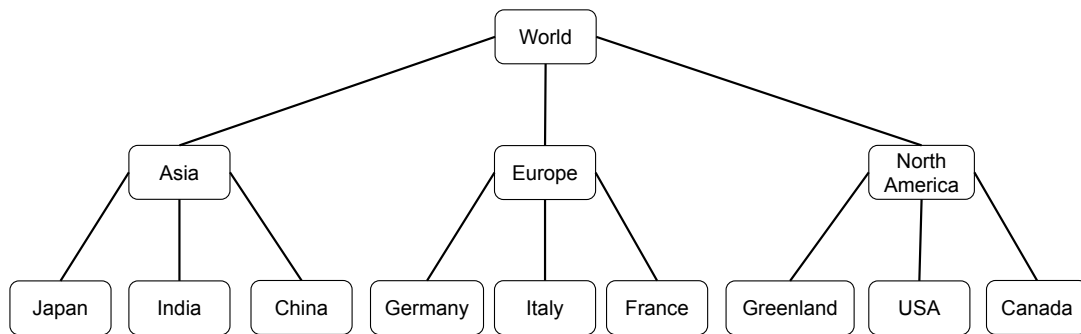


Figure 32: Generalization hierarchy for attribute Country

were 115. The second bucket (i.e., the central portion) of Figure 31(c) would have value 115 on all records (i.e., the 4-th, 5-th and 6-th). Therefore, in the scenario when an attacker is able to associate a particular individual to the second bucket, the sensitive value is revealed.  $\ell$ -diversity prevents this type of attribute disclosure by requiring that each equivalence class (i.e., each set of tuples with the same values for the QI attributes) contains at least  $\ell$  different values for the sensitive attribute. In the example above, the dataset would be 2-diverse if there were at least 2 different top speeds for the respondents who are 25 years old.  $\ell$ -diversity is a stronger privacy property than  $k$ -anonymity, and it is more difficult to achieve. However, it provides better protection against attribute disclosure.

**Mondrian.** Mondrian [LDR06] is a multi-dimensional algorithm that uses spatial partitioning to anonymize data, supporting  $k$ -anonymity and  $\ell$ -diversity. It does this by creating a spatial representation of the data, where each quasi-identifier attribute is mapped to a dimension and each combination of values of the quasi-identifier attributes is mapped to a point in space. Mondrian then recursively partitions the space into regions that contain a certain number of points. This corresponds to splitting the dataset represented by the points in the space into fragments that contain a certain number of records. Mondrian works by iteratively partitioning the data into smaller and smaller fragments  $F$ . At each iteration, Mondrian chooses a quasi-identifier attribute and partitions the data based on the values of that attribute. For example, for a numerical attribute, Mondrian might partition the data into two fragments: those with values lower than the median and those with values higher than the median. The algorithm terminates when any further partitioning would create fragments with fewer than  $k$  tuples. At this point, the values of the quasi-identifier attributes in each fragment are replaced with their generalization. The spaces identified by the Mondrian algorithm for the data reported in Figure 31(a) are visualized in Figure 31(b). The spaces have been identified performing two consecutive cuts: the first one is on the Age quasi-identifier (between values 38 and 42), and the second on the quasi-identifier Country (between the values USA and France, only for the records with value lower than 38 on the Age quasi-identifier). Any further partitioning (i.e., cut), would violate the 3-anonymity requirement.

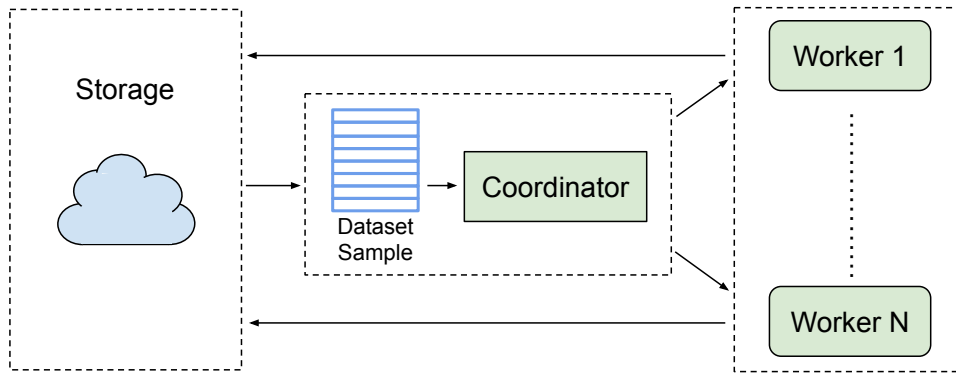


Figure 33: Overall view of the distributed anonymization process

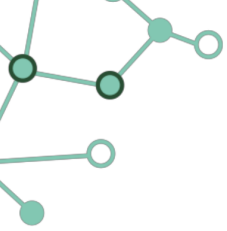
### 4.3 Anonymization process

We consider a scenario where a large and possibly distributed dataset  $\mathcal{D}$ , that may not entirely fit into the main memory of a single machine, needs to be anonymized. The idea is to distribute the anonymization process of  $\mathcal{D}$  to a set  $W = \{w_1, \dots, w_n\}$  of workers so to obtain a parallel and independent computation from one another, guaranteeing the quality of the solution without compromising the performance (with respect to a traditional centralized anonymization of  $\mathcal{D}$ ). The proposed extension of Mondrian assigns workers in  $W$  to (non-overlapping) partitions of  $\mathcal{D}$  (i.e., sets of tuples of  $\mathcal{D}$ , which we call *fragments*) and it operates limiting the need for data exchanges with other workers.  $K$ -anonymity and  $\ell$ -diversity are used to anonymize the fragment assigned to each worker, ensuring that the combination of all anonymized fragments is a version of  $\mathcal{D}$  that satisfies  $k$ -anonymity and  $\ell$ -diversity. The reference scenario we are considering is represented in Figure 33. The Coordinator, executed on a dedicated node, oversees the entire process consisting of a *pre-processing phase* and a *wrap-up phase*. The pre-processing phase partitions  $\mathcal{D}$  in fragments and assigns them to workers. The wrap-up phase collects and recombines the anonymized fragments from the workers and evaluates the quality of the computed solution.

In the reference scenario there is a (*distributed*) *storage platform*, which stores and manages the dataset  $\mathcal{D}$  to be anonymized, its anonymized version  $\hat{\mathcal{D}}$  (after workers' anonymization process), the anonymizing *workers*, and the Coordinator. In the remainder of this chapter, given a dataset  $\mathcal{D}$  with quasi-identifier  $QI = \{a_1, \dots, a_q\}$  and privacy parameters  $k$  and  $\ell$ , we denote with  $\hat{\mathcal{D}}$  the  $k$ -anonymous and  $\ell$ -diverse version of  $\mathcal{D}$ ; with  $\hat{t} \in \hat{\mathcal{D}}$  the generalized version of  $t$  in the anonymized dataset,  $\forall t \in \mathcal{D}$ ; and with  $\hat{F}$  the anonymized version of fragment  $F$  (i.e.,  $\forall t \in F, \exists \hat{t} \in \hat{F}$  s.t.  $\hat{t}$  is the generalized version of  $t$ ).

The proposed distributed anonymization process relies on a pre-processing phase which is crucial to guarantee the correctness of the solution. An important point is the Coordinator's definition of a fragmentation strategy. This strategy regulates how each tuple is assigned to each fragment. Our experiments (Section 4.8) show that fragmenting the dataset  $\mathcal{D}$  based on the values of (some of the) quasi-identifying attributes  $a_1, \dots, a_h$ , so that a tuple  $t$  of  $\mathcal{D}$  is assigned to a fragment  $F$  based on the values of  $t[a_1], \dots, t[a_h]$ , is an effective solution.





Considering the dataset in Figure 31(a) we can suppose to define two fragments  $F_1$  and  $F_2$  based on the values of Age. A possible strategy defined by Coordinator could be one such that  $F_1$  contains all tuples of  $\mathcal{D}$  with values up to 38, and  $F_2$  the remaining ones. The Coordinator would need a complete visibility over  $\mathcal{D}$  to define each fragment.

However, since  $\mathcal{D}$  space requirements might be too large and Coordinator's main memory could be limited, we propose a strategy that Coordinator can use to define the conditions that regulate  $\mathcal{D}$ 's fragmentation based on a *sample* of it. The *sample*'s size can be dynamically adjusted based on the storage capabilities of the Coordinator. The Coordinator communicates these conditions to the workers, which will download the tuples in  $\mathcal{D}$  that satisfy such conditions from the storage platform, without sharing any dataset between the Coordinator and the workers.

Considering the example above, the Coordinator sends to workers the value ranges of Age for the tuples in their fragments. The anonymization process operates in parallel at the workers after the pre-processing phase. Defining this phase, we specifically focus on the support of generalization hierarchies for categorial attributes to the aim of producing anonymous data by semantically generalizing them. The pre-processing phase is discussed in Section 4.4, while the anonymization and wrap-up phases are described in Sections 4.5 and 4.6, respectively.

## 4.4 Data pre-processing

---

The pre-processing phase of our approach operates on a sample  $D$  of  $\mathcal{D}$ , the size of which is adjusted according to Coordinator's storage capabilities. Since  $D$  is a sample of the original dataset  $\mathcal{D}$  to be anonymized, the quasi-identifying attributes considered for  $D$  are those defined for  $\mathcal{D}$ .

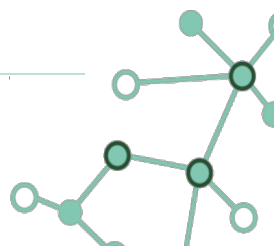
### 4.4.1 Partitioning techniques

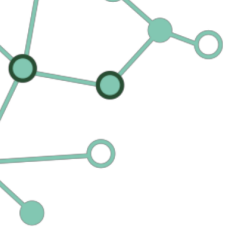
---

Selecting a strategy for partitioning the dataset is crucial in our scenario, as random partitioning can cause significant information loss. Indeed, if the fragments include tuples with heterogeneous values for the quasi-identifier, each worker, operating independently, would require a great number of generalizations to satisfy the  $k$ -anonymity requirement. On the other hand, the amount of information loss is reduced if the data are partitioned in such a way that similar tuples are placed in the same fragments. To illustrate this concept, consider a dataset with four tuples  $t_1, \dots, t_4$  having values 25, 25, 60, and 60 for Age, which needs to be partitioned in two fragments. If partitioning generates two fragments  $F_1 = \{t_1, t_2\}$  and  $F_2 = \{t_3, t_4\}$ , no generalization is needed to enforce 2-anonymity. On the contrary, fragments  $F_1 = \{t_1, t_3\}$  and  $F_2 = \{t_2, t_4\}$  require generalizing Age to the range [25,60] in each fragment, causing higher information loss.

To limit the information loss implied by the partitioning of a sample  $D$  with quasi-identifier QI among a set  $W = \{w_1, \dots, w_n\}$  of workers, we propose two strategies.

- The *quantile-based approach* selects an attribute  $a$  from the QI, and partitions  $D$  in  $n$  fragments  $F_1, \dots, F_n$  according to the  $n$ -quantiles of  $a$  in  $D$ .
- The *multi-dimensional approach* recursively partitions  $D$  in a similar way as the Mondrian approach (see Section 4.2). Given a fragment  $F$  (the entire sample  $D$  at the first iteration), the





## Q\_PARTITION( $D, W$ )

- 1: **let**  $a$  be the attribute used to partition  $D$
- 2:  $R := \{\text{rank}(t[a]) \mid t \in D\}$  /\* rank of  $a$ 's values in the ordering \*/
- 3: **let**  $q_i$  be the  $i^{\text{th}}$   $|W|$ -quantile for  $R, \forall i = 1, \dots, |W|$
- 4:  $F_1 := \{t \in D \mid \text{rank}(t[a]) \leq q_1\}$
- 5: **for each**  $i = 2, \dots, |W|$  **do**
- 6:      $F_i := \{t \in D \mid q_{i-1} < \text{rank}(t[a]) \leq q_i\}$

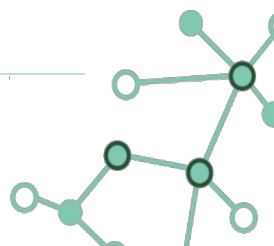
Figure 34: Quantile-based partitioning

multi-dimensional strategy selects an attribute  $a \in QI$  and partitions  $F$  in two fragments according to the median value of  $a$  in  $F$ . Each of the resulting fragments is then further partitioned, until  $n$  fragments have been obtained.

Both methods rely on the fact that the values of the attribute  $a$  selected for partitioning can have an order. This order is used to compute quantiles (for the quantile-based approach) and median values (for the multi-dimensional approach). In fact, given the sample  $D$  and the attribute  $a$  selected for partitioning, the tuples in  $D$  are first arranged in ascending order according to their value of  $a$ . This establishes a ranking among the attribute values. Quantiles and the median values are then computed on this ranking. If  $a$  is numerical, the ordering among values is naturally defined. If  $a$  is categorical and has a generalization hierarchy  $\mathcal{H}(a)$ , the attribute values are considered in the order in which they appear in the leaves of  $\mathcal{H}(a)$ . This is done in order to keep values that generalize to a more specific value in the same fragment.

Indeed, leaf values that are similar will have a common ancestor that is more general than both of them (see Section 4.5), thus limiting information loss. For instance, with reference to the hierarchy in Figure 32, we use order  $\langle \text{Italy, France, Spain, USA, Canada, Greenland, China, Japan, India} \rangle$ . This ordering would combine in the same fragment values *Italy* and *France*, which generalize to a more specific value (i.e., *Europe*) than a fragment with values *Italy* and *Canada* (i.e., *World*).

The process followed by the Coordinator to anonymize a dataset according to the quantile-based partitioning strategy is shown in Figure 34. Given a dataset  $D$  and a set  $W$  of workers, the procedure selects the attribute  $a$  for partitioning, orders the tuples in  $D$  according to  $t[a]$  (i.e., the value of attribute  $a$  in tuple  $t$ ), and determines the rank  $\text{rank}(t[a])$  of each value  $t[a]$  (lines 1–2). After these steps have been performed, the Coordinator computes the  $|W|$ -quantiles of such ranking  $R$  (line 3). The first fragment  $F_1$  is obtained by including all the tuples  $t \in D$  with rank of  $t[a]$  lower than or equal to the first computed quantile (line 4). The remaining fragments  $F_2, \dots, F_{|W|}$  are obtained by including in  $F_i$  all the tuples  $t \in D$  with ranks of  $t[a]$  in the interval  $(q_{i-1}, q_i]$ , with  $q_i$  the  $i^{\text{th}}$   $|W|$ -quantile of the computed ranks (lines 5–6). To exemplify this procedure, consider again to partition the data in Figure 31(a) with the quantile-based approach over attribute *Age*. For the first tuple  $t_1$ , we have that  $\text{rank}(t_1[\text{Age}]) = \text{rank}(25) = 1$ , while for the third tuple  $t_3$ ,  $\text{rank}(t_3[\text{Age}]) = \text{rank}(30) = 2$ . Hence, the 4-quantiles  $q_1, \dots, q_4$  given the ranks are  $q_1 = 2, q_2 = 3, q_3 = 4, \text{ and } q_4 = 6$ . The first fragment  $F_1$  then includes all the tuples  $t$  such that  $\text{rank}(t[\text{Age}]) \leq 2$ , that is, all the tuples such that  $t[\text{Age}] \leq 30$ . Fragment  $F_2$  includes all tuples  $t$  such that  $2 < \text{rank}(t[\text{Age}]) \leq 3$ . Similar considerations apply to fragments  $F_3$  and  $F_4$ .





---

```
M_Partition( $D, W, i$ )
1: let  $a$  be the attribute used to partition  $D$ 
2:  $R := \{rank(t[a]) \mid t \in D\}$  /* rank of  $a$ 's values in the ordering */
3: let  $m$  be the median of  $R$ 
4:  $F_1 := \{t \in D \mid rank(t[a]) \leq m\}$ 
5:  $F_2 := \{t \in D \mid rank(t[a]) > m\}$ 
6: if  $i < \lceil \log_2 |W| \rceil$  then
7:   M_Partition( $F_1, W, i + 1$ )
8:   M_Partition( $F_2, W, i + 1$ )
```

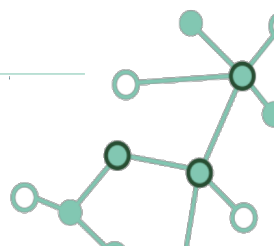
---

Figure 35: Multi-dimensional partitioning

The sequence of quantile-based cuts we have implemented is based on a recursive procedure, which is shown in Figure 35. At each step, this procedure receives a dataset  $D$ , the set  $W$  of workers, and the recursive level of iteration  $i$  (starting from 1). Initially, it selects an attribute  $a$  for partitioning, then it orders the tuples according to  $t[a]$  so to determine the rank  $rank(t[a])$  of each value  $t[a]$  (lines 1–2). This operations permits to compute the median value  $m$  for  $R$  (line 3), and based on it, two fragments are identified (lines 4–5). Finally, the procedure recursively calls itself (lines 7–8), until the necessary number of iterations have already been performed.

The quantile-based and multi-dimensional partitioning have different ways of defining fragments. The quantile-based partitioning ensures that all  $n$  fragments have (approximately) the same number of tuples, but it can only be used if the number of workers is less than or equal to the number of distinct values in the attribute used for partitioning. The multi-dimensional partitioning can be used for any number of workers, but it may result in some workers having more work than others. If the number of workers  $n$  is a power of 2, the multi-dimensional partitioning can be used to create  $n$  fragments of (approximately) the same size by recursively partitioning the data  $\log_2 n$  times. However,  $n$  may not be a power of 2. Aiming at using all the workers, the partitioning strategy stops when  $n \leq 2^i$  (multi-dimensional partitioning generates  $2^i$  fragments at the  $i$ -th iteration) and  $2^i - n$  workers are assigned two fragments, resulting in some workers having twice the workload of the others. For instance, assume  $W = \{w_1, \dots, w_7\}$  and  $|D| = 1000$ . Multi-dimensional partitioning needs 3 iterations for generating at least 7 fragments ( $2^2 < 7 \leq 2^3$ ). Since  $2^i - n = 8 - 7 = 1$ , one worker (e.g.,  $w_1$ ) will be assigned two fragments, resulting in a workload of nearly 250 tuples for  $w_1$  and of 125 tuples for each of the other workers.

The computational complexity of the two approaches is slightly different, with quantile-based partitioning resulting more efficient than multi-dimensional partitioning (as confirmed by the experimental results in Section 8). Procedure  $Q\_Partition$  costs  $O(|D|)$ , while procedure  $M\_Partition$  costs  $O(|D| \log |W|)$ . The first two steps (lines 1-2) are the same for the two procedures and cost  $O(|D|)$ , and the computation of quantiles has the same cost as the computation of the median and cost  $O(|D|)$ . The *for each* loop at line 5 of procedure  $Q\_Partition$  has cost  $O(|W|)$ , and therefore the overall cost of quantile-based partitioning is  $O(|D|)$ , since the number of workers is smaller than the number of tuples in the dataset. The recursive calls of procedure  $M\_Partition$  imply a cost of  $O(|D| \log |W|)$  since the procedure recursively calls itself with  $i$  from 1 to  $\lceil \log_2 |W| \rceil$  and, for each value of  $i$ , the overall size of the fragments input to the different recursive calls is  $|D|$ .





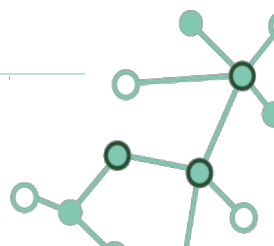
## 4.4.2 Fragments retrieval

---

Workers need to operate on the whole fragment assigned to them. To minimize communication overhead, our solution defines fragments assigned to workers following the *partitioning conditions* defined by the Coordinator. These conditions, are communicated to workers (see Section 4.7). Each worker can then retrieve the tuples in its fragment directly, without need for the Coordinator to retrieve and communicate such tuples. For example, we can consider two fragments  $F_1$  and  $F_2$  computed over a sample  $D$  of  $\mathcal{D}$  using the multi-dimensional approach, and assume to adopt attribute *Age* for partitioning and that the median of the ranking corresponds to value 38 in the attribute domain. The worker responsible of the anonymization of  $F_1$  will retrieve all the tuples in  $\mathcal{D}$  having  $\text{Age} \leq 38$ , while the worker responsible of the anonymization of  $F_2$  will retrieve all the tuples having  $\text{Age} > 38$ .

Given a set  $W = \{w_1, \dots, w_n\}$  of workers,  $c_i$  denotes the condition describing the fragment  $F_i$  assigned to  $w_i$ ,  $i = 1, \dots, n$ . With regard to the quantile-based approach, condition  $c_i$ ,  $i = 1, \dots, n$ , indicates the values for  $a$  that are included in the  $i$ -th  $n$ -quantile of  $D$ . Continuing the example in Section 4.4.1, for the quantile-based partitioning of the dataset in Figure 31(a) in 4 fragments over attribute *Age*, the conditions identifying fragments  $F_1, \dots, F_4$  would be defined as follows:  $c_1 = \text{"(Age} \leq 30\text{"}$ ;  $c_2 = \text{"(Age} > 30\text{) AND (Age} \leq 38\text{"}$ ;  $c_3 = \text{"(Age} > 38\text{) AND (Age} \leq 42\text{"}$ ; and  $c_4 = \text{"(Age} > 42\text{) AND (Age} \leq 50\text{"}$ . When instead the multi-dimensional strategy is used, condition  $c_i$ ,  $i = 1, \dots, n$ , is a conjunction of conditions of the form  $a \leq v$  or  $a > v$  describing the recursive partitioning performed by the Coordinator to obtain fragment  $F_i$ . Intuitively, each recursive call to `M_Partition` (Figure 35) partitions the input fragment  $D$  into two fragments  $F_1$  and  $F_2$ , described by condition  $c$  AND  $(a \leq v)$  or  $c$  AND  $(a > v)$ , respectively, with  $c$  the condition describing the input fragment  $D$  (empty at the first iteration),  $a$  the attribute selected for partitioning, and  $v$  the value in the domain of  $a$  corresponding to the median  $m$  of the ranking of the tuples in  $D$  according to  $a$  (i.e.,  $v = t[a]$  s.t.  $\text{rank}(t[a]) = m$ ). To give an example of this, consider again the scenario depicted in Section 4.4.1, partitioning the dataset in Figure 31(a) in 4 fragments. Initially, partitioning based on attribute *Age* produces  $F_{12}$  and  $F_{34}$  with conditions  $\text{Age} \leq 38$  and  $\text{Age} > 38$ . At next iteration, suppose that  $F_{12}$  is split into  $F_1$  and  $F_2$  base on the values of attribute *Country*, which is *Italy* or *France* in  $F_1$  and *USA* or *Canada* in  $F_2$ . The conditions describing fragments  $F_1$  and  $F_2$  would be  $c_1 = \text{"(Age} \leq 38\text{) AND (Country IN \{Italy, France\})\text{"}$ ;  $c_2 = \text{"(Age} \leq 38\text{) AND (Country IN \{USA, Canada\})\text{"}$ . Figure 36 and Figure 37 show the pre-processing operations carried out by the Coordinator to partition a sample  $D$  of  $\mathcal{D}$  to produce 4 fragments (each one to be assigned to a specific worker).

As a final note in this section, we discuss the possibility of leveraging multiple workers for the pre-processing. The multi-dimensional approach (Figures 35 and 37) to data partitioning can be parallelized by workers. For example, each of the two fragments ( $F_1$  and  $F_2$ ) produced by the partitioning of a fragment  $F$  can be assigned to a different worker for further partitioning. This would allow the partitioning to run in parallel, which would reduce the computation effort required by the Coordinator. However, it would also require data transfer between workers, which could result in lower performance than the traditional, non-parallelized multi-dimensional partitioning approach.



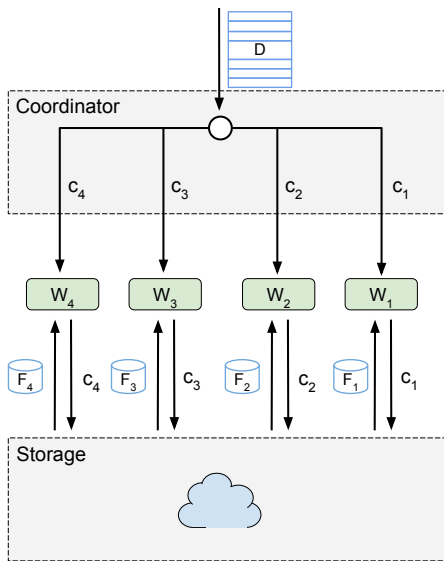


Figure 36: Quantile-based partitioning

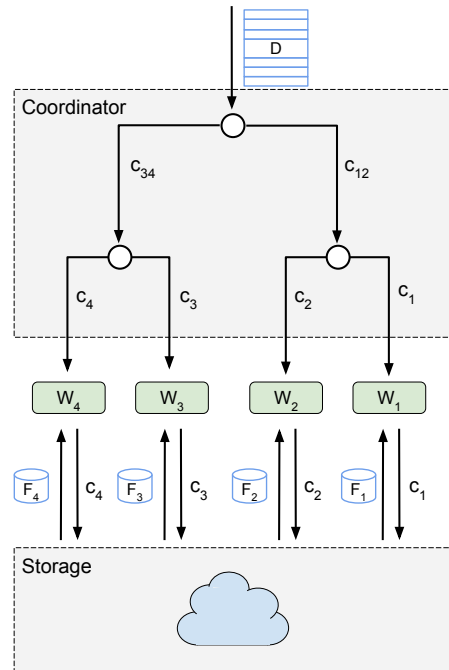


Figure 37: Multi-dimensional partitioning

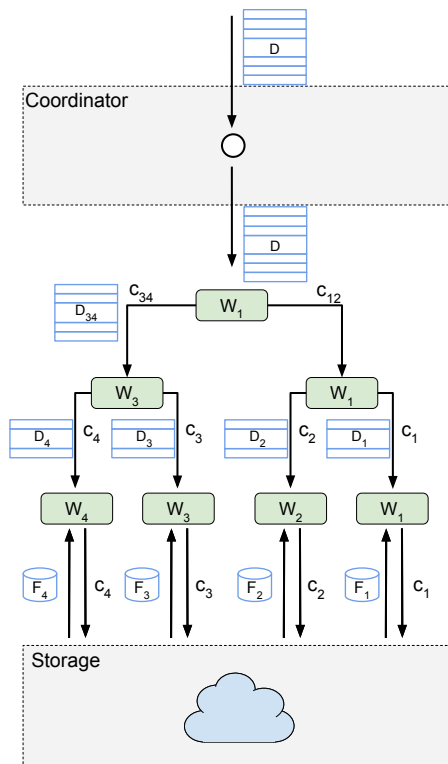
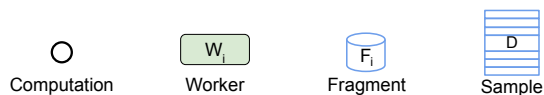
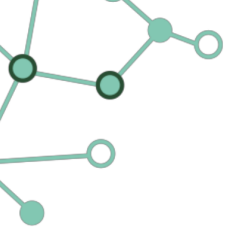


Figure 38: Parallelized multi-dimensional partitioning





### 4.4.3 Partitioning attributes

The first step for partitioning the dataset, regardless of the approach (i.e., quantile-based or multi-dimensional) adopted, is the selection of the quasi-identifying attribute  $a$  used to split (line 1 in procedures Q\_Partition in Figure 34 and M\_Partition in Figure 35).

For quantile-based partitioning, we select the attribute  $a_i \in \text{QI}$  with more distinct values in  $D$ . This strategy distributes the values for  $a_i$  among different fragments, limiting the necessary number of generalizations over  $a_i$ . Indeed, generalization needs to operate only over the subset of values for  $a_i$  appearing in the fragment, which are expected to be close. On the contrary, partitioning according to a different attribute  $a_j$  having a limited number of values might cause excessive generalization for  $a_i$ , if a fragment has tuples with values at the extremes of the domain for the attribute.

For multi-dimensional partitioning, similarly to the original Mondrian approach, we select the attribute  $a \in \text{QI}$  that has, in the fragment  $F$  to be partitioned, the highest representativity of the values it assumed in  $D$ . If  $a$  is numerical, its representativity is defined as the ratio between the span (i.e., the width of the range) of the values in  $F$  and the span of the values in the entire dataset  $D$ . If  $a$  is categorical, its representativity can be defined as the ratio between the number of distinct values in  $F$  and the number of distinct values in  $D$ . Formally, the representativity  $rep(a)$  of a quasi-identifying attribute  $a \in \text{QI}$  is defined as follows:

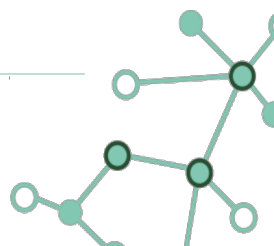
$$rep(a) = \begin{cases} \frac{\max_F\{t[a]\} - \min_F\{t[a]\}}{\max_D\{t[a]\} - \min_D\{t[a]\}} & \text{if } a \text{ is numerical} \\ \frac{\text{count}_F(\text{distinct } t[a])}{\text{count}_D(\text{distinct } t[a])} & \text{if } a \text{ is categorical} \end{cases} \quad (4.1)$$

where  $\max_F\{t[a]\}$  and  $\min_F\{t[a]\}$  ( $\max_D\{t[a]\}$  and  $\min_D\{t[a]\}$ , resp.) are the maximum and minimum values for  $a$  assumed by the tuples in fragment  $F$  (in dataset  $D$ , resp.); and  $\text{count}_F(\text{distinct } t[a])$  ( $\text{count}_D(\text{distinct } t[a])$ , resp.), is the number of distinct values assumed by attribute  $a$  in  $F$  (in  $D$ , resp.). For both numerical and categorical attributes,  $rep(a) \in (0, 1]$ . For example, consider a fragment  $F$  with  $\text{QI} = \{a_1, a_2, a_3\}$ , where  $a_1$  is categorical and  $a_2$  and  $a_3$  are numerical. Suppose that: *i*) the distinct values for  $a_1$  are 1000 in  $D$  and 100 in  $F$ ; *ii*) the values for  $a_2$  are in a range of width 1000 in  $D$  and of width 500 in  $F$ ; and *iii*) the values of  $a_3$  are in a range of width 1000 in  $D$  and of width 700 in  $F$ . Since  $rep(a_1) = 100/1000 = 0.1 < rep(a_2) = 500/1000 = 0.5 < rep(a_3) = 700/1000 = 0.7$ ,  $a_3$  is chosen for partitioning  $F$ .

Note that, at the first iteration of multi-dimensional partitioning, all quasi-identifying attributes have representativity equal to 1 because  $F=D$  (the entire dataset  $D$  is to be partitioned). In these cases, we select the attribute with the maximum number of distinct values in the dataset (like in quantile-based partitioning).

## 4.5 Data anonymization

In our approach, each worker anonymizes its assigned fragment independently of the other workers. The workers use our distributed version of the Mondrian algorithm, which enforces both  $k$ -anonymity and  $\ell$ -diversity. The algorithm works by recursively partitioning the data into smaller and smaller fragments. The partitioning stops when each fragment contains at least  $k$  records for the same



---

```

ANONYMIZE( $F$ )
1: if no partitioning can be done without violating
    $k$ -anonymity or  $\ell$ -diversity then
2:   generalize  $F[QI]$ 
3: else
4:   let  $a$  be the attribute for partitioning
5:    $R := \{rank(t[a]) \mid t \in F\}$  /* rank of  $a$ 's values in the ordering */
6:   let  $m$  be the median of  $R$ 
7:    $F_1 := \{t \in F \mid rank(t[a]) \leq m\}$ 
8:    $F_2 := \{t \in F \mid rank(t[a]) > m\}$ 
9:   Anonymize( $F_1$ )
10:  Anonymize( $F_2$ )

```

---

Figure 39: Anonymization algorithm for a fragment  $F$

combination of quasi-identifying attributes (to ensure  $k$ -anonymity) and at least  $\ell$  different values for the sensitive attribute (to ensure  $\ell$ -diversity). In our distributed version of the Mondrian algorithm, the workers communicate with each other only to exchange the minimum and maximum values of the sensitive attribute. This allows the workers to anonymize their fragments independently without compromising the privacy of the data

The anonymization phase of our approach has computational complexity  $O(|F| \log |F|)$ , with  $F$  the fragment input to function `Anonymize` in Figure 39. Indeed, the cost of lines 1–8 is  $O(|F|)$ , as discussed in Section 4.4. Due to the recursive calls on a partition of  $F$  including two fragments of size  $\frac{|F|}{2}$ , the overall complexity is  $O(|F| \log |F|)$ , which is in line with the complexity of the original Mondrian algorithm [LDR06].

Figure 39 illustrates the anonymization algorithm executed by each worker for anonymizing the fragment assigned to it. The recursive partitioning (lines 3–10), which operates according to the same logic as multi-dimensional partitioning in the pre-processings phase (Section 4.4), terminates when any further sub-partitioning of a fragment would violate  $k$ -anonymity or  $\ell$ -diversity (lines 1–2). The attribute  $a$  with maximum representativity (Equation 4.1) is chosen for partitioning (line 4). Clearly, since during the anonymization phase each worker  $w_i$  has visibility only on its fragment  $F_i$ , representativity is computed over  $F_i$  (in contrast to the entire dataset  $D$  in Equation 4.1). Like in multi-dimensional partitioning, at the first recursive call, representativity is equal to 1 for all quasi-identifying attributes. Our approach then selects the attribute that has the highest number of distinct values in the fragment. When a fragment  $F$  cannot be further partitioned as this would violate  $k$ -anonymity or  $\ell$ -diversity, the anonymization algorithm produces the anonymized version of  $F$ , obtained generalizing the values of the quasi-identifying attributes to guarantee that all the tuples share the same (generalized) quasi-identifier values (line 2). Our distributed Mondrian approach supports the following generalization strategies.

- *Generalization hierarchies*: applicable to categorical attributes only, the values for attribute  $a \in QI$  are substituted with their lowest common ancestor in the generalization hierarchy  $\mathcal{H}(a)$  defined for  $a$ . To illustrate, consider the dataset in Figure 31(a) and suppose, for simplicity, it is a fragment retrieved by a worker for anonymization. Its anonymized version in Figure 31(c) is



obtained generalizing attribute *Country* according to the generalization hierarchy in Figure 32, considering the partitions in Figure 31(b). For example, values *Italy*, *Italy*, and *France* of the first three tuples (partition at the bottom-left of Figure 31(b)) are generalized to their lowest common ancestor in the hierarchy (i.e., *Europe*).

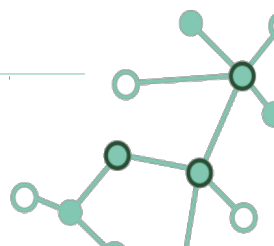
- *Common prefix*: This method of anonymizing data can be applied to both categorical and numerical attributes that are interpreted as strings. To anonymize a value, we first find the longest common prefix of all the values of that attribute. Then, we replace all the characters after the common prefix with a wildcard character. For example, the values 10010, 10020, and 10030 for the attribute ZIP would be generalized to 100\*\*\*. This preserves the common prefix of the values (100) while redacting the last two characters, which makes it more difficult to identify an individual based on their ZIP code.
- *Interval definition*: applicable to numerical attributes defined on a totally ordered domain, the values for attribute  $a \in \text{QI}$  are replaced with a range of values including all of them. The smallest range that includes all the original values is the most natural choice, but larger ranges (possibly predefined) can also be used. Note that this generalization is different from set definition: In set definition, all the original values are explicitly maintained in the set. In interval definition, only the extremes of the range are maintained. To illustrate, consider the dataset in Figure 31(a) and suppose it is a fragment retrieved by a worker for anonymization. Its anonymized version in Figure 31(c) is obtained generalizing attribute *Age* by grouping its values in intervals, considering the partitions in Figure 31(b). For example, values 42, 50, and 43 (fourth, fifth, and sixth tuples, corresponding to the partition on the right-hand-side of Figure 31(b)) are generalized to [42,50].
- *Set definition*: applicable to both categorical and numerical attributes, to anonymize a value, we replace it with the set of all possible values for that attribute. E.g., the values for  $a \in \text{QI}$  are replaced with the set of values including all of them. For example, values *Italy*, *Italy*, and *France* for attribute *Country* would be generalized to the set  $\{\textit{Italy}, \textit{France}\}$ . This makes it impossible to identify an individual based on their country of origin.

## 4.6 Summary and information loss analysis

---

The last phase of our approach is to collect the anonymized fragments and assess the information loss. To do this, each worker stores the anonymized fragment  $\hat{F}$  assigned to it in the storage platform and calculates the information loss caused by the anonymization of that fragment. The information loss of each fragment  $\hat{F}$  is then collected and combined by the master node to assess the information loss of the entire dataset. In the following, we will describe the information loss metrics that we use. For the sake of clarity, we will refer to a dataset  $D$  and its anonymized version  $\hat{D}$ , with the note that each worker operates on the fragment  $F$  (and its anonymization  $\hat{F}$ ) assigned to it.

- *Discernibility Penalty* (DP [LDR06, BA05]) is a measure of how easily a tuple can be distinguished from other tuples in the same equivalence class. The size of the equivalence class  $E$





is the number of tuples that have been generalized to the same values. The larger the equivalence class, the larger the discernibility penalty. This is because a tuple in a large equivalence class is more likely to be indistinguishable from other tuples in the class. The Discernibility Penalty of an anonymized dataset  $\hat{D}$  is computed as follows:

$$DP(\hat{D}) = \sum_{E \in \hat{D}} |E|^2 \quad (4.2)$$

- **Normalized Certainty Penalty (NCP [XWP<sup>+</sup>06])** Normalized Certainty Penalty (NCP) is a measure of the information loss caused by data generalization. NCP assigns penalties to tuples based on the amount of generalization applied to their values. The more generalization applied, the higher the penalty. NCP is applicable to both numerical and categorical attributes: for numerical attributes, generalization can be applied by rounding or binning the values; for categorical attributes, generalization can be applied by collapsing the categories into larger groups. Given a tuple  $t \in D$ , the normalized certainty penalty  $NCP_a(\hat{t})$  of its generalization  $\hat{t} \in \hat{D}$  for attribute  $a \in QI$  is computed as follows:

$$NCP_a(\hat{t}) = \begin{cases} \frac{v_{max} - v_{min}}{Range(a)} & \text{if } a \text{ is numerical} \\ \frac{|Ind(\hat{v})|}{|Dom(a)|} & \text{if } a \text{ is categorical} \end{cases} \quad (4.3)$$

where  $\hat{t}[a] = [v_{max}, v_{min}]$  and  $Range(a)$  is the range of the values in  $a$  if  $a$  is a numerical attribute. When instead  $a$  is categorical and it is generalized using a hierarchy,  $\hat{t}[a] = \hat{v}$  and  $Ind(\hat{v})$  is the set of values in  $Dom(a)$  that could be generalized to  $\hat{v}$ .

Given an anonymized dataset  $\hat{D}$  with quasi-identifier QI, the Normalized Certainty Penalty of  $\hat{D}$  is computed summing the Normalized Certainty Penalties of the attributes in QI for all the tuples in  $\hat{D}$  as follow:

$$NCP(\hat{D}) = \sum_{\hat{t} \in \hat{D}} \sum_{a \in QI} NCP_a(\hat{t}) \quad (4.4)$$

Given the information loss measures  $DP(\hat{F}_1), \dots, DP(\hat{F}_{|W|})$  ( $NCP(\hat{F}_1), \dots, NCP(\hat{F}_{|W|})$ ), resp., where  $|W|$  represents the total number of fragments) for the fragments, the Coordinator computes the information loss for the whole dataset by summing the information loss measures for the fragments. The information loss measures for the fragments can be calculated using either the DP metric or the NCP metric. The DP metric is independent from the amount of generalization applied and only considers the size of equivalence classes, while the NCP metric precisely evaluate the amount of generalization, regardless of the equivalent classes' sizes.

## 4.7 Implementation

In this section, we illustrate the architectural design and the deployment of our distributed anonymization approach.

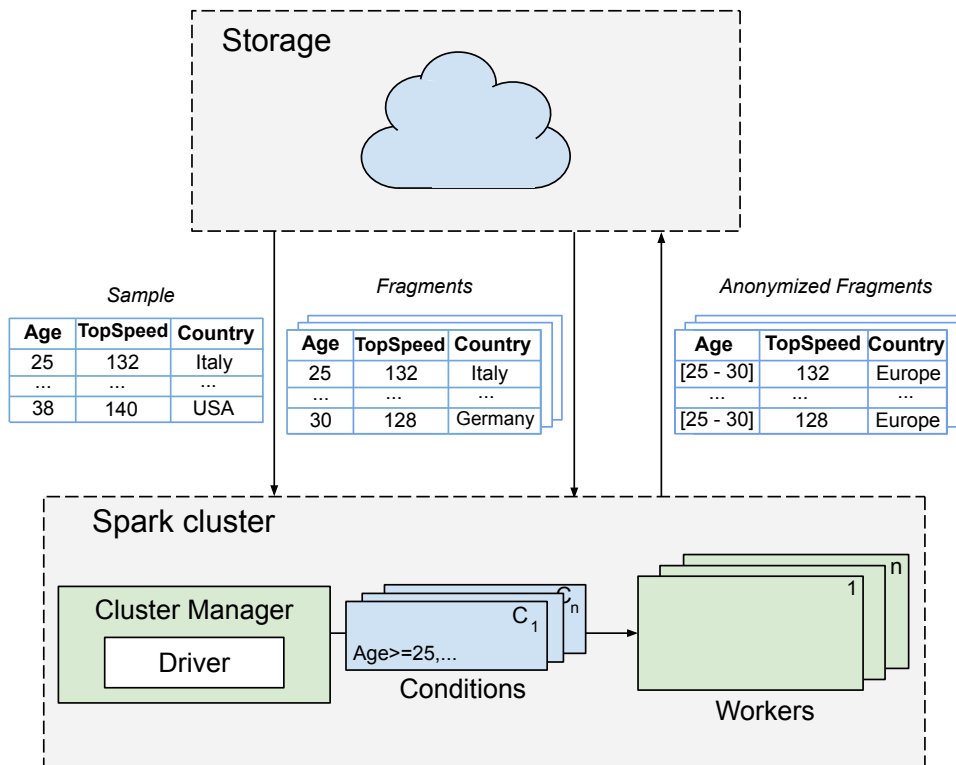


Figure 40: Spark-based distributed anonymization system

### 4.7.1 Architecture

Our implementation uses an Apache Spark cluster to perform the three phases of our approach: pre-processing, anonymization, and wrap-up. An illustration of the components and implementation is shown in Figure 40. The dataset to be anonymized ( $\mathcal{D}$ ) can be stored on any storage platform that is reachable by Apache Spark with a URL. The Spark cluster includes a *Spark Cluster Manager*, which coordinates the cluster, and a set  $W$  of *Spark Workers*, which perform the tasks assigned to them by the Cluster Manager. We built our distributed anonymization application in Python, using the Pandas framework [McK10, R<sup>+</sup>20] to efficiently manage large datasets.

In the Spark cluster architecture (Figure 40), the *Spark Driver* is responsible for coordinating the execution of the anonymization application, playing the role of our Coordinator. It calls the *Spark Context* in the user-written code, which is responsible for partitioning the dataset and distributing it to the Spark Workers. Spark Driver and the Spark Cluster Manager are not necessarily hosted on the same node of the cluster. The Spark Driver then translates the code into jobs, which are further divided into smaller *tasks*. These tasks are then executed by the Spark Workers.

To anonymize a dataset  $\mathcal{D}$ , the Spark Driver first downloads a sample of the dataset, called  $D$ , from the storage platform. The sample must be small enough to fit into the memory of the Spark Driver. The Spark Driver then locally executes the pre-processing phase on the sample  $D$ . Specifically, the Spark Driver locally partitions the downloaded sample  $D$  running procedure  $Q\_Partition$  in Figures 34 (for quantile-based approach), or procedure  $M\_Partition$  in Figure 35 (for



the multi-dimensional approach), keeping track of the conditions describing the computed fragments. After the Spark Driver has completed the pre-processing phase, it defines a set of Spark Tasks ( $|W|$ ). Each task is responsible for anonymizing a fragment  $F_i$  of the dataset. The Spark Driver also includes the conditions that define each fragment  $F_i$  in the task definition. Once the Spark Driver has defined the set of Spark Tasks, it submits them to the Spark Cluster Manager. The Spark Cluster Manager then selects the workers that will be involved in the distributed anonymization process. The workers will execute the Spark Tasks in parallel, anonymizing the fragments of the dataset.

The Spark Driver then sends the anonymization task to each of the identified workers,  $w_i$ . This enables  $w_i$  to download the tuples in the fragment  $F_i$  that is assigned to it. Each Spark Worker  $w_i$  then retrieves the fragment of  $\mathcal{D}$  from the storage platform that satisfies the conditions included in the task assigned to it. The Spark Worker anonymizes the fragment using our distributed Mondrian algorithm (Figure 39, Section 4.5). It then computes the amount of information loss introduced by the anonymization (Equations 4.2 and 4.4, Section 4.5). Finally, the Spark Worker stores the results in the storage platform using the services of the Spark Driver. The Spark Driver combines the information loss computed by the Spark Workers to obtain the information loss of the overall dataset.

## 4.7.2 Deployment

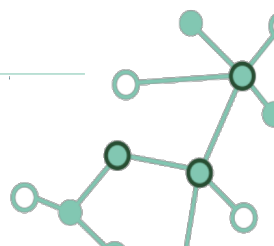
---

We wanted to create a solution that could be easily deployed in a cloud infrastructure, such as AWS or Google Cloud. To do this, we opted for a multi-container application, leveraging Docker containers. This allowed us to deploy our approach as a set of independent containers, each with a specific role. The Spark architecture in Figure 40 was deployed with the following containers:

- *A Docker container for the Spark Driver.* This container is responsible for coordinating the execution of the anonymization application.
- *A Docker container for the Spark Cluster Manager.* This container is responsible for managing the Spark cluster, including the allocation of resources to the Spark Workers.
- *A variable number of Docker containers for the Spark Workers.* These containers are responsible for executing the anonymization tasks.
- *A Docker container to expose a Spark History Server.* This server provides information about the task scheduling and assignment performed by Spark.

This multi-container deployment makes it easy to scale the anonymization application to meet the needs of different datasets and workloads. It also makes it easy to deploy the application in a cloud infrastructure, as each container can be deployed on a separate VM.

In our implementation, we use Docker Compose to spawn Docker containers. To manage the distribution of the containers to the nodes of the Spark cluster, we can use different orchestrator tools, such as Kubernetes. We chose to use Docker Swarm because of its simplicity. Figure 41 illustrates an example of how Docker Swarm distributes Docker containers to the nodes of a Spark cluster. In the figure, white solid boxes represent the nodes in the Spark cluster, while blue boxes represent Docker containers. One of the nodes in the cluster acts as the *Docker Swarm Manager*,



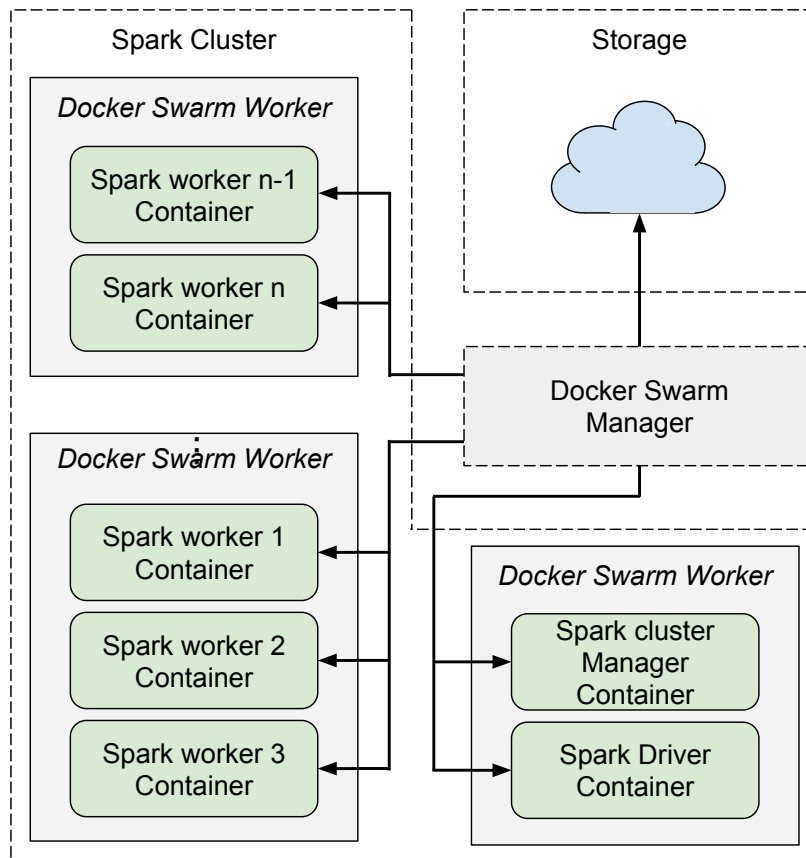


Figure 41: Container deployment in a cloud environment

while the other nodes act as Docker Swarm Workers. The Docker Swarm Manager coordinates and distributes the workload on the Docker Swarm Workers. The Docker Swarm Workers are then used to spawn the Docker containers that model the Spark Manager, Spark Driver, and Spark Workers. One of the Docker Swarm Workers is dedicated to spawning one container for the Spark Cluster Manager and one container for the Spark Driver. The other Docker Swarm Workers spawn the containers that model Spark Workers.

## 4.8 Experimental results

We conducted a series of experiments to assess the scalability and applicability of our distributed Mondrian anonymization approach. We compared our approach to the traditional centralized Mondrian algorithm, which we used as a baseline.

### 4.8.1 Experimental settings

In the following, we describe the settings and dataset used in our experimental evaluation.



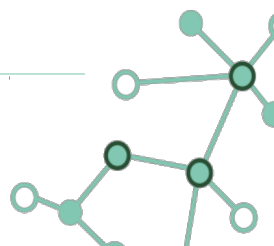
**Cloud server requirements and deployment.** Our solution is cloud-agnostic, so to obtain reproducible results, we simulated a cloud environment using Docker Compose. We ran our experiments on a machine with the following specifications: an AMD Ryzen 3900X (12 physical cores, 24 logical cores) CPU, 64GB of RAM, 2TB of SSD storage, Ubuntu 20.04 LTS with Apache Spark 3.0.1, Hadoop 3.2.1, and Pandas 1.1.3 to manage data. Each worker was equipped with 2GB of RAM and 1 CPU core. Centralized Mondrian relies on 1 CPU core, with no limitation on the use of the RAM. To prove the applicability and scalability of our solution in a real-world distributed environment, we deployed it on Amazon Elastic Compute Cloud (*t2.medium* instances equipped with 2 cores, 4GB of RAM, and 8GB of gp3 SSD, running Ubuntu 20.04 LTS, and located in the *us-east-1* region). The results obtained in this real-world cloud environment confirm the ones obtained in our simulated environment illustrated in Section 4.8.2.

**Storage platform.** We used Hadoop Distributed File System (HDFS) as the distributed storage platform for storing the dataset to be anonymized. We created a Hadoop Distributed File System (HDFS) cluster using Docker containers. The cluster consisted of one container for the Hadoop Namenode and multiple containers for the Hadoop Datanodes. The Hadoop Namenode is responsible for managing the cluster, including storing the filesystem metadata and coordinating access to the data. The Hadoop Datanodes are responsible for storing the actual data blocks and servicing read and write requests.

**Dataset.** We used the Poker Hand dataset [CO07] and ACS PUMS USA 2019 dataset [RFG<sup>+</sup>20] as very large datasets to test the scalability of our approach. We refer the reader to [DFF<sup>+</sup>21c] for performance and information obtained by our solution on the well known (but smaller) ACS PUMS USA 2018 dataset [RFG<sup>+</sup>20].

The Poker Hand dataset is a collection of 1,000,000 records, each of which represents a hand of poker. Each card in a hand is described by two attributes: its seed ( $\{1, \dots, 4\}$ ) and its rank ( $\{1, \dots, 13\}$ ). We considered these attributes to be quasi-identifiers, which are attributes that can be used to identify an individual with a high degree of confidence. We also considered an additional attribute that identifies the entire hand (an integer  $\{0, \dots, 9\}$ ) to be a sensitive attribute.

For the ACS PUMS USA 2019 dataset, we extracted a sample of 1,500,000 tuples. Each tuple of the dataset represents an individual respondent with attributes *ST*, *OCCP*, *AGEP*, and *WAGP*, representing respectively the respondent's US State of residence, occupational status (expressed with a numeric code), age, and annual income. We considered *ST*, *OCCP*, *AGEP* as the quasi-identifier, and *WAGP* as the sensitive attribute. While *OCCP*, *AGEP*, and *WAGP* are numeric attributes, *ST* is categorical. For attribute *ST*, we consider a generalization hierarchy  $\mathcal{H}(ST)$  defined according to the criteria adopted by the US Census Bureau [reg20], where States are at the leaf level of the hierarchy and are grouped in Divisions, which are in turn grouped in Regions. For example, States NJ, NY, and PA (leaf level) can be generalized to MiddleAtlantic (which is their parent in the hierarchy), which in turn can be generalized to Northeast, which in turn can be generalized to US (which is the root of the hierarchy).



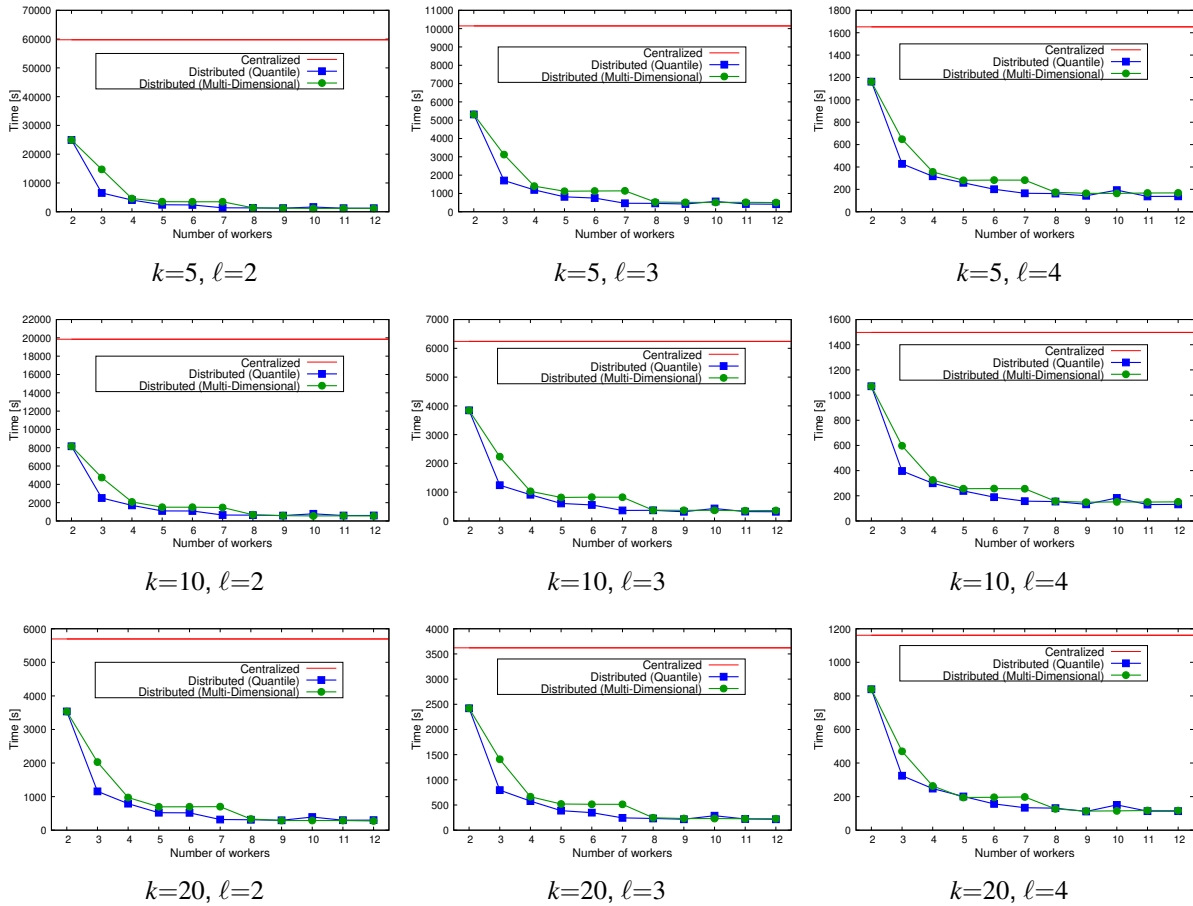
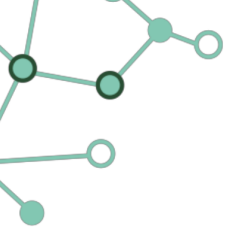


Figure 42: Execution times of centralized Mondrian and distributed Mondrian varying the number of workers,  $k$ , and  $\ell$

## 4.8.2 Results

To assess the scalability of our approach, we analyzed the computation time of our distributed Mondrian algorithm as the number of workers increased. We also analyzed the information loss of our distributed Mondrian algorithm as the size of the sample used for partitioning the dataset among workers increased. Both computation times and information loss values were averaged over 5 runs. We compared the results of our distributed Mondrian algorithm to those of the centralized Mondrian algorithm, which we used as a *baseline*. We report the results for the Poker Hand dataset in more detail, as the evolution of execution time and information loss is more visible for this dataset. However, we note that the results for the ACS PUMS USA 2019 dataset are similar.

**Computation time.** Figure 42 compares the computation times of our distributed Mondrian anonymization algorithm over Poker Hand dataset, using both quantile-based and multi-dimensional partitioning, with centralized Mondrian anonymization, varying the number of workers and parameters  $k$  and  $\ell$ . In particular, we considered  $k$  varying in  $\{5, 10, 20\}$ ,  $\ell$  varying in  $\{2, 3, 4\}$ , and a number of workers between 2 and 12 workers (12 is the largest number of partitions that quantile-based par-

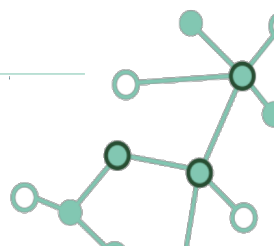


tioning can produce due to the domain of the attributes of the dataset). As expected, the execution time decreases as the number of workers increases. This is because the work of anonymizing the data is distributed across the workers, which reduces the amount of work that each worker has to do. The time reduction with respect to centralized Mondrian is between 28% and 98%, which confirms the scalability of our distributed approach (similar results were observed on the ACS PUMS USA 2019 dataset, with a time reduction between 45% and 98%). The chosen preprocessing strategy, as shown in Figure 42, does not significantly affect the overall execution time. When using two workers, quantile-based and multi-dimensional partitioning exhibit the same execution time because they both perform the same partitioning. However, when using more than two workers, the execution time of quantile-based partitioning decreases with each additional worker, while the execution time of multi-dimensional partitioning only decreases when the number of workers reaches a power of 2 (i.e., it is necessary to have  $2^i$  workers for saving on computation time). When the number of workers is not a power of 2, some workers are assigned twice the workload as the others. This can create a bottleneck, which can slow down the overall process. For example, if there are 6 workers, then 2 of the workers will be assigned twice the workload as the other 4 workers, resulting into marginal performance improvement when using 7 workers instead of 6. However, when the number of workers increases to 8, then all of the workers are assigned the same workload. This can lead to a considerable saving, since the bottleneck is no longer present.

**Information loss.** Our distributed Mondrian algorithm might cause additional information loss compared to the centralized Mondrian algorithm. This is because each worker independently operates on its fragment without coordinating with other workers. Two factors can make the information loss increase: i) the number of workers, and ii) the size of the sample.

Figure 43 compares the average information loss (and its variance) obtained with 5 runs of the distributed Mondrian (with 5 and 10 workers), with respect to the average information loss associated with the centralized Mondrian for computing a  $k$ -anonymous (with  $k = 5$ ,  $k = 10$ , and  $k = 20$ ) 2-diverse version of the Poker Hand dataset, assuming different sampling sizes (0.1%, 0.01%, 0.001%). While in the figure we report only the values obtained with  $\ell = 2$  for Poker Hand dataset, we tested also  $\ell = 3$  and  $\ell = 4$ , also considering ACS PUMS USA 2019, obtaining similar results. The results in Figure 43 show that, for all values of  $k$ , sampling has a very limited impact on information loss. The results also confirm that, for all values of  $k$ , also the number of workers has negligible impact on information loss. More precisely, multi-dimensional partitioning performs similarly for all tested numbers of workers, while quantile-based partitioning produces higher information loss when the number of workers grows. The values in Figure 43 for DP reveal that multi-dimensional partitioning produces equivalence classes similar to the ones produced by centralized Mondrian, while the quantile-based approach produces slightly smaller equivalence classes, especially when the sample used for partitioning is small. The results also show that, in some of the tested scenarios, the values for DP are higher in the centralized scenario. Even if the difference is negligible, it reveals that, when using sampling for partitioning the dataset, (a subset of) the equivalence classes are smaller compared to the equivalence classes obtained without sampling. Smaller equivalence classes, however, do not imply less generalization, as testified by the values of NCP. Indeed, quantile-based partitioning, multi-dimensional partitioning, and the centralized algorithm present similar (small) values for NCP.

The experiments confirm that our distributed Mondrian algorithm is highly scalable and has a lim-



| Sampling    | Partitioning      | Information Loss (DP) |                   | Information Loss (NCP) |                   |
|-------------|-------------------|-----------------------|-------------------|------------------------|-------------------|
|             |                   | 5 workers             | 10 workers        | 5 workers              | 10 workers        |
| 0.1%        | Quantile          | 7.14e06 ± 5.13e02     | 7.10e06 ± 8.31e03 | 1.83e06 ± 3.18e02      | 1.97e06 ± 3.99e02 |
|             | Multi-dimensional | 7.22e06 ± 1.63e04     | 7.22e06 ± 9.25e03 | 1.80e06 ± 2.46e03      | 1.80e06 ± 2.15e03 |
| 0.01%       | Quantile          | 7.14e06 ± 2.78e04     | 7.10e06 ± 9.35e03 | 1.83e06 ± 3.41e03      | 1.96e06 ± 1.01e04 |
|             | Multi-dimensional | 7.17e06 ± 1.93e04     | 7.15e06 ± 1.36e04 | 1.79e06 ± 4.02e03      | 1.80e06 ± 7.44e03 |
| 0.001%      | Quantile          | 7.14e06 ± 2.78e04     | 7.13e06 ± 1.27e04 | 1.83e06 ± 3.41e03      | 1.91e06 ± 7.92e02 |
|             | Multi-dimensional | 7.20e06 ± 5.02e04     | 7.20e06 ± 5.02e04 | 1.80e06 ± 3.82e03      | 1.80e06 ± 3.82e03 |
| Centralized |                   | 7.23e06               |                   | 1.50e06                |                   |

(a)  $k=5, \ell=2$ 

| Sampling    | Partitioning      | Information Loss (DP) |                   | Information Loss (NCP) |                   |
|-------------|-------------------|-----------------------|-------------------|------------------------|-------------------|
|             |                   | 5 workers             | 10 workers        | 5 workers              | 10 workers        |
| 0.1%        | Quantile          | 1.42e07 ± 7.44e03     | 1.41e07 ± 9.94e03 | 2.20e06 ± 1.38e03      | 2.34e06 ± 1.14e03 |
|             | Multi-dimensional | 1.42e07 ± 2.39e04     | 1.43e07 ± 1.47e04 | 2.17e06 ± 6.50e02      | 2.17e06 ± 8.62e02 |
| 0.01%       | Quantile          | 1.42e07 ± 2.16e04     | 1.41e07 ± 7.92e03 | 2.20e06 ± 1.67e03      | 2.34e06 ± 1.10e04 |
|             | Multi-dimensional | 1.42e07 ± 2.35e04     | 1.42e07 ± 1.71e04 | 2.17e06 ± 4.90e03      | 2.17e06 ± 4.29e03 |
| 0.001%      | Quantile          | 1.42e07 ± 2.16e04     | 1.41e07 ± 2.95e04 | 2.20e06 ± 1.67e03      | 2.24e06 ± 8.00e04 |
|             | Multi-dimensional | 1.43e07 ± 2.36e04     | 1.47e07 ± 2.36e04 | 2.17e06 ± 4.67e03      | 2.17e06 ± 4.51e03 |
| Centralized |                   | 1.43e07               |                   | 1.82e06                |                   |

(b)  $k=10, \ell=2$ 

| Sampling    | Partitioning      | Information Loss (DP) |                   | Information Loss (NCP) |                   |
|-------------|-------------------|-----------------------|-------------------|------------------------|-------------------|
|             |                   | 5 workers             | 10 workers        | 5 workers              | 10 workers        |
| 0.1%        | Quantile          | 2.88e07 ± 2.82e04     | 2.86e07 ± 2.06e04 | 2.50e06 ± 3.30e01      | 2.66e06 ± 1.55e03 |
|             | Multi-dimensional | 2.88e07 ± 4.44e04     | 2.88e07 ± 5.77e04 | 2.47e06 ± 1.56e03      | 2.46e06 ± 4.32e03 |
| 0.01%       | Quantile          | 2.88e07 ± 6.73e04     | 2.85e07 ± 4.21e04 | 2.50e06 ± 1.32e03      | 2.65e06 ± 1.76e04 |
|             | Multi-dimensional | 2.88e07 ± 5.03e04     | 2.88e07 ± 1.86e04 | 2.47e06 ± 5.75e03      | 2.46e06 ± 4.00e03 |
| 0.001%      | Quantile          | 2.88e07 ± 6.73e04     | 2.86e07 ± 2.86e04 | 2.50e06 ± 1.32e03      | 2.63e06 ± 3.43e03 |
|             | Multi-dimensional | 2.88e07 ± 7.23e04     | 2.88e07 ± 6.83e04 | 2.47e06 ± 8.32e03      | 2.46e06 ± 6.17e03 |
| Centralized |                   | 2.88e07               |                   | 2.07e06                |                   |

(c)  $k=20, \ell=2$ Figure 43: DP and NCP information loss varying the number of workers and  $k$ 

ited impact on information loss. When the number of workers is a power of 2, both multi-dimensional and quantile-based partitioning have similar computation times. However, when the number of workers is not a power of 2, quantile-based partitioning provides better performance. Both partitioning approaches have a limited impact on information loss, with multi-dimensional partitioning having smaller values for NCP and higher values for DP than quantile-based partitioning (and vice versa).



## 5 Secure Collaborative Computation

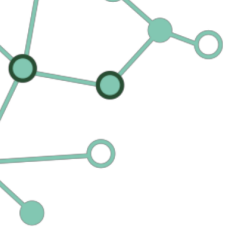
---

In this chapter, we discuss secure collaborative computation which allows data owners to collaboratively perform computations on their joint data that only reveals the result and keeps everything else hidden. Specifically, we focus on training in the context of machine learning (ML) as the computation to perform securely. Collaborative training aims to learn a more accurate model on a larger and more diverse amount of sensitive, distributed data. For example, different hospitals train a model for the early detection of rare diseases on their combined datasets [JAC<sup>+</sup>17].

For businesses collaborations, training on joint business data helps to improve forecasting models such as predictive maintenance in the context of asset performance management or demand forecasting [TTP18]. However, privacy laws such as GDPR [EU18] and CCPA [Leg18] impose strict requirements on sharing sensitive data, potentially hindering collaboration, and data exchange. Additionally, a lack of trust between the collaboration parties (e.g., competing businesses) can impede sharing of unprotected business data due to concerns about data misuse and breaches or allowing inference of business-relevant trade secret or short-term financial outlook that could negatively impact market sentiments. To enable such collaborations, protection solutions based on cryptographic techniques, e.g., secure *multi party computation* (MPC), *homomorphic encryption* (HE) and *masking* [BIK<sup>+</sup>17, MR18], provide formal security guarantees, allowing multiple parties to collaboratively train a model while keeping their respective inputs secret. However, cryptographic techniques do not provide output privacy making the model vulnerable to inference attacks, such as membership inference [SSSS17, YGFJ18] and model inversion [TZJ<sup>+</sup>16, FJR15]. These attacks allow an adversary to recover information on training data or extract model parameters by accessing ML outputs. For example, given the prediction of a Machine Learning as a Service platform, an adversary can recover the training data [TZJ<sup>+</sup>16]. To hinder inference attacks, *differential privacy* (DP) bounds the information leakage from the output (i.e., output privacy) by introducing carefully calibrated noise. In the central model of DP (CDP), clients send raw data to a trusted aggregator, which adds noise to the computation output. To avoid sharing unprotected data with a third party, in the local model (LDP), each client adds noise to its input. LDP has been widely adopted in the industry, e.g., by Google [EPK14] and Apple [Tea17], but it comes at the cost of accuracy. The noise magnitude in LDP can be very high compared to CDP since each party adds its own noise. For Google telemetry data, even billions of users' data fail to identify a common signal from up to 1 million users [BEM<sup>+</sup>17].

*Secure and private collaborative learning* (SPCL) combines DP and cryptography to ensure input secrecy and output privacy. The clients replace the trusted aggregator with cryptographic protocols, removing the need for a trusted party, as in LDP, with accuracy as CDP. Furthermore, cryptography can prevent poisoning attacks [CJG21, SCD<sup>+</sup>21] since it can ensure values are not skewed, e.g., within a fixed range [CSU21]. Distributed differential privacy for collaborative learning is an emerging topic, as shown by the increasing number of contributions in this area, e.g., [KLS21, JWEG18, RXF<sup>+</sup>23, ASY<sup>+</sup>18, BMPB22], and by the first industrial implementation of SPCL for smart-text selection by Google [HK23]. In this system, each client adds non-DP noise to its input that, when aggregated, achieves CDP. The input is protected with masking (described in Sec. 5.2.3).

We are the first to provide a comprehensive and structured analysis of the main cryptographic



techniques and DP mechanisms for collaborative learning. Similar works (Sec. 5.5) focus either only on cryptography [MÖJC23, CHP21], or on analytics [WJS20].

Our contributions are: we identify the phases of SPCL, i.e., setup, protection, and noise generation. We classify the works according to different network architectures (Sec. 5.3.1) (e.g., single-server, multiple servers), and for each architecture, we describe the cryptographic approaches to guarantee input secrecy (Sec. 5.3.2) (e.g., MPC, HE). As the foundational aspect of SPCL, we identify the noise generation and detail different approaches (Sec. 5.3.3) (e.g., secure collaborative sampling in MPC [CGBL<sup>+</sup>17], per-client addition of non-DP noise that is DP in aggregate [KLS21]). For each approach, we describe the underlying cryptographic technique, the DP mechanism (e.g., binomial, discrete Gaussian, gamma), how the noise is sampled, by whom, and which security guarantees are provided. We analyze malicious secure protocols and compare the different noise generation approaches, highlighting how noise generation protocols can be applied to non-DP collaborative learning solutions (Sec. 5.4). Furthermore, we investigate the security of continuous and discrete noise distributions providing a comparison of discrete distributions. Based on our survey of existing works and systematization of their approaches, we point out key observations that identify promising contributions and gaps in the literature that help to guide and advance future research (Sec. 5.6). Next, we provide our work's scope and methodology in Section 5.1 and preliminaries on DP, cryptography, and secure machine learning in Section 5.2.

## 5.1 Scope & Methodology

---

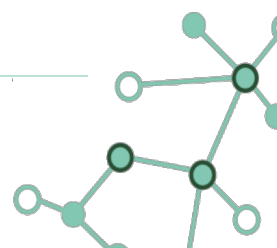
This work aims to systematize the landscape of secure and private collaborative training of ML models. We include works that combine differential privacy and cryptographic techniques (e.g., HE, MPC). The combination of DP and cryptography ensures input secrecy and output privacy.

In this context, we consider works based only on cryptography (e.g., secure aggregation) out of scope since they leak information of the training dataset [SSSS17, YGFJ18]. Works only providing differential privacy are also out of scope since they either require strong trust assumption (in central DP) or result in a non-negligible utility drop (in local DP). Also, we do not further consider solutions based on trusted execution environments (TEEs) since they suffer from side-channels attacks [SWG<sup>+</sup>17, GESM17, LJJ<sup>+</sup>17, LK17, VBWK<sup>+</sup>17, JLLK17, WCLJ18, FYDX21], e.g., timing differences in cache misses – or page faults – reveal sensitive information.

To find relevant works, we used the advanced search on Google Scholar<sup>1</sup> with specific keywords that are a clear reference to our scope, e.g., differential privacy, multiparty or privacy-preserving, collaborative, training. We focused on top-ranked conferences for security, cryptography, and machine learning. We considered a five years time frame, i.e., publication going back to 2018, to focus on recent advancements and current state-of-the-art. Based on the results of our systematic search, we analyzed the abstract and, if in scope, the whole work. Then we applied snowball sampling from the initial set of works, analyzing referenced and referencing papers to include relevant contributions in the literature.

---

<sup>1</sup>Our search also included BASE, but the results were partially overlapped and not as noteworthy.



## 5.2 Preliminaries

---

Next, we introduce preliminary notions on DP, secure ML, and cryptography which serve as building blocks for secure and private collaborative learning.

### 5.2.1 Differential Privacy

---

Differential privacy (DP) is a formal privacy definition that provides strong guarantees for users' private data. Intuitively, differential privacy ensures that the inclusion or exclusion of a user in an analysis does not have a significant impact on the result. Next, we formally define differential privacy.

**Definition 5.2.1** A randomized mechanism  $\mathcal{M}$  satisfies  $(\epsilon, \delta)$  differential privacy if for any neighboring dataset  $D_1, D_2$  (differing on at most one element), and for  $\mathcal{S} \subseteq \text{Range}(\mathcal{M})$ :

$$\Pr[\mathcal{M}(D_1) \in \mathcal{S}] \leq \exp(\epsilon) \times \Pr[\mathcal{M}(D_2) \in \mathcal{S}] + \delta$$

for  $\epsilon > 0, \delta > 0$ , for all possible outputs of  $\mathcal{M}$  ( $\text{Range}(\mathcal{M})$ ).

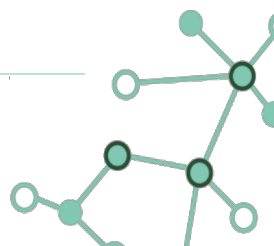
The parameter  $\epsilon$  quantifies the privacy protection, smaller  $\epsilon$  gives more protection, as it bounds the difference of the outputs from  $\mathcal{M}$  on the neighboring datasets  $D_1, D_2$ . The parameter  $\delta$  represents the probability of violating this privacy guarantee. For  $\delta = 0$ , we get pure differential privacy ( $\epsilon$ -DP). In the context of machine learning, typically  $\delta \ll \frac{1}{n}$  where  $n$  is the number of samples in the dataset [DR14, PHK<sup>+</sup>23].

**Local differential privacy.** Note that in Def. 5.2.1, also called *central DP* (CDP), a trusted party requires access to the entire dataset in clear to apply  $\mathcal{M}$ . To avoid relying on a trusted party, Kasiviswanathan et al. [KLN<sup>+</sup>11] define *local differential privacy* (LDP). In LDP, neighboring datasets  $D_1, D_2$  of Def. 5.2.1, are replaced by a pair of input values  $x_1, x_2 \in D$ , from the same dataset  $D$ . LDP is a stricter guarantee than  $(\epsilon, \delta)$ -DP since it requires  $\mathcal{M}$  to give bounded indistinguishable output between any possible pair of data points in the same dataset  $D$  [PHK<sup>+</sup>23]. However, this stronger privacy guarantee comes at the cost of accuracy. In LDP, answering a count query in an  $n$  party setting leads to an error of  $O(\sqrt{n})$  while in CDP, the error is  $O(1)$  [WHMM21]. Bittau et al. [BEM<sup>+</sup>17] introduce the *shuffle model* whose accuracy lies in between CDP and LDP. It leverages a trusted party between the users and the aggregator, the *shuffler*, which is in charge of breaking the correlations between users' messages allowing for reducing the required noise magnitude. We do not consider approaches based on the shuffle model since its accuracy is lower than CDP [CSU<sup>+</sup>19], and it still requires a trusted party or cryptography.

**Differential privacy mechanisms.** To satisfy DP, a mechanism  $\mathcal{M}$  can add noise to a function output<sup>2</sup>. Formally,  $\mathcal{M}(D, f(\cdot)) = f(D) + \mathcal{Z}$ , where  $f : \mathcal{D}^n \rightarrow \mathbb{R}$  is a function applied over a dataset  $D$  and  $\mathcal{Z}$  is the noise sampled from a well-defined distribution. The magnitude of

---

<sup>2</sup>An alternative is a probabilistic selection, i.e., exponential mechanism [MT07], which was not used in surveyed works.





the noise is calibrated to the output range of the function  $f(\cdot)$ . This is defined as the *sensitivity*:  $\Delta_p = \max_{D_1, D_2} \|f(D_1) - f(D_2)\|_p$ , where  $D_1, D_2$  are two neighboring databases and  $p$  defines the type of  $l_p$  norm used. The most common mechanisms are the Laplace and the Gaussian mechanism. The *Laplace mechanism* [DR14] samples a random variable from the zero-concentrated Laplace distribution  $Lap(x; \Delta_1/\epsilon) = \frac{1}{2\Delta_1/\epsilon} \exp(-|x|/(\Delta_1/\epsilon))$ , parametrized with  $l_1$  sensitivity ( $\Delta_1$ ) and  $\epsilon$ . The noisy output obtained with the Laplace mechanism satisfies pure differential privacy ( $\delta = 0$ ). The *Gaussian mechanism* [DKM<sup>+</sup>06] samples a random variable from the Normal distribution  $\mathcal{N}(x; 0, \sigma^2) = \frac{1}{\sigma\sqrt{2\pi}} \exp(-x^2/2\sigma^2)$ . It satisfies  $(\epsilon, \delta)$ -DP, with  $\epsilon < 1$  [DR14] and  $\sigma = \sqrt{2 \ln(1.25/\delta)} \cdot \Delta_2/\epsilon$ . The Gaussian mechanism is widely used in ML as it provides comparable privacy guarantees to the Laplace mechanism with less noise for high dimensional output [PHK<sup>+</sup>23].

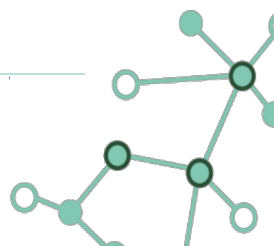
**DP properties.** Differential privacy allows for the composition of multiple mechanisms and is immune to post-processing. The immunity to *post-processing* guarantees that an adversary can not reduce the privacy of a DP mechanism. More formally, any data-independent function  $f$ , applied over the output of an  $(\epsilon, \delta)$ -DP mechanism  $\mathcal{M}$ , can not reduce the privacy guarantees [DR14]. The composition of multiple DP mechanisms remains differentially private, composing them over the same or disjoint datasets. *Sequential composition* [DR14] defines the worst  $(\epsilon, \delta)$  bound for mechanisms applied sequentially on the same dataset  $D$ , e.g.,  $\mathcal{M}_n(\dots(\mathcal{M}_1(D)))$ . Applying  $n$   $(\epsilon_i, \delta_i)$ -DP mechanisms on  $D$ , is equivalent to applying an  $(\epsilon', \delta')$ -DP mechanism on  $D$ , where  $\epsilon' = \sum_{i=1}^n \epsilon_i$  and  $\delta' = \sum_{i=1}^n \delta_i$ . *Parallel composition* [McS09] improves the  $(\epsilon, \delta)$ -bound when different DP mechanisms are applied to disjoint datasets. When applying  $n$   $(\epsilon_i, \delta_i)$ -DP mechanisms  $\mathcal{M}_i$  to  $n$  disjoint datasets, the combination of them satisfies  $(\epsilon', \delta')$ -DP, where  $\epsilon' = \max_{i=1, \dots, n}(\epsilon_i)$  and  $\delta' = \sum_{i=1, \dots, n}(\delta_i)$ . Dwork et al. [DR14] defines a more general form of composition for any number of mechanisms applied in any combination of sequential or parallel composition. This composition is called *advanced composition* and defines a tighter bound for the overall privacy guarantee.

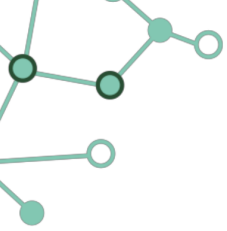
**Alternative relaxed DP definitions.** The composition bound for  $(\epsilon, \delta)$ -DP is still loose even for advanced composition [PHK<sup>+</sup>23]. Different relaxations with tighter bounds for composition have been introduced, such as zero-Concentrated Differential Privacy (zCDP) [BS16] and Rényi Differential Privacy (RDP) [Mir17]. Both definitions can be converted to  $(\epsilon, \delta)$ -DP guarantees [BS16, Propositions 1.3-1.4] [Mir17, Proposition 3]. Abadi et al. [ACG<sup>+</sup>16] introduce the *moments accountant*, a powerful tool for RDP or zCDP definitions, to analyze the privacy loss across training epochs of complex machine learning models. Notably, while  $(\epsilon, \delta)$ -DP defines the worst case privacy loss, which can be infinite with non-zero probability ( $\delta$ ) [KLS21, Mir17], zCDP and RDP do not allow for such privacy breaches as they are defined on the average privacy loss [Des22].

## 5.2.2 Differentially Private Machine Learning

In this section, we introduce the basic notation for ML, in particular supervised learning. Then, we focus on the basics of collaborative learning and how to guarantee privacy.

**Supervised Learning.** Given a dataset  $D = \{(x_i, y_i)\}_{i=1}^N$ , where  $x_i \in \mathbb{R}^d$  is the  $d$ -dimensional input vector and  $y_i \in \mathcal{C}$  the associated label from the label space  $\mathcal{C}$ . The goal of supervised learning is to use  $D$  to learn a function that can predict the labels  $y'$  of unseen samples  $x'$  with high prob-





ability; this process is called training. The function defines a model composed of a set of layers – input, hidden, and output – parameterized with model parameters  $\theta$ . Simple models like logistic regression directly connects the input and output layers. In contrast, more complex models, e.g., neural networks (NN) [Pri23], include one or more hidden layers that apply non-linear transformations. The goal of the training process is to find a set of optimal model parameters  $\theta^*$  that minimizes the loss function  $\theta^* = \arg \min_{\theta} J(\theta)$ . Where  $J(\theta)$  is the average loss function, called *empirical risk*:  $J(\theta) = \frac{1}{N} \sum_{i=1}^N \ell(\theta_i, x_i, y_i)$  and  $\ell(\cdot)$  is a convex loss such as mean squared error. This problem is also known as *empirical risk minimization* (ERM).

**Stochastic Gradient Descent.** Typically, ERM is solved with optimization techniques such as gradient descent (GD). GD updates the model parameter  $\theta$  in the opposite direction of the gradient of the empirical risk:

$$\theta_{t+1} = \theta_t - \eta \nabla J(\theta_t) \tag{5.1}$$

Where  $\theta_{t+1}$  is the model parameter  $\theta$  at step  $t + 1$  of the training process,  $\nabla(\cdot)$  is the gradient operator, measuring the rate of the function change, and  $\eta$  is the learning rate, which determines the size of the optimization step. Gradient descent can be applied to any ML model, from linear models to complex NNs. To allow a faster and more efficient convergence on high-dimensional datasets, *stochastic gradient descent* (SGD) approximates GD applying the descent algorithm to a *random subset* of the training data, named *batch* [Pri23].

## Privacy Guarantees

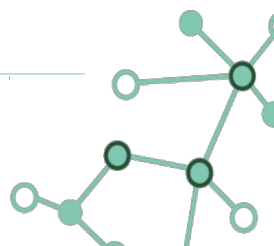
Differential privacy mitigates the impact of inference attacks against ML. ML models suffer from model inversion [TZJ<sup>+</sup>16, FJR15] and membership inference attacks [SSSS17, YGFJ18], which allows an adversary to respectively recover training data and reveal if the training data includes a specific data point. Those attacks work having only *black-box* access to the Model, i.e., querying *MLaaS* platforms that expose prediction API and observing how the output changes for varying input. The information recovered can be sensitive data such as medical conditions or sexual orientation. DP alleviates the effect of those attacks by introducing noise in the training process, ensuring privacy guarantees for the training data and the model itself [PHK<sup>+</sup>23].

**Perturbation mechanisms.** During training, the perturbation inherent to differential privacy can be applied at different steps – as visualized in Algorithm 1 and detailed next.

**PerturbInputs** [DJW13] adds noise to the input data randomizing the sensitive fields. However, noise added to the input data can obscure patterns, making it harder for the Model to learn accurate representations.

**PerturbLoss** [CM08] adds noise to the loss function. It can be difficult to apply, as it assumes a strongly convex loss function, and this is typically not the case in NNs [JE19].

**PerturbGradients** [Dwo06b, SCS13] adds noise to the gradients update. It is widely employed in the literature as it can be applied to any gradient-based optimization method (e.g., SGD).



**PerturbOutput** [PRR10, JE19] adds noise to the model parameters *after* the training process. The noise added is proportional to the sensitivity of the training output [PHK<sup>+</sup>23, CM08] (Sec. 5.2.1). The analysis of the sensitivity bound can be performed only for simple models (e.g., linear regression) due to the complex dependencies between the data and the weights in deep learning (DL) models [PHK<sup>+</sup>23, ZZ<sup>+</sup>12].

---

**Algorithm 1:** Perturbation mechanism in training process (adapted from [JE19])
 

---

**Input:** Training dataset  $\mathcal{D}$ , privacy parameters  $(\epsilon, \delta)$   
**Output:** Model parameters  $\theta$

```

 $\mathcal{D} \leftarrow \text{PerturbInputs}(\mathcal{D})$  /* Option #1 */
 $\theta \leftarrow \text{InitializeModel}()$ 
for each epoch do
  for each batch  $B \subseteq \mathcal{D}$  do
     $L(\theta, B) \leftarrow \text{PerturbLoss}(L(\theta, B))$  /* Option #2 */
     $g \leftarrow \nabla_{\theta} L(\theta, B)$ 
     $g \leftarrow \text{PerturbGradients}(g)$  /* Option #3 */
     $\theta \leftarrow \theta - \eta * g$ 
   $\theta \leftarrow \text{PerturbOutput}(\theta)$  /* Option #4 */
return  $\theta$ 
  
```

---

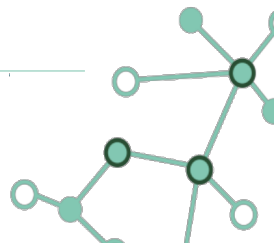
Perturbation options come with different privacy guarantees for each component of the ML pipeline (e.g., training set, Model, prediction labels).

**Differentially Private SGD.** Differentially private SGD (DP-SGD) [SCS13, ACG<sup>+</sup>16] implements **PerturbGradients**. In Algorithm 1, we can expand **PerturbGradients** modifying Eq. (5.1):

$$\theta_t = \theta_{t-1} - \alpha(\nabla J(\theta_t) + \mathcal{Z}) \quad (5.2)$$

where  $\mathcal{Z}$  is a random variable sampled from a suitable distribution (Sec. 5.2.1). The magnitude of additive noise depends on the sensitivity of the gradient computation, i.e., the gradient norm. Since the gradient norm can be unbounded, Abadi et al. [ACG<sup>+</sup>16] introduce *norm clipping*, to bound the gradient norm and limit its sensitivity. They define a clipping parameter  $C$  and clip each gradient in  $l_2$  norm. Formally,  $\nabla J(\theta_t) = \nabla J(\theta_t) / \max(1, \|\nabla J(\theta_t)\|_2 / C)$ . The batch sampling in DP-SGD can enhance users' privacy by having each user sample a subset of data for each training iteration. The uncertainty of the sample being included in a batch allows us to achieve a better accuracy-privacy tradeoff [PHK<sup>+</sup>23, ACG<sup>+</sup>16].

**Differentially Private Distributed Learning.** Here, we focus on the distributed setting, where different users want to train an ML model jointly. *Federated learning* (FL) [MMR<sup>+</sup>17] allows multiple users to keep their data private while collaborating in the training process. The naive implementation of FL training is distributed SGD in which each user performs an SGD locally. The locally computed gradients are shared with the aggregator, a trusted server that aggregates users' local gradients, to update a global model, which is shared with the users. In an alternative approach, called *split*





*learning* [VGSR18], each party  $P_i$  trains a partial NN up to a certain layer. Then  $P_i$  shares the output of this layer with the user or server responsible for the subsequent layers of the NN. Backpropagation starts once the forward pass is completed, and the gradient updates go in the opposite direction. Since none of the surveyed works considers split learning, we focus on FL.

Next, we describe how to make FL differentially private. To make FL differentially private, the aggregator can apply **PerturbGradients** over the aggregated gradients to satisfy CDP. This approach presents a privacy issue as the aggregator has access to the local gradient update of each user. The aggregator can retrieve users' sensitive data, e.g., actively manipulating the aggregated gradient making the local gradients reveal individual data points [BDS<sup>+</sup>21]. To reduce the trust assumption for the aggregator, each user can apply **PerturbGradients** on the local gradient before sending it to the aggregator to satisfy LDP. As discussed in Section 5.2.1, LPD negatively impacts the accuracy since each user adds her noise contribution.

*Distributed noise generation* achieves accuracy close to CDP with privacy guarantees similar to LDP by leveraging cryptography to simulate the aggregator in CDP or to reduce its trust assumption. Next, we introduce cryptographic techniques before detailing distributed noise sampling generation (Sec. 5.3.3).

## 5.2.3 Cryptography

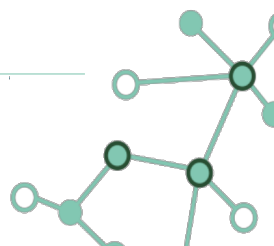
---

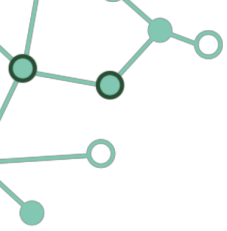
In privacy-preserving collaborative learning, cryptographic techniques serve two primary purposes: to relax or remove the trust assumption for the aggregator and to improve the accuracy by securely generating CDP noise in a distributed fashion. First, we define the security models and then introduce the cryptographic techniques (Sec. 5.2.3).

**Security Models.** We first define our naming convention for the different parties of a collaborative learning protocol. In the context of MPC, there are three different types of parties: *input parties* ( $IP_i$ ), *computing parties* ( $CP_i$ ), and *output parties*. Input parties provide input to the computing parties executing the protocol, and output parties receive the protocol output. In collaborative learning, each IP has its local dataset and can be involved in the training protocol, participating in the computation and in the noise generation. We refer to input parties also as *clients* ( $C_i$ ) and to computing parties also as *servers* ( $S_i$ ).

To define the security guarantees of a cryptographic protocol, we model adversaries bounding their capabilities. There are two main adversarial models: *semi-honest* and *malicious*. The *semi-honest model* considers a passive adversary who tries to infer private information from the protocol execution. In the *malicious model*, an active adversary can manipulate a subset of parties. The adversary can modify their inputs to manipulate the protocol output or recover honest parties' private data. Note that different parties can have different trust assumptions (e.g. semi-honest CP and malicious IP).

While CDP and LDP consider an unbounded adversary, we bound the adversary's capabilities due to our use of cryptography. Specifically, *computational differential privacy* adapts Def. 5.2.1 to consider a bounded polynomial-time adversary adding the term  $negl(\kappa)$ , where  $\kappa$  is the security parameter and  $negl(\cdot)$  is a negligible function, i.e., any function growing more slowly than the inverse of any polynomial. The term  $negl(\kappa)$  represents the probability of breaking a cryptographic





technique, e.g., guessing the decryption key. While  $(\epsilon, \delta)$ -DP is *information-theoretic* secure, even a computationally unbounded adversary can not learn more than what the output reveals (e.g., post-processing), computational DP is only *computationally secure*, i.e., there is a negligible chance to learn more.

## Cryptographic Techniques

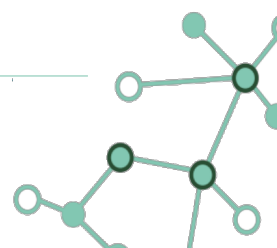
Next, we discuss cryptographic techniques for collaborative learning. Namely, masking, homomorphic encryption, learning with errors, and secure multi-party computation — specifically, garbled circuits and secret sharing.

**Masking.** Bonawitz et al. [BIK<sup>+</sup>17] introduce the masking technique for secure aggregation. As analyzed by [MÖJC23], it is based on two main algorithms to *Mask* and *UnMask* the data, respectively as  $c = (m + k) \bmod r$  and mask  $m = (c - k) \bmod r$ . Here  $k$  is the mask,  $m$  is the data to be masked, and  $r$  is the modulus. The basic idea is that each pair of clients shares a mask and sends the masked input to an aggregator. The aggregator learns only the aggregated output since the masks cancel out when summed. More details are in Sec. 5.3.2.

**Homomorphic Encryption.** Homomorphic Encryption (HE) [Gen09] allows for arbitrary computations on encrypted data. In this context, homomorphic means that an operation in the plaintext space can be mapped to an operation in the ciphertext space. For example, *additive homomorphic schemes* (AHE) [Pai99] permit to compute an encrypted sum  $Enc(a + b)$  based on the encryptions  $Enc(a)$  and  $Enc(b)$  of  $a$  and  $b$ . *Fully homomorphic encryption* (FHE) [CKKS17] schemes enable both multiplication and addition, allowing for the evaluation of arbitrary functions. In a threshold variant of AHE [DJ01, SCR<sup>+</sup>11], the public key is shared among all the parties, and the aggregated result can be decrypted only when  $t$  parties (where  $t$  is the threshold) collaborate to build the secret key.

**Learning with Errors.** Regev [Reg09] introduced the Learning with Errors (LWE) problem, which can be seen as the problem of decoding from a random linear code. The LWE problem is formalized as finding an unknown vector  $s \in \mathbb{Z}_q^n$  such that  $b = As + e \in \mathbb{Z}_q^m$ . Where  $A \in \mathbb{Z}_q^{m \times n}$  is a matrix built from  $m$  random vectors  $a_i \in \mathbb{Z}_q^n$ , and  $e \in \mathbb{Z}_q^m$  is the error vector. The challenge of LWE comes from the  $e$  entries which disrupt the linear relation among the equations and are sampled from a suitable probability distribution on  $\mathbb{Z}_q$ . Notably, cryptographic schemes based on LWE are additively homomorphic for both the key and the message, i.e., given two messages  $(m_1, m_2)$ , an encryption algorithm  $Enc$  and two keys  $(s_1, s_2)$ ,  $Enc(s_1, m_1) + Enc(s_2, m_2) = Enc(s_1 + s_2, m_1 + m_2)$  [BGL<sup>+</sup>22].

**Secret Sharing.** A  $t$ -out-of- $n$  secret sharing scheme  $((t, n)$ -SS) splits a secret into  $n$  parts, called *shares*, and distributes the shares to  $n$  parties. At least  $t$  shares are required to reconstruct the secret. Note that all the operations in the context of secret sharing are defined as modular operations, and we omit the modulo for simplicity.  $(t, n)$ -SS is defined as a pair of two algorithms, share ( $Shr$ ) to generate shares from a secret, and reconstruct ( $Rec$ ) to recover the secret from the shares. The pair  $(Shr, Rec)$  satisfies two properties: *correctness*, i.e.,  $Rec$  outputs the correct secret given any  $t$  shares, and *perfect secrecy*, i.e., less than  $t$  shares reveal nothing about the







secret [EKR<sup>+</sup>18]. A  $(t, n)$ -SS scheme is secure against the collusion of at most  $t - 1$  parties. The parameter  $t$  is the threshold, and  $(t, n)$ -SS is also called threshold secret sharing scheme.

The surveyed works focus on linear secret sharing schemes, where  $Rec$  is a linear mapping, and any linear operation on the shares reflects in the reconstructed secret. We define two concrete linear SS schemes, *additive secret sharing* and *Shamir's secret sharing*. Additive secret sharing is a  $(n, n)$ -SS scheme, where to reconstruct the secret  $s$ , we need to sum all the shares, i.e., given secret  $s \in D$ ,  $Shr(s) = (s_1, \dots, s_{n-1}, s_n) = (r_1, \dots, r_{n-1}, s - \sum_{i=1}^{n-1} r_i)$  where  $r_i$  are random numbers, and  $Rec(s_1, \dots, s_n) = \sum_{i=1}^n s_i$ . *Shamir's secret sharing* is a  $(t, n)$ -SS which hides a secret  $s$  in the zero point of a random polynomial  $\mathcal{P}$  of degree  $t - 1$ . Each party  $P_1$  holds a point  $\mathcal{P}(i)$ , where  $i > 0$ , which is a *share*  $s_i$ . To reconstruct the polynomial to recover  $s = \mathcal{P}(0)$ , Shamir's secret sharing uses Lagrange interpolation on  $t$  points [EKR<sup>+</sup>18].

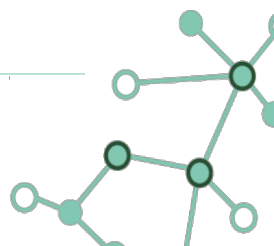
**Garbled Circuits.** Garbled circuits (GC) [Yao86] lets two parties securely evaluate any polynomial-time function  $f$  expressed as a Boolean circuit  $C$ . Garbled circuits are defined by two algorithms *Garble* and *Eval* performed by party  $P_1$  (called *Garbler*) and  $P_2$  (called *Evaluator*), respectively. Garbled circuits fulfill two main properties: *correctness*, which ensures that the output of the garbled circuits must be exactly the output of the function  $f$ , and *secrecy*, which ensures that only the output of the GC is revealed to  $P_2$ . The evaluator cannot learn any other information about the intermediate outputs and the input of  $P_1$ . To let the evaluator only learn the garbling of her input and nothing else,  $P_1$  and  $P_2$  implement *oblivious transfer* (OT) [Rab05].

## 5.3 Secure and Private Distributed Learning

---

This section systematizes approaches combining DP and cryptography for secure and private collaborative learning. Table 3 in Sec. 5.4 summarizes key features of analyzed works. This section is organized as follows; first, we define different architectures to classify the works (Sec. 5.3.1). Then, we describe the cryptographic approaches for all architectures (Sec. 5.3.2). Finally, we identify how to securely generate noise (Sec. 5.3.3). Unless explicitly stated otherwise, all the approaches train NN with DP-SGD applying **PerturbGradients** and linear models with **PerturbOutput**. Servers and clients are semi-honest unless explicitly stated otherwise, and we analyze the malicious secure protocols in Sec. 5.4.1. We generalize all works by fixed phases before proceeding.

**Protocols phases.** Within all the protocols, we identify five main phases. The **setup** phase shares cryptographic and DP parameters. Then, each client  $C_i$  computes its **local update** and ensures the **protection** of the update via, e.g., cryptography. The parties (clients or servers) execute a **noise generation** protocol adding noise to their local update. As the last step, a server (or more servers) performs **aggregation** on the noisy local updates. By update, we mean gradient update for **PerturbGradients** and model parameters update for **PerturbOutput**. The different phases can be combined and executed by different parties, depending on the architecture, the cryptographic techniques, and the noise generation protocol.



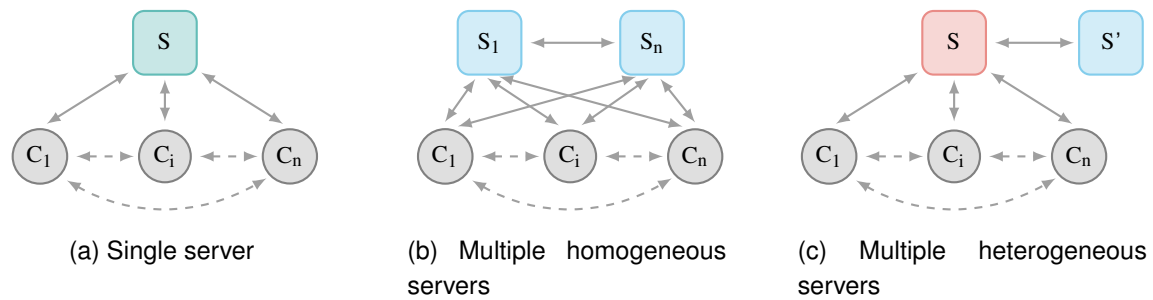


Figure 44: Architectures for secure and private collaborative learning, optional communication channels have dashed lines.

### 5.3.1 Architectures

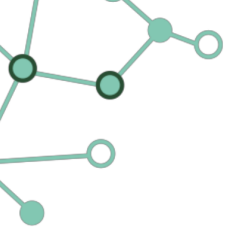
From our study, we identify three types of architectures, i.e., single server, multiple homogeneous servers and multiple heterogeneous servers, depicted in Figure 44. The architectures are mainly distinguished by the number of servers, the roles of the servers, and the communication patterns. Next, we describe the architectures in detail.

**Single server.** Figure 44a shows a single server  $S$  in charge of the aggregation. Here, we distinguish two types of client communications: *direct* communication between clients [ASY<sup>+</sup>18, MPBB19, KLS21, BMPB22, SSV<sup>+</sup>22] (i.e., requiring communication channels between all clients) and *indirect* communication, where the clients interact only with the server, i.e., identifying a star topology [BRB<sup>+</sup>17, TBA<sup>+</sup>19]. In the star topology, the server acts as a relay, i.e., forwarding messages to other clients.

**Multiple homogeneous servers.** Figure 44b shows an architecture with two or more servers with the same role. Here, we distinguish two approaches: *partial outsourcing* and *full outsourcing*. In partial outsourcing, the servers perform the aggregation phase and can also be in charge of the noise generation [CGBL<sup>+</sup>17, JE19]. In full outsourcing, the clients can act as computing parties and execute the entire protocol in MPC [RXF<sup>+</sup>23], or secret share their data to a set of servers [PRM<sup>+</sup>22].

**Multiple heterogeneous servers.** Figure 44c shows servers with different roles, e.g., a set of server acts as aggregators, and another set adds DP noise. This architecture seems to have gotten less attention in the recent literature [CCDD<sup>+</sup>21, FTPSH20] as we observed this setting only in two recent works.

We further distinguish works having the same architecture based on their implemented cryptographic techniques and differential privacy mechanisms. Next, we describe the different cryptographic approaches for each architecture (Sec. 5.3.2), highlighting the primitives and the communication rounds required. Then, we discuss the DP mechanisms with respect to the privacy guarantees, the type of noise and how it is combined, and for which architecture the mechanism is feasible (Sec. 5.3.3).



## 5.3.2 Cryptographic approaches

---

In this section, we focus on the protection phase describing the cryptographic aspects of the different works per architecture. The cryptographic techniques described here guarantee confidentiality for the local updates.

**Single server.** Works with direct communication among clients implement masking to guarantee the protection of the local updates. Most of the protocols [BMPB22, MPBB19, ASY<sup>+</sup>18, KLS21] are based on *secure aggregation* (SecAgg) [BIK<sup>+</sup>17]. Recall, in Sec. 5.2.3, each pair of clients  $(C_i, C_j)$  has to share a common mask. In the setup phase, each  $(C_i, C_j)$  runs a Diffie-Hellman (DH) key exchange protocol [DH22] to establish a common seed. This seed is the input of a pseudo-random number generator to generate the masks, avoiding exchanging the masks for every protocol iteration. Each client  $C_i$  computes the local update, adds the mask established with  $C_{j>i}$ , and subtracts the masks established with  $C_{j<i}$ . When the server aggregates the contributions from each party, the masks cancel out.

To reduce the communication costs of the local updates, some works [ASY<sup>+</sup>18, KLS21] leverage quantization to map the updates to a discrete set of values. Quantization introduces some errors which result in a trade-off between communication and accuracy [KLS21].

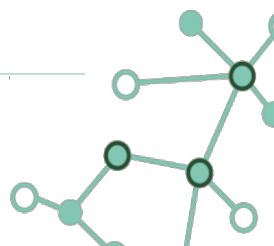
To minimize the number of communication rounds of SecAgg, Stevens et al. [SSV<sup>+</sup>22] rely on LWE for masking to avoid DH key exchange. Recall, in LWE (Sec. 5.2.3), an error vector  $e$  is introduced to break the linearity of a set of equations  $b = As + e$ , where the matrix  $A$  is publicly known by all the input parties and  $s$  is the secret vector. In Stevens et al. [SSV<sup>+</sup>22], each client  $C_i$  samples its secret  $s_i$  and the error vector  $e_i$  from a discrete Gaussian distribution.  $C_i$  masks the local update  $v_i$  with  $b_i$ , resulting in  $h_i = v_i + b_i$  where  $b_i = As_i + e_i$ , and sends  $h_i$  to the server  $S$ .  $S$  aggregates the local updates obtaining  $h_{sum} = b_{sum} + v_{sum}$  where  $b_{sum} = As_{sum} + e_{sum}$ . To remove the term  $As_{sum}$  and retrieve the noisy global update  $v_{sum} + e_{sum}$ , each client secret shares, via *packed Shamir secret sharing* [FY92], its  $s_i$  to all the clients<sup>3</sup>. A set of  $t$  clients reconstruct  $s_{sum}$  and send the value to  $S$ . Stevens et al. [SSV<sup>+</sup>22] expand the secret reconstruction protocol to malicious security via verifiable secret sharing [Ben00] (Described in Sec. 5.4.1).

Truex et al. [TBA<sup>+</sup>19] and Bindschaedler et al. [BRB<sup>+</sup>17] consider star topology protocols implementing a threshold variant of the AHE Paillier scheme [DJ01, DJN10]. In Truex et al. [TBA<sup>+</sup>19], each party sends the homomorphically encrypted local update [DJ01] to the server. The server sums all the updates and selects at least  $n - t - 1$  clients (assuming at most  $t$  colluding clients) to reach the threshold for the decryption of the aggregated update. Bindschaedler et al. [BRB<sup>+</sup>17] combines the Paillier scheme [DJN10] with Shamir's secret sharing. Each client encrypts the polynomial evaluation of  $n$  points using the public keys of the other parties. In the aggregation phase, each client can decrypt the polynomial point encrypted with their key and send back the plain texts to the server. Bindschaedler et al. [BRB<sup>+</sup>17] implements a malicious secure protocol for partial noise aggregation that we discuss later (Sec. 5.4.1).

Overall, the cryptographic approaches in single-server architecture can be applied to multi-

---

<sup>3</sup>Bell et al. [BGL<sup>+</sup>22] stated that the reconstruction protocol for the secret  $s$  could reveal some information about  $s$ , breaking the confidentiality of the protocol.



ple models, ranging from linear models to deep neural networks, applying **PerturbGradients** or **PerturbOutput** accordingly.

**Multiple homogeneous servers.** Here (Fig. 44b), multiple servers replace the single aggregator running an MPC protocol. Recall that in the partial outsourcing approach, the clients compute the local update, and the servers perform the aggregation. A common approach is to ensure confidentiality by secret sharing the local updates to two servers, which perform the aggregation and the noise generation via GCs. Jayaraman et al. [JWEG18] implement Yao's GC [Yao86] with the framework of Tian et al. [TJGE16]. They implement this approach for both **PerturbOutput** and **PerturbGradients** with linear models but can be extended to any ML model.

In CPGD, Chase et al. [CGBL<sup>+</sup>17] train a NN implementing an optimization of Yao's GC by Araki et al. [AFL<sup>+</sup>16]. In Pentylala et al. [PRM<sup>+</sup>22], the clients can be active computing parties or outsource their computation to a set of computing parties. In the second case, in the setup phase, the clients secret share their data to the computing parties. Their solution applies to the DP training of logistic regression leveraging **PerturbGradients**. They implemented the protocol for up to 4 computing parties with the MPC framework MP-SPDZ [Kel20]. Their approach also works for vertically partitioned data, where different attributes of the same user are stored in different datasets.

PEA, by Ruan et al. [RXF<sup>+</sup>23], implements DP-SGD leveraging linear secret sharing [Sha79, GMW19] in which each client is also a computing party. Ruan et al. [RXF<sup>+</sup>23] propose two optimizations to improve the accuracy and efficiency of secure multiparty training: data-independent feature extraction and a global model initialization protocol. PEA has two phases: in the local phase, each party performs the computation in clear on their local dataset, whereas in the global phase, the computation is performed on secret shares of their joint global dataset. In the local phase, each party performs a data-independent feature extraction on their dataset leveraging foundational models. Then, each party trains a local model via DP-SGD [RXF<sup>+</sup>23, Algorithm 1]. The global phase starts with the global model initialization protocol [RXF<sup>+</sup>23, Protocol 4], where parties generate candidate models aggregating the local ones. Then, the parties jointly evaluate the accuracy of all the candidate models initializing the global model with the most accurate candidate model. The global model is then trained using MPC of DP-SGD [RXF<sup>+</sup>23, Protocol 3]. PEA introduces a new protocol for the inverse square root [RXF<sup>+</sup>23, Protocol 1] to clip the  $l_2$  norm of a gradient vector to add DP noise based on linear secret sharing. Key features are feature extraction, which reduces the dimensionality of the data and allows training simpler models, and global model initialization, which reduces the number of iterations for the training algorithm.

**Multiple heterogeneous servers.** In contrast to the multiple homogeneous servers architecture, the servers here (Fig. 44c) have distinct roles. CAPC [CCDD<sup>+</sup>21] is a secure and private collaborative labeling protocol that extends PATE [PSM<sup>+</sup>18], enhancing security and privacy guarantees. In a collaborative labeling protocol, a party  $St$ , namely *student*, queries a set of parties  $T_i$ , *teachers*, to label its dataset to be used for training. In CAPC, an additional server, the Privacy Guardian (PG), is responsible for DP noise sampling. All teachers perform inference using their local model, and then the label is set using majority voting. CAPC implements a 2-party secure inference protocol based on FHE [BLCW19].  $St$  homomorphically encrypts its data  $x$  and sends  $Enc(x)$  to each  $T_i$  who starts the inference protocol. After the inference each teacher  $T_i$  holds the homomorphically encrypted label  $Enc(r_i)$ , she subtracts a random value  $\hat{r}_i$  and send to  $St$  the value  $Enc(r_i - \hat{r}_i)$ .  $St$

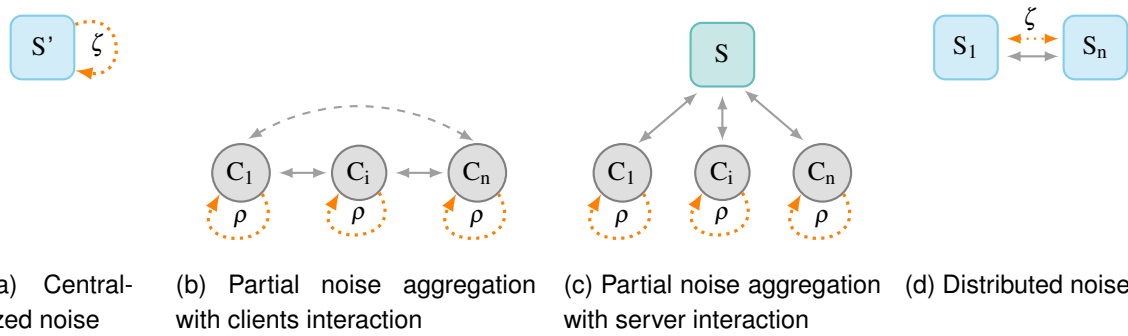


Figure 45: Typical distributed noise generation scenarios where dotted orange arrows indicate noise sampling, and the sampling of central/partial noise is shown as  $\zeta/\rho$ , respectively. Optional interactions have dashed lines.

decrypts, obtaining  $r_i - \hat{r}_i$ , and runs Yao's GC with each  $T_i$ , where  $S_t$  and  $T_i$  get respectively  $s_i$  and  $\hat{s}_i$  where  $s_i + \hat{s}_i = \text{OneHot}(\arg \max_{\text{labels}}(r_i))$ , which is the one-hot-encoding of the label  $r_i$ . Each  $T_i$  sends its  $\hat{s}_i$  with the PG, which aggregates all the  $\hat{s}_i$  received and adds DP noise obtaining  $\hat{s}$ .  $S_t$  aggregates all  $s_i$ , obtaining  $s$ , and run Yao's GC protocol with the PG. From the GC,  $S_t$  receives  $\arg \max(s + \hat{s})$ , i.e., majority vote over the labels, and PG receives nothing.

Froelicher et al. [FTPSH20] presents Drynx, which allows external *querier* to train logistic regression models. Drynx contains different types of servers, *computation parties* (CP) and *verification parties* (VP). The CPs are arranged in a tree-based hierarchy for aggregation and noise generation. Through VPs, Drynx achieves malicious security against  $n - 1$  colluding CPs (Sec. 5.4.1). The aggregation is based on an AHE elliptic curve ElGamal scheme [EIG85] with key homomorphism, i.e., given  $\text{Enc}(k_1, x_1) + \text{Enc}(k_2, x_2) = \text{Enc}(k_1 + k_2, x_1 + x_2)$ . Each CPs generate a public key pair, and the clients encrypt their local update with the aggregated public key. The CPs aggregate the local update from lower levels in the hierarchy until the root. To make the model accessible only to the querier, the root of the hierarchy re-encrypts the aggregated update under the public key of the querier. One CP executes an oblivious noise selection for DP (Sec. 5.3.3).

### 5.3.3 Distributed Noise Generation

In this section, we focus on the noise generation phase of the protocols outlining four typical noise scenarios. All but one surveyed works provide CDP by leveraging cryptographic techniques (Sec. 5.3.2). Only Mugunthan et al. [MPBB19] provide LDP via cryptography using partial noise aggregation to provide resistance for up to  $n - 1$  colluding parties.

We provide some helpful preliminary notions used in secure noise generation protocols. *Inverse transform sampling* (ITS) allows sampling random variables  $\mathcal{Z}$  from a desired probability distribution  $f(x)$ . ITS has as input a random variable  $u \sim U(0, 1)$  and returns  $\mathcal{Z} = \text{CDF}^{-1}(u)$  with  $\mathcal{Z} \sim f(x)$  where  $\text{CDF}^{-1}(\cdot)$  is the inverse *cumulative density function* of  $f(x)$  and  $U(0, 1)$  is the uniform distribution in the range  $(0, 1)$  [Law]. The *Box-Muller transform* [BM58] is an alternative method to ITS to sample  $Z$  from the standard normal distribution  $\mathcal{Z} \sim N(0, 1)$  in a more computationally efficient way. We define as *secure noise sampling* a 2-party protocol in which each party samples a random

variable  $(x_1, x_2)$ . The two variables are *XOR*ed to get a random variable  $x = x_1 \oplus x_2 \sim U(0, 1)$  which is given in input to ITS (or Box-Muller transform). The XOR ensures that if at least one party is honest, the result  $x$  is correctly sampled. We denote as *oblivious noise selection* any multiparty noise generation protocol in which none of the parties knows its noise contribution.

Next, we describe three different approaches for CDP noise generation (Fig. 45): centralized noise sampling, partial noise aggregation, and distributed noise sampling. For each approach, we distinguish the type of noise, how much noise each party adds, and the type of DP guarantee (e.g., CDP, RDP, zCDP). Finally, we describe partial noise aggregation for LDP.

**Centralized noise sampling.** A single party samples and adds CDP noise as shown in Figure 45a. In CAPC [CCDD<sup>+</sup>21], this central party is the privacy guardian, which samples noise from a Gaussian distribution and adds the noise to the secret shares of the aggregated labels. With this mechanism, the PG can not infer any label information.

**Partial noise aggregation.** Each party samples an amount of noise, adding it to the local update. The noisy updates are not differentially private, as the partial noises alone are insufficient to guarantee DP, but in sum, they satisfy CDP. Threshold secret sharing and threshold variants of HE ensure the confidentiality of the input until  $t$  ciphertexts are combined. With  $n$  parties in the protocol, assuming up to  $t$  colluding parties, we have to select the amount of noise by each party accordingly. For  $t = 0$ , the sum of the partial noises achieves CDP, but for  $t = n - 1$ , each party has to add LDP noise to protect its input. Partial noises can lie between those two extremes, i.e., requiring less noise than LDP but still more than CDP. Aggregating partial noises provides better accuracy than LDP since the aggregated model update carries less noise. All single-server architecture (Fig. 44a) works consider partial noise, and only PEA [RXF<sup>+</sup>23] has multiple servers (Fig. 44b).

In the following, we focus on approaches sampling noise from a continuous distribution., i.e., distributed version of the Gaussian and Laplace mechanism. Recall, the sum of  $N$  independent Gaussian random variables  $\mathcal{N}_i(\mu_i, \sigma_i^2)$ , is still a Gaussian  $\mathcal{N}(\sum_{i=1}^N \mu_i, \sum_{i=1}^N \sigma_i^2)$ . To achieve  $(\epsilon, \delta)$ -DP, the sum of the Gaussian should have a variance  $\sigma_{DP}$  (Sec. 5.2.1). Truex et al. [TBA<sup>+</sup>19] define the variance  $\sigma_i^2$  that each  $\mathcal{N}_i$  must have, as:

$$\sigma_i^2 \geq \frac{1}{n-t-1} \sigma_{DP}^2 \quad (5.3)$$

Note that the  $-1$  in Equation (5.3) can be omitted depending on who receives the noisy outputs. An input party (who adds partial noise) which is also an output party (who receives the noisy output) can remove its own noise [HLK<sup>+</sup>17]. Such potential noise removal reduces the DP guarantee and requires adding an additional partial noise term, so the result remains DP even if the party removes its noise. However, if the input parties are not output parties, or the party selects noise obliviously, the  $-1$  can be removed.

A distributed version of the Laplace mechanism can be realized via independently sampled gamma random variables. A Laplace random variable (*Lap*) can be generated from the sum of the difference of  $n$  i.i.d. random variables from the gamma ( $\gamma$ ) distribution, i.e.  $Lap(0, \lambda) = \sum_{k=1}^n \gamma_k - \gamma'_k$  [KKP01, Table 2.3][GX15]. Due to the assumption of up to  $t$  colluding parties, there could be more than  $N$  gamma random variables summed, which increases the error as analyzed by [AC12].

The works in [MPBB19, BMPB22, BRB<sup>+</sup>17] implement oblivious noise selection from a gamma



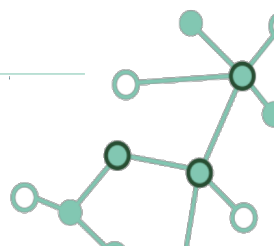
distribution. In SMPAI [MPBB19], each client  $C_i$  has to choose between two masked gamma variables  $(\gamma_0^*, \gamma_1^*)$  sent by a neighboring client (Fig. 45b). Then  $C_i$  samples its own gamma random variable  $\gamma$  and sends to the server the local update plus the difference of the two gamma terms  $(\gamma_i^* - \gamma)$ , where  $i = \{0, 1\}$  and is randomly selected. In Byrd et al. [BMPB22] each client sends two masked gamma terms  $(\gamma_0^*, \gamma_1^*)$  to the server who shuffles and forwards them to a client (Fig. 45c). Then it proceeds as SMPAI [MPBB19]. In Bindschaedler et al. [BRB<sup>+</sup>17], each client samples  $c$  gamma noises. To achieve CDP, the server  $S$  has to select  $n/m$  gammas for each client (Fig. 45c). To this purpose,  $S$  generates  $n$  selector vectors, composed of  $\{0, 1\}$ , where the 1 is used to select a noise value. The selector vectors are homomorphically encrypted and sent to clients. Each client multiplies the encrypted selector vectors by their noise vector to get the homomorphically encrypted gamma noise the server will add. Since the server could decrypt those noise terms, the client masks the encrypted noise that will be unmasked only when aggregated. Byrd et al. [BMPB22] and Bindschaedler et al. [BRB<sup>+</sup>17] are resistant against  $n - 1$  colluding clients since even if the colluding clients remove their local update, they only learn the noisy local update of the honest parties with CDP guarantee.

CPSGD [ASY<sup>+</sup>18] and PEA [RXF<sup>+</sup>23] implement the *binomial mechanism* [DKM<sup>+</sup>06] which samples binomial random values (Fig. 45b). Note that the Gaussian noise can be approximated with binomial noise. If the clients hold enough shares of many unbiased coins, they can sum them to obtain a share of approximately correct Gaussian noise [DKM<sup>+</sup>06]. Formally, with  $m$  tosses of an unbiased  $\pm 1$  coin are tallied and divided by 2, with  $m \geq 64 \log(2/\delta)/\epsilon^2$  [DKM<sup>+</sup>06]. The straightforward solution proposed by Dwork et al. [DKM<sup>+</sup>06] implements verifiable secret sharing [DFK<sup>+</sup>06] to generate shares of  $\{0, 1\}$  coins, then converts the  $\{0, 1\}$  shares to share of integers, and then sum the shares. The binomial mechanism guarantees *Byzantine resistance*, so the total variance for the noise is sufficient to provide CDP with at least  $2/3$  honest parties. CPSGD [ASY<sup>+</sup>18] combines the binomial mechanism with the quantization of the local gradients to improve communication costs. In PEA, Ruan et al. [RXF<sup>+</sup>23] implement two techniques to reduce the noise, feature extraction, which allows training simpler models and global model initialization to achieve a faster convergence (Sec. 5.3.2).

Kairouz et al. [KLS21] and Stevens et al. [SSV<sup>+</sup>22] both implement the discrete Gaussian mechanism [CKS20] (Fig. 45b). Here, the noise is sampled from a discrete Gaussian  $\mathcal{N}_{\mathbb{Z}}(x; \mu, \sigma^2) = g(x) / \sum_{y \in \mathbb{Z}} g(y)$  where  $g(x) = \exp(-(x - \mu)^2 / 2\sigma^2)$ . Note that the sum of discrete Gaussians does not result in a discrete Gaussian [KLS21, Sec. 3]. However, the sum still satisfies  $(\alpha, \epsilon)$ -RDP. The work by Stevens et al. [SSV<sup>+</sup>22] uses a noise-based cryptographic technique to realize the noise-dependent privacy guarantee. Recall, Stevens et al. [SSV<sup>+</sup>22] uses the LWE to mask the local updates. The error vector  $e_i$  is sampled by each client from a discrete Gaussian distribution. The summation of  $e_i$  guarantees RDP for the aggregated gradient  $v_{sum} + e_{sum}$ .

**Distributed noise sampling.** Next, we describe approaches to sample noise securely via MPC. The MPC protocol replaces the aggregator of CDP. It adds the noise once after the aggregation, leading to a more accurate model compared to partial noise aggregation [JWEG18]. Due to MPC, distributed noise sampling requires multiple servers architecture (Fig. 45d).

Chase et al. [CGBL<sup>+</sup>17] and Jayaraman et al. [JWEG18] aggregate local updates and sample noise within a garbled circuits [AFL<sup>+</sup>16] (Fig. 45d). Both leverage secure noise sampling, i.e., via





*XOR*. Jayaraman et al. [JWEG18] samples Laplace noise via ITS, for **PerturbOutput**, and Gaussian noise via Box-Muller transform [BM58], for **PerturbGradients**. Since **PerturbGradients** is iterative, Jayaraman et al. [JWEG18] consider zCDP to achieve a lower privacy loss thanks to moments accountant.

Pentyala et al. [PRM<sup>+</sup>22] leverage MPC to add a noise vector to the secret-shared model coefficients in logistic regression training. At the end of the noise generation, the noisy coefficients of the model are secret shared between the parties that are oblivious to the noise in their shares and combine them in the final model. The noise is sampled from the gamma distribution, and the protocol implemented applies **PerturbOutput** by [CM08]. The noise generation protocol [PRM<sup>+</sup>22, Protocol 2] is executed by all the parties via MP-SPDZ [Kel20]. First, the parties generate a  $d$ -dimensional vector of standard Gaussian noise via Box-Mueller transform [BM58], applying secure noise sampling. Then, the vector entries are normalized by entry-wise division with its  $l_2$  norm. Finally, the vector entries are multiplied by the gamma random variable to obtain the gamma-distributed noise term. The gamma-distributed noises are secret shared, and the shares are obtained by adding exponential random variables sampled via ITS to the vector entries.

Drynx [FTPSH20] implements an oblivious noise selection protocol with multiple servers. One server  $S_i$  is elected as a *leader* and is in charge of quantizing a Laplace distribution and sampling a list of noise values (Fig. 45d). All servers sequentially perform a verifiable shuffle protocol on the list of noise values [Nef03]. The leader selects and adds the first value in the shuffled noise list, which is encrypted, so none of the servers know which noise value has been selected.

**Partial noise aggregation LDP.** Mugunthan et al. [MPBB19] implements a partial noise aggregation protocol to achieve LDP with **PerturbOutput**. The difference with their CDP protocol is that each party samples  $n - 1$  gamma-distributed noise values, which are masked and sent to the other  $n - 1$  parties (Fig. 45b). Then each party adds the  $n - 1$  masked gamma terms to its locally sampled noise value, achieving LDP before the local update is sent to the server. This approach achieves malicious security against  $n - 1$  colluding parties since the local updates of the honest parties satisfy LDP.

## 5.4 Analysis

---

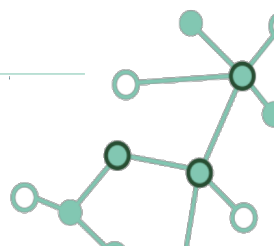
Next, we analyze key features of the surveyed works, focusing on security features and noise generation scenarios.

### 5.4.1 Security Features

---

This section analyzes malicious secure protocols and compares the cryptographic techniques implemented in the surveyed works.

**Malicious security.** Maliciously secure protocols ensure that all the parties follow the protocol correctly. The parties provide proof of the computation or input correctness. The proof does not have to reveal the secret input of the party. Most of the surveyed works leverage *zero knowledge proofs* (ZKP), which allows a party to prove the validity of a statement over a secret without disclosing the secret itself [EKR<sup>+</sup>18].





| Architecture (Fig. 44) | Paper                  | Client interactivity | # servers | ML models | Differential Privacy (Sec. 5.3.3) |                      |                       |             |                 |                       | Cryptography (Sec. 5.3.2) |                       |                              | Security (Sec. 5.4.1) |           |
|------------------------|------------------------|----------------------|-----------|-----------|-----------------------------------|----------------------|-----------------------|-------------|-----------------|-----------------------|---------------------------|-----------------------|------------------------------|-----------------------|-----------|
|                        |                        |                      |           |           | DP definition                     | Noise gen. (Fig. 45) | Perturbation (Alg. 1) | Noise mech. | Oblivious noise | Masking               | (*)HE                     | Secret sharing        | Garbled circuits             | Malicious             | Colluding |
| Single server          | [ASY <sup>+</sup> 18]  | ✓                    | 1         | NN        | $(\epsilon, \delta)$              | Part                 | Grad                  | Bin         | ×               | [BIK <sup>+</sup> 17] |                           | [BIK <sup>+</sup> 17] |                              | ×                     | ×         |
|                        | [KLS21]                | ✓                    | 1         | NN        | zCDP                              | Part                 | Grad                  | DGauss      | ×               | [BIK <sup>+</sup> 17] |                           | [BIK <sup>+</sup> 17] |                              | ×                     | ×         |
|                        | [MPBB19]               | ✓                    | 1         | LoR       | $(\epsilon, \delta)$              | Part                 | Out                   | Gamma       | ✓               | [BIK <sup>+</sup> 17] |                           | [BIK <sup>+</sup> 17] |                              | ×                     | $n - 1$   |
|                        | [BMPB22]               | ✓                    | 1         | LoR       | $(\epsilon, \delta)$              | Part                 | Out                   | Gamma       | ✓               | [BIK <sup>+</sup> 17] |                           |                       |                              | ×                     | $n - 1$   |
|                        | [SSV <sup>+</sup> 22]  | ✓                    | 1         | NN        | RDP                               | Part                 | Grad                  | DGauss      | ×               | LWE                   |                           | [Ben00]               |                              | CS                    | $n/2$     |
|                        | [BRB <sup>+</sup> 17]  |                      | 1         | —         | $(\epsilon, \delta)$              | Part/Loc             | —                     | Gamma       | ✓               |                       | (A) [DjN10]               | [Sha79]               |                              | S                     | $n - 1$   |
|                        | [TBA <sup>+</sup> 19]  |                      | 1         | NN        | $(\epsilon, \delta)$              | Part                 | Grad                  | Gamma       | ×               |                       | (A) [DJ01]                | [DJ01]                |                              | ×                     | $n - 1$   |
| Multi Hom.             | [CGBL <sup>+</sup> 17] |                      | 2         | NN        | $(\epsilon, \delta)$              | Dist                 | Grad                  | Lap/Gaus    | ✓               |                       |                           | Lin                   | [AFL <sup>+</sup> 16, Yao86] | ×                     | ×         |
|                        | [PRM <sup>+</sup> 22]  |                      | $m$       | LoR       | $(\epsilon, \delta)$              | Dist                 | Out                   | Gamma       | ✓               |                       |                           | [Kel20]               |                              | ×                     | ×         |
|                        | [JWEG18]               |                      | 2         | Lo/iR     | $(\epsilon, \delta) / zCDP$       | Dist                 | Out/Grad              | Lap/Gauss   | ✓               |                       |                           | [TJGE16]              | [TJGE16, Yao86]              | ×                     | ×         |
|                        | [RXF <sup>+</sup> 23]  | ✓                    | $n$       | NN        | $(\epsilon, \delta)$              | Part                 | Grad                  | Bin         | ×               |                       |                           | [Sha79, GMW19]        |                              | ×                     | $n/2$     |
| Het.                   | [CCDD <sup>+</sup> 21] | ✓                    | $n + 1$   | NN        | $(\epsilon, \delta)$              | Cen                  | Label                 | Lap/Gauss   | ×               |                       | (F) [BLCW19]              | Lin                   | [WMK16]                      | ×                     | ×         |
|                        | [FTPSH20]              |                      | $m$       | LoR       | $(\epsilon, \delta)$              | Dist                 | Out                   | Lap         | ✓               |                       | (A) [EIG85]               |                       |                              | CS                    | $m - 1$   |

Table 3: Key features of analyzed works with  $n$  clients and  $m$  servers where clients can assume server role. Symbols ✓ and × denote existing and missing features, respectively, and — indicates missing descriptions in [BRB<sup>+</sup>17]. Column *Architecture* uses abbreviations (e.g., het. refers to heterogeneous). *Client interactivity* means clients are active MPC parties (no outsourcing). *ML models* are neural networks (NN), logistic regression (LoR), or linear regression (LiR). *Noise generation* distinguishes central, local, partial noise aggregation and distributed noise sampling. *Perturbation* types are output, gradient, or label (i.e., collaborative labeling). *Noise mechanisms* can be Laplace, Gaussian, Binomial, Gamma and Discrete Gaussian. *Oblivious noise* means no party learns the DP noise. *Cryptography* provides references or, if unspecified in the paper, abbreviations: Linear, Additive, Fully. *Security* abbreviates Clients and Servers and reports colluding parties ( $n$  clients,  $m$  servers).

Drynx [FTPSH20] achieves malicious security against  $m - 1$  colluding computing parties (CP) and malicious clients. To ensure computation correctness and result robustness, *verification parties* (VP) verify ZKPs and store verification results in a blockchain [ZXD<sup>+</sup>18], which guarantees integrity, immutability, and audibility of the system. Clients provide range proofs [CCS08] for their input, while CPs provide signed proofs of correctness for each protocol step, e.g., during aggregation, CPs provide input and output ciphertext that constitutes a ZKP. The correctness of the oblivious noise selection protocol, based on a collective DP noise values shuffle, is ensured by verifiable shuffle [Nef03]. To improve the performance of the verification protocol, Froelicher et al. [FTPSH20] implements a probabilistic verification to detect misbehaving parties with high probability. The range proofs in Drynx ensure protection against input manipulation attacks, e.g., poisoning attacks [CJG21, SCD<sup>+</sup>21] allow a malicious party to skew the output of the analysis.

Stevens et al. [SSV<sup>+</sup>22] and Bindschaedler et al. [BRB<sup>+</sup>17] consider the *covert security* model, in which a malicious party can deviate from the protocol but is detected with a fixed probability

[EKR<sup>+</sup>18]. Stevens et al. [SSV<sup>+</sup>22] detect a misbehaving client in the key reconstruction protocol via verifiable  $(t, n)$ -SS [Ben00] assuming at least  $t/2 + 1$  honest clients. Recall, to remove the mask from the aggregated update, the server needs a secret key  $s = \sum_{i=0}^{t-1} s_i$ , where  $s_i$  is the secret key sampled by each client  $C_i$ . For the verifiable reconstruction protocol, each client secret shares  $s_i$ , via packed Shamir's secret sharing [FY92], to all the clients  $C_j$  who receives share  $s_{ij}$ . Each  $C_i$  aggregates the shares as  $sum_i = \sum_{j=0}^{k-1} s_{ij}$ , where  $k$  is the number of clients in the key reconstruction protocol, and then sends  $sum_i$  to all clients. The verifiable  $(t, n)$ -SS is run on at least  $t + 1$  shares. Each honest party aggregates two subsets of shares, containing either  $t$  or  $t + 1$  shares of the secret  $s$ , comparing the two results. If they differ, there is a malicious party in the protocol and the honest party aborts; otherwise, the client shares  $s$  with the server. The server can now unmask the aggregated gradient update. Stevens et al. [SSV<sup>+</sup>22] extend their protocol to protect against malicious server, where the clients aggregate the local updates with the same verifiable  $(t, n)$ -SS protocol described above [Ben00]. If no cheating is detected, the clients send the aggregated update to the server.

Bindschaedler et al. [BRB<sup>+</sup>17] detect a cheating server in the oblivious noise selection protocol via ZKP by Stern [Ste98]. Recall, the server generates a set of selector vectors to obviously select  $m$  noise values from the clients' sampled noises. The server provides the ZKP that each selector vector contains a 1 and that the other elements are all 0s to obtain CDP noise.

Recently, Bell et al. [BGL<sup>+</sup>22] augmented secure aggregation [BIK<sup>+</sup>17] with malicious security and check that the masked inputs of clients have a bounded norm and the protocol message satisfies some pre-defined notion of validity.

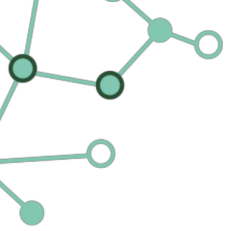
**Cryptographic techniques.** Here, we discuss cryptographic techniques from surveyed works, focusing on security guarantees, communication, and computation overhead.

HE and Masking are suitable for single-server architectures. Masking is lightweight since it involves only modular addition and subtraction. The most expensive part is the setup, i.e., each pair of clients runs a key agreement protocol to share a random seed for the mask, which increases the communication cost. Stevens et al. [SSV<sup>+</sup>22] replace the key agreement leveraging LWE, in which the clients know a public matrix  $A$ .

Homomorphic encryption has a high computational cost to execute operations over the ciphertext. It requires the transfer of large ciphertexts to be processed and decrypted. In distributed settings, the secret key has to be secret shared among the clients, who can encrypt their local updates using the public key. The clients run a key reconstruction protocol to decrypt the aggregated result after the aggregation.

Secret sharing and garble circuits suit multiple homogeneous servers architecture.  $(t, n)$ -SS base its security on non-collusion and requires each party to secret share its input with all the other parties, leading to a non-negligible communication overhead. The message size is small, but the number of messages is  $O(n^2)$  for  $n$  parties. Secret sharing is efficient for arithmetic operations.  $(t, n)$ -SS resists dropouts, requiring only a subset of  $t$  parties to reconstruct the secret.

Garbled circuits are suited for evaluating Boolean circuits, e.g., comparison. GC has a high computational cost to generate the garbled circuit. It requires a setup phase for OT protocol to exchange the encrypted labels of the garbler and a second communication round to share the garbled circuit, so there are few large messages. GC can be combined with secret sharing, as in Chase et al. [CGBL<sup>+</sup>17], where the clients can secret share their input to the servers.



Next, we analyze the noise generation phase focusing on partial noise aggregation and discrete noise mechanisms.

### 5.4.2 Noise Generation Techniques

---

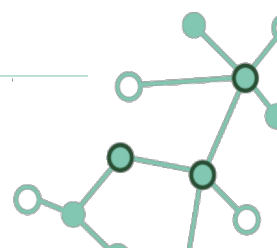
This section analyzes the noise generation phase by comparing the different approaches. We focus on partial noise aggregation, studying the security guarantees, and comparing discrete noise mechanisms.

**Secure noise generation.** In centralized noise sampling, a semi-trusted third party samples the CDP noise, and this party cannot learn the noisy output. Recall, in CAPC [CCDD<sup>+</sup>21], a semi-honest server (privacy guardian) adds noise over secret shares without learning the local updates. While distributed noise sampling removes the need for a third party, it introduces communication and computation overhead due to MPC. Between centralized noise sampling and distributed noise sampling lies partial noise aggregation, which reduces the trust assumption on the central server, which only aggregates partial noises. Partial noise aggregation can be implemented in single and multiple servers architectures achieving CDP with less computation overhead than distributed noise sampling. On the other hand, partial noise aggregation needs additional noise to protect against colluding parties. We analyze this aspect below.

The noise generation scenarios we described can be integrated with existing collaborative learning approaches which do not provide DP (e.g., works surveyed by [CHP21]). Secure aggregation protocols can leverage partial noise aggregation mechanisms, whereas MPC protocols based on SGD can be extended with DP-SGD applying either partial noise aggregation or distributed noise sampling. Note that noise generation is data-independent and can be executed offline.

**Partial noise aggregation: security.** We analyze the security guarantees of partial noise aggregation in single-server architecture (Fig. 44a) for the semi-honest model both with colluding and non-colluding adversaries. If the input parties do not collude, masking with any threshold  $t > 1$  guarantees the privacy of each input party's local update. Recall the discussion on Equation (5.3), since the clients have access to the aggregated update, the noise added by each party has to guarantee that if one party removes its noise, the aggregation is still differentially private. Oblivious noise selection [BRB<sup>+</sup>17, BMPB22] reduces the noise since each party does not know the noise she added. In the case of  $m$  colluding clients, the threat model has to assume that the threshold  $t$  for decryption is greater than the number of colluding clients  $t > m$ . Otherwise,  $m > t$  clients can reduce the noise in the global update, i.e., they remove their partial noises, recovering the private data of honest parties. Oblivious noise selection [BRB<sup>+</sup>17, BMPB22, MPBB19], solves this problem, even if  $m > t$  with  $m = n - 1$ . With oblivious noise selection, the colluding clients can not remove their partial noises, and they can only learn honest parties update with CDP noise.

**Partial noise aggregation: why discrete noise.** Note that sampling from continuous distributions could be vulnerable to attacks on floating-point representation. Mironov [Mir12] first proposed an attack exploiting the finite precision of computers. Mironov [Mir12] attacked the Laplace mecha-





nism to remove the noise and reveal private values. Jin et al. [JMRO22] showed that a similar attack is successful against the Gaussian mechanism.

All the works sampling from continuous distributions probably suffer from the aforementioned attacks. In collaborative learning, each party has access to the aggregated update and could run floating point attacks to recover local updates of honest parties, e.g., performing membership inference on the de-noised update. Ruan et al. [RXF<sup>+</sup>23] highlight that in DP-SGD, an external attacker without the knowledge of the original model and any information on the intermediate gradients may not be able to exploit floating point attacks since noises are added iteratively, and only the trained model is released.

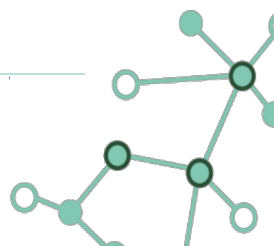
**Partial noise aggregation: discrete noise mechanisms.** Here, we focus on discrete mechanisms, more precisely on binomial and discrete Gaussian. Agarwal et al. [ASY<sup>+</sup>18] implement the binomial mechanism which approximates Gaussian noise, achieving  $(\epsilon, \delta)$ -DP, if the clients hold enough shares of unbiased coins [DKM<sup>+</sup>06] (Sec. 5.3.3). The binomial mechanism achieves neither concentrated DP [BS16] nor Rényi DP [Mir17], so it is not applicable for tighter accounting methods, e.g., sub-sampling and moments accountant, leading to a loose noise bound.

Kairouz et al. [KLS21] implement the discrete Gaussian mechanism [CKS20] whose composition bound can be analyzed using moments accountant and sub-sampling leading to concentrated DP or Rényi DP guarantee. The bound achieved on the sum of discrete Gaussian leads to privacy guarantees extremely close to continuous Gaussian, e.g., the sum of two discrete Gaussian with variance  $\sigma^2 = 3$  leads to an error bound of at most  $10^{-12}$  on the max divergence with the sum of two continuous Gaussian [KLS21, Theorem 9]. Recently, Agarwal et al. [AKL21] introduced the *Skellam* mechanism: a discrete mechanism matching the RDP bound of the discrete Gaussian mechanism [AKL21, Figure 3]. The main advantages of the Skellam mechanism over the discrete Gaussian are that a Skellam random variable can be efficiently sampled as the difference of two independent Poisson random variables, and it is closed under summation, i.e., the sum of two Skellam-distributed noises is Skellam-distributed.

## 5.5 Related Works

---

To the best of our knowledge, we are the first to systematize all works combining cryptography and differential privacy for collaborative learning. Wagh et al. [WHMM21] survey approaches combining DP and cryptography for distributed analytics, highlighting the need for DP to be combined with cryptography to reduce the amount of noise injected and improve accuracy. Mansouri et al. [MÖJC23] systematize secure aggregation techniques, formally defining the problem. They do not consider DP, but [MÖJC23, Observation 4] highlights the need for differential privacy in secure aggregation. Cabrero-Holgueras and Pastrana [CHP21] focus on deep learning mainly on inference, analyzing the difference between the cryptographic techniques in multiple settings, but do not provide an analysis on the combination of DP and cryptography, excluding DP from their list of further analyzed protection techniques.





## 5.6 Observation & Future research directions

---

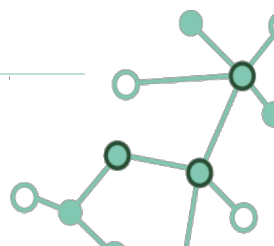
This section summarizes our main observations and lessons learned from the analysis of the surveyed works (overviewed in Table 3). We identify gaps in the research contributions of SPCL and propose future research directions.

**1) DP needs cryptography.** The combination of DP and cryptography keeps the local update private while reducing the noise amount compared to LDP. Compared to GDP, distributed noise sampling generation reduces the information the aggregator can learn for single-server architectures (i.e., only DP values) and replaces the trusted party in multiple servers architectures. Furthermore, cryptography can protect against malicious parties, e.g., preventing data poisoning attacks. Various protocols [FTPSH20, BMPB22, BRB<sup>+</sup>17] use cryptography for oblivious noise selection, reducing the injected noise, since each party is oblivious to its noise contribution in the aggregated output. On the other hand, cryptography introduces communication and computation overhead for distributed noise sampling generation. A potential research direction is the optimization of secure noise generation protocols. For example, by using DP-suitable noise inherent to cryptographic techniques, e.g., Stevens et al. [SSV<sup>+</sup>22] use LWE noise to achieve RDP.

**2) Quantization for privacy and efficiency.** Quantization allows for trading communication for accuracy while making the model less prone to floating-point attacks [AKL21]. Most of the surveyed works do not consider quantization and discrete noise sampling. This observation aligns with Kairouz et al. [KLS21], who note that most of the approaches combining DP with secure aggregation ignore the finite precision and the modular summation of secure aggregation. On the resistance to floating point attacks, Ruan et al. [RXF<sup>+</sup>23] point out that DP-SGD adds the noise iteratively. An attacker without the knowledge of intermediate gradient steps and the initial model could not exploit floating point attacks. The literature lacks implementations of floating point attacks to distributed noise sampling generation and studies on the quantization error as DP noise.

**3) Moments accountant reduces privacy loss.** Privacy analyses based on relaxed definitions, e.g., RDP or zCDP, reduce the noise in the training process, making the model more accurate. The accounting of privacy loss through moments accountant leads to a tighter analysis, i.e., in  $T$  iterations of DP-SGD, the bound on  $\epsilon$  decreases by more than  $\sqrt{T}$ , whereas the bound on  $\delta$  reduces of  $qT$ , where  $q$  is the sampling parameter [PHK<sup>+</sup>23, Table 2]. Most of the surveyed works do not consider relaxed DP definitions, as we show Table 3, which prevents them from applying moments accountant.

**4) Optimization for noise reduction.** Faster convergence and simpler models help to reduce the noise amount. Ruan et al. [RXF<sup>+</sup>23] manage to achieve both objectives moving complexity to local computations. The parties reduce the data dimensionality by feature extraction and initialize the global model with the most accurate among pre-trained local ones to ensure a faster convergence. Tramer and Boneh [TB20], in a centralized learning setting, show that features learned from public data via *transfer learning* improve feature extraction accuracy yet provide strong privacy guarantees. A similar approach is fine-tuning large public pre-trained models on private data. Another way to reduce the noise is to perform many local iterations, as in DP-FedAvg [MRTZ17]. Recently, Wang et al. [WDJ<sup>+</sup>22] showed that in many practical applications, DP-FedAvg achieves better utility than distributed DP-SGD thanks to better convergence properties.





**5) Guarantee malicious security.** Most protocols assume semi-honest parties since malicious security requires additional computational overhead [PRM<sup>+</sup>22, BRB<sup>+</sup>17]. Only Drynx [FTPSH20] provides malicious security for servers and clients by a mechanism to verify input range, avoiding poisoning attacks [CJG21, SCD<sup>+</sup>21], and a verifiable noise selection protocol for DP. None of the works leveraging partial noise aggregation ensure correct sampling of DP noise. Recent works [KCY21, MGSB22] implement verifiable noise sampling, which, combined with oblivious noise selection, reduces the magnitude of the partial noise aggregation to the optimal amount. Recall partial noise aggregation should consider  $t$  misbehaving client, but with verifiable noise sampling  $t = 0$ . Furthermore, ensuring protection against malicious servers is crucial since the server can, e.g., create fake clients injecting fake updates to retrieve honest clients' private data [BDS<sup>+</sup>21].

Like Cabrero-Holgueras and Pastrana [CHP21], we note that the analyzed works lack standardized models and benchmarks for a fair comparison. Furthermore, only a few works provide open-source code [KLS21]<sup>4</sup> [JWEG18]<sup>5</sup> [CCDD<sup>+</sup>21]<sup>6</sup> [FTPSH20]<sup>7</sup>, making it difficult to replicate the experiments.

Finally, Chapter 5 studied the landscape of secure and private collaborative learning, i.e., works combining DP and cryptography for encrypted processing of inputs (input secrecy) with privacy guarantees for the outputs (output privacy). Three types of architectures and four different distributed noise sampling generation techniques have been identified. The chapter described different approaches for each architecture based on communication patterns and the cryptographic protocols, highlighting the main features. For each noise generation scenario, the chapter explained for which architecture they are suitable, detailing how the noise is sampled by each party, depicting different noise mechanisms. Then, maliciously secure protocols have been analyzed, with a comparison of the cryptographic techniques. Finally, a comparison of the different noise sampling techniques has been presented, focusing on partial noise aggregation and the impact of the discrete noise distribution. This analysis permitted to report the key observations that will offer a starting point for future research in GLACIATION.

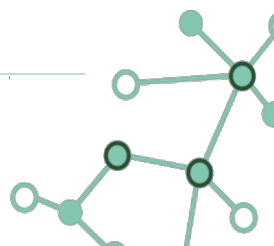
---

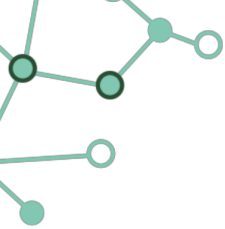
<sup>4</sup>[https://github.com/google-research/federated/tree/master/distributed\\_dp](https://github.com/google-research/federated/tree/master/distributed_dp)

<sup>5</sup><https://github.com/bargavj/distributedMachineLearning>

<sup>6</sup><https://github.com/cleverhans-lab/capc-iclr>

<sup>7</sup><https://github.com/ldsec/drynx>





## 6 Conclusions

---

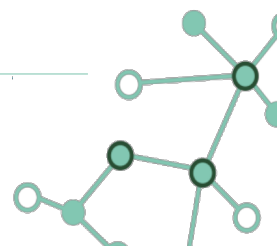
The goal of this deliverable is to present some of the most significant research results produced by the tasks in Work Package 4 within the first year of the GLACIATION project. Several lines of work have been investigated, producing contributions that will be enhanced with additional results by the end of the project.

Chapter 2 described the results of the investigation on current authorization frameworks, with the goal of identifying the open-source platform to use to support the realization of the GLACIATION policy model. Several options have been analyzed and the best candidate has been identified in the Open Policy Agent framework with the language Rego used in this framework for the representation and evaluation of policies.

Chapter 3 first has a brief description of a probabilistic technique for integrity verification of large scale computations, presenting our approach for tuning the control information added to the workload. The chapter then focuses on presenting our approach for efficiently enforcing access revocation on encrypted resources stored at external providers. The solution enables data owners to effectively revoke access by simply overwriting a small portion of the (potentially large) resource and is resilient against attacks by users locally maintaining copies of previously-used keys. The experimental evaluation confirms the efficiency and effectiveness of the proposal, which enjoys orders of magnitude of improvement in throughput with respect to resource re-writing, and confirms its compatibility with current cloud storage solutions, making it also immediately applicable to the GLACIATION scenario and many other application domains.

Chapter 4 presented our scalable approach for anonymizing very large datasets in a distributed scenario. The approach divides the dataset to be anonymized into fragments, which are then distributed to multiple workers that operate in parallel and independently. Several distinct partitioning strategies that operate on a sample of the dataset have been proposed, and the chapter presented the design of a distributed version of the Mondrian anonymization algorithm that minimizes the amount of information that needs to be exchanged between workers. The experimental results confirm that the approach is scalable and does not significantly impact the quality of the anonymized data.

Finally, Chapter 5 studied the landscape of secure and private collaborative learning, i.e., works combining DP and cryptography for encrypted processing of inputs (input secrecy) with privacy guarantees for the outputs (output privacy). Three types of architectures and four different distributed noise sampling generation techniques have been identified. The chapter described different approaches for each architecture based on communication patterns and the cryptographic protocols, highlighting the main features. For each noise generation scenario, the chapter explained for which architecture they are suitable, detailing how the noise is sampled by each party, depicting different noise mechanisms. Then, maliciously secure protocols have been analyzed, with a comparison of the cryptographic techniques. Finally, a comparison of the different noise sampling techniques has been presented, focusing on partial noise aggregation and the impact of the discrete noise distribution. This analysis permitted to report the key observations that will offer a starting point for future research in GLACIATION.

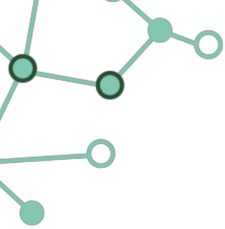


## References

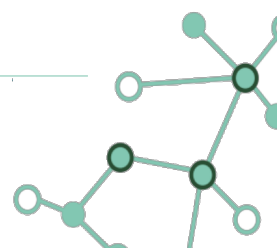
---

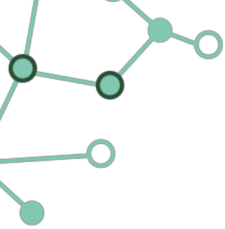
- [ABD19] E. Arfelt, D. Basin, and S. Debois. Monitoring the GDPR. In *Proc. of the 24th European Symposium on Research in Computer Security (ESORICS)*, Luxembourg, September 2019.
- [ABF<sup>+</sup>23] M. Abbadini, M. Beretta, D. Facchinetti, G. Oldani, M. Rossi, and S. Paraboschi. Poster: Leveraging ebpf to enhance sandboxing of webassembly runtimes. In *Proceeding of the 18th ACM ASIA Conference on Computer and Communications Security (ACM ASIACCS 2023)*, 2023.
- [ABH<sup>+</sup>18] S. Antonatos, S. Braghin, N. Holohan, Y. Gkoufas, and P. Mac Aonghusa. Prima: an end-to-end framework for privacy at scale. In *Proc. of ICDE 2018*, Paris, France, April 2018.
- [ABM14] E. Andreeva, A. Bogdanov, and B. Mennink. Towards understanding the known-key security of block ciphers. In *Proc. of FSE*, Hong Kong, November 2014.
- [AC12] G. Acs and C. Castelluccia. Dream: Differentially private smart metering. *arXiv preprint arXiv:1201.2531*, 2012.
- [ACG<sup>+</sup>16] M. Abadi, A. Chu, I. Goodfellow, H. B. McMahan, I. Mironov, K. Talwar, and L. Zhang. Deep learning with differential privacy. In *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security, CCS*, pages 308–318, 2016.
- [AFL<sup>+</sup>16] T. Araki, J. Furukawa, Y. Lindell, A. Nof, and K. Ohara. High-throughput semi-honest secure three-party computation with an honest majority. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 805–817, 2016.
- [AFO<sup>+</sup>23a] M. Abbadini, D. Facchinetti, G. Oldani, M. Rossi, and S. Paraboschi. Cage4deno: A fine-grained sandbox for deno subprocesses. In *Proceeding of the 18th ACM ASIA Conference on Computer and Communications Security (ACM ASIACCS 2023)*, 2023.
- [AFO<sup>+</sup>23b] M. Abbadini, D. Facchinetti, G. Oldani, M. Rossi, and S. Paraboschi. Natisand: Native code sandboxing for javascript runtimes. In *Proceedings of the 26th International Symposium on Research in Attacks, Intrusions and Defenses*, 2023.
- [AKL21] N. Agarwal, P. Kairouz, and Z. Liu. The skellam mechanism for differentially private federated learning. *Advances in Neural Information Processing Systems*, 34:5052–5064, 2021.
- [AKS21] F. Ashkouti, K. Khamforoosh, and A. Sheikahmadi. DI-Mondrian: Distributed improved Mondrian for satisfaction of the  $\ell$ -diversity privacy model using Apache Spark. *Information Sciences*, 546:1–24, 2021.
- [AKSK21] F. Ashkouti, K. Khamforoosh, A. Sheikahmadi, and H. Khamfroush. DHkmeans- $\ell$ -diversity: distributed hierarchical  $k$ -means for satisfaction of the  $\ell$ -diversity privacy model using Apache Spark. *The Journal of Supercomputing*, 78(6):2616–2650, 2021.



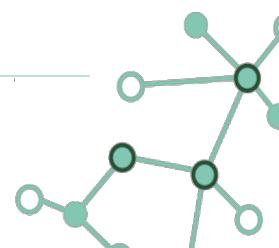


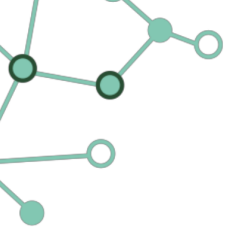
- [ASY<sup>+</sup>18] N. Agarwal, A. T. Suresh, F. X. X. Yu, S. Kumar, and B. McMahan. cpsgd: Communication-efficient and differentially-private distributed sgd. *Advances in Neural Information Processing Systems*, 31, 2018.
- [Aut23] T. K. Authors. Admission Controllers Reference. <https://kubernetes.io/docs/reference/access-authn-authz/admission-controllers/>, 2023.
- [BA05] R. J. Bayardo and R. Agrawal. Data privacy through optimal  $k$ -anonymization. In *Proc. of ICDE 2005*, Tokoyo, Japan, April 2005.
- [BDF<sup>+</sup>20] E. Bacis, S. De Capitani di Vimercati, S. Foresti, S. Paraboschi, M. Rosa, and P. Samarati. Securing resources in decentralized cloud storage. *IEEE TIFS*, 15(1):286–298, December 2020.
- [BDF<sup>+</sup>23] E. Bacis, S. De Capitani di Vimercati, S. Foresti, S. Paraboschi, M. Rosa, and P. Samarati. Mix&slice for efficient access revocation on outsourced data. *IEEE Transactions on Dependable and Secure Computing (TDSC)*, 2023. to appear.
- [BDS<sup>+</sup>21] F. Boenisch, A. Dziedzic, R. Schuster, A. S. Shamsabadi, I. Shumailov, and N. Papernot. When the curious abandon honesty: Federated learning is not private. *arXiv preprint arXiv:2112.02918*, 2021.
- [BEKS20] D. Boneh, S. Eskandarian, S. Kim, and M. Shih. Improving speed and security in updatable encryption schemes. In *Proc. of ASIACRYPT*, December 2020. (virtual).
- [BEM<sup>+</sup>17] A. Bittau, Ú. Erlingsson, P. Maniatis, I. Mironov, A. Raghunathan, D. Lie, M. Rudominer, U. Kode, J. Tinnes, and B. Seefeld. Prochlo: Strong privacy for analytics in the crowd. In *Proceedings of the 26th symposium on operating systems principles*, pages 441–459, 2017.
- [Ben00] J. C. Benaloh. Secret sharing homomorphisms: Keeping shares of a secret secret. In *Advances in Cryptology—CRYPTO’86: Proceedings*, pages 251–260. Springer, 2000.
- [BFG<sup>+</sup>21] E. Bacis, D. Facchinetti, M. Guarnieri, M. Rosa, M. Rossi, and S. Paraboschi. I told you tomorrow: Practical time-locked secrets using smart contracts. In *Proc. of the 16th International Conference on Availability, Reliability and Security (ARES 2021)*, Aug. 2021.
- [BGL<sup>+</sup>22] J. Bell, A. Gascón, T. Lepoint, B. Li, S. Meiklejohn, M. Raykova, and C. Yun. Acorn: Input validation for secure aggregation. *Cryptology ePrint Archive*, 2022.
- [BIK<sup>+</sup>17] K. Bonawitz, V. Ivanov, B. Kreuter, A. Marcedone, H. B. McMahan, S. Patel, D. Ramage, A. Segal, and K. Seth. Practical secure aggregation for privacy-preserving machine learning. In *proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 1175–1191, 2017.
- [BJJA21] S. U. Bazai, J. Jang-Jaccard, and H. Alavizadeh. A novel hybrid approach for multi-dimensional data anonymization for Apache Spark. *ACM TOPS*, 25(1):5:1–5:25, 2021.



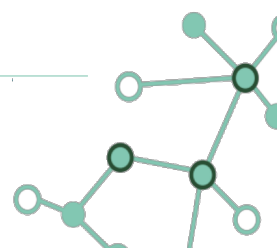


- [BK19] P. Bonatti and S. Kirrane. Big data and analytics in the age of the GDPR. In *Proc. of the 2019 IEEE International Congress on Big Data*, Los Angeles, CA, USA, December 2019.
- [BLCW19] F. Boemer, Y. Lao, R. Cammarota, and C. Wierzynski. Ngraph-he: A graph compiler for deep learning on homomorphically encrypted data. In *Proceedings of the 16th ACM International Conference on Computing Frontiers*, CF '19. Association for Computing Machinery, 2019.
- [BM58] G. E. Box and M. E. Muller. A note on the generation of random normal deviates. *The annals of mathematical statistics*, 29(2):610–611, 1958.
- [BMPB22] D. Byrd, V. Mugunthan, A. Polychroniadou, and T. Balch. Collusion resistant federated learning with oblivious distributed differential privacy. In *Proceedings of the Third ACM International Conference on AI in Finance*, pages 114–122, 2022.
- [Boy99] V. Boyko. On the security properties of OAEP as an all-or-nothing transform. In *Proc. of CRYPTO*, Santa Barbara, CA, USA, August 1999.
- [BR94] M. Bellare and P. Rogaway. Optimal asymmetric encryption. In *Proc of EUROCRYPT*, Perugia, Italy, May 1994.
- [BRB<sup>+</sup>17] V. Bindschaedler, S. Rane, A. E. Brito, V. Rao, and E. Uzun. Achieving differential privacy in secure multiparty data aggregation protocols on star networks. In *Proceedings of the Seventh ACM on Conference on Data and Application Security and Privacy*, CODASPY '17. Association for Computing Machinery, 2017.
- [BS16] M. Bun and T. Steinke. Concentrated differential privacy: Simplifications, extensions, and lower bounds. In *Theory of Cryptography: 14th International Conference, TCC 2016-B, Beijing, China, October 31-November 3, 2016, Proceedings, Part I*, pages 635–658. Springer, 2016.
- [CCDD<sup>+</sup>21] C. A. Choquette-Choo, N. Dullerud, A. Dziedzic, Y. Zhang, S. Jha, N. Papernot, and X. Wang. Capc learning: Confidential and private collaborative learning. *arXiv preprint arXiv:2102.05188*, 2021.
- [CCS08] J. Camenisch, R. Chaabouni, and A. Shelat. Efficient protocols for set membership and range proofs. In *Proceedings of the 14th International Conference on the Theory and Application of Cryptology and Information Security*, pages 234–252. Springer, 2008.
- [Cer23a] Cerbos.dev. Cerbos. <https://cerbos.dev/>, 2023.
- [Cer23b] Cerbos.dev. Cerbos github page. <https://github.com/cerbos/cerbos>, 2023.
- [Cer23c] Cerbos.dev. Painless access control for your software. <https://docs.cerbos.dev/cerbos/latest/index.html>, 2023.
- [CGBL<sup>+</sup>17] M. Chase, R. Gilad-Bachrach, K. Laine, K. Lauter, and P. Rindal. Private collaborative neural network learning. *Cryptology ePrint Archive*, Paper 2017/762, 2017. <https://eprint.iacr.org/2017/762>.



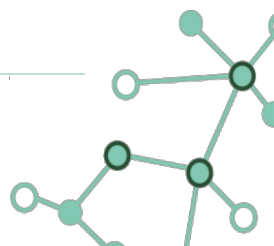


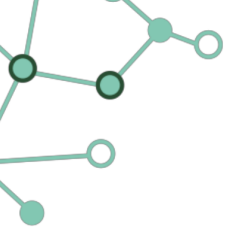
- [CHHS13] C. Cachin, K. Haralambiev, H. Hsiao, and A. Sorniotti. Policy-based secure deletion. In *Proc. of CCS*, Berlin, Germany, November 2013.
- [CHP21] J. Cabrero-Holgueras and S. Pastrana. Sok: Privacy-preserving computation techniques for deep learning. *Proceedings on Privacy Enhancing Technologies*, 2021(4):139–162, 2021.
- [CJG21] X. Cao, J. Jia, and N. Z. Gong. Data poisoning attacks to local differential privacy protocols. In *30th {USENIX} Security Symposium ({USENIX} Security 21)*, 2021.
- [CKKS17] J. H. Cheon, A. Kim, M. Kim, and Y. Song. Homomorphic encryption for arithmetic of approximate numbers. In *International conference on the theory and application of cryptology and information security*, pages 409–437. Springer, 2017.
- [CKS20] C. L. Canonne, G. Kamath, and T. Steinke. The discrete gaussian for differential privacy. *Advances in Neural Information Processing Systems*, 33:15676–15688, 2020.
- [CM08] K. Chaudhuri and C. Monteleoni. Privacy-preserving logistic regression. *Advances in neural information processing systems*, 21, 2008.
- [CO07] R. Catral and F. Oppacher. Poker Hand Data Set, UCI machine learning repository, 2007. <https://archive.ics.uci.edu/ml/datasets/Poker+Hand>.
- [Com23] A. R. Community. Apache ranger. <https://ranger.apache.org/>, 2023.
- [CRR20] S. Contiu, L. Réveillère, and E. Rivière. Practical active revocation. In *Proc. of Middleware*, Delft, Netherlands, 2020.
- [CSU<sup>+</sup>19] A. Cheu, A. Smith, J. Ullman, D. Zeber, and M. Zhilyaev. Distributed differential privacy via shuffling. In *Advances in Cryptology–EUROCRYPT 2019*, pages 375–403. Springer, 2019.
- [CSU21] A. Cheu, A. Smith, and J. Ullman. Manipulation attacks in local differential privacy. In *2021 IEEE Symposium on Security and Privacy (SP)*, 2021.
- [CWR14] A. Chakravorty, T. W. Wlodarczyk, and C. Rong. A scalable  $k$ -anonymization solution for preserving privacy in an aging-in-place welfare intercloud. In *Proc. of IC2E 2014*, Boston, MA, USA, March 2014.
- [Des22] D. Desfontaines. Averaging risk: Rényi dp & zero-concentrated dp, 2022. Ted is writing things (personal blog).
- [DFF<sup>+</sup>21a] S. De Capitani di Vimercati, D. Facchinetti, S. Foresti, G. Oldani, S. Paraboschi, M. Rossi, and P. Samarati. Artifact: Scalable distributed data anonymization. In *Proc. of IEEE PerCom 2021*, Kassel, Germany, March 2021.
- [DFF<sup>+</sup>21b] S. De Capitani di Vimercati, D. Facchinetti, S. Foresti, G. Oldani, S. Paraboschi, M. Rossi, and P. Samarati. Multi-dimensional indexes for point and range queries on out-sourced encrypted data. In *Proc. of the 2021 IEEE Global Communications Conference (GLOBECOM 2021)*, Madrid, Spain, December 2021.



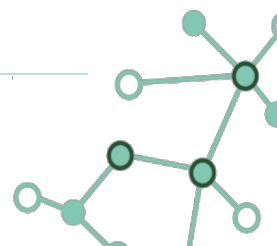


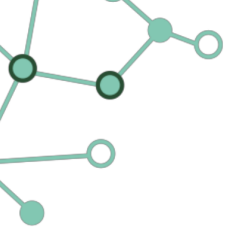
- [DFF<sup>+</sup>21c] S. De Capitani di Vimercati, D. Facchinetti, S. Foresti, G. Oldani, S. Paraboschi, M. Rossi, and P. Samarati. Scalable distributed data anonymization. In *Proc. of IEEE PerCom 2021*, Kassel, Germany, March 2021.
- [DFF<sup>+</sup>23] S. De Capitani di Vimercati, D. Facchinetti, S. Foresti, G. Livraga, G. Oldani, S. Paraboschi, M. Rossi, and P. Samarati. Scalable distributed data anonymization for large datasets. *IEEE Transactions on Big Data (TBD)*, 9(3):818–831, May/June 2023.
- [DFJ<sup>+</sup>07] S. De Capitani di Vimercati, S. Foresti, S. Jajodia, S. Paraboschi, and P. Samarati. Over-encryption: Management of access control evolution on outsourced data. In *Proc. of VLDB*, Vienna, Austria, September 2007.
- [DFJ<sup>+</sup>10a] S. De Capitani di Vimercati, S. Foresti, S. Jajodia, S. Paraboschi, and P. Samarati. Encryption policies for regulating access to outsourced data. *ACM TODS*, 35(2):12:1–12:46, April 2010.
- [DFJ<sup>+</sup>10b] S. De Capitani di Vimercati, S. Foresti, S. Jajodia, S. Paraboschi, and P. Samarati. Fragments and loose associations: Respecting privacy in data publishing. *PVLDB*, 3(1):1370–1381, September 2010.
- [DFJ<sup>+</sup>15] S. De Capitani di Vimercati, S. Foresti, S. Jajodia, G. Livraga, S. Paraboschi, and P. Samarati. Loose associations to increase utility in data publishing. *JCS*, 23(1):59–88, 2015.
- [DFJ<sup>+</sup>23] S. De Capitani di Vimercati, S. Foresti, S. Jajodia, S. Paraboschi, R. Sassi, and P. Samarati. Sentinels and twins: Effective integrity assessment for distributed computation. *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, 34(1):108–122, January 2023.
- [DFK<sup>+</sup>06] I. Damgård, M. Fitzi, E. Kiltz, J. B. Nielsen, and T. Toft. Unconditionally secure constant-rounds multi-party computation for equality, comparison, bits and exponentiation. In *Theory of Cryptography: Third Theory of Cryptography Conference, TCC 2006, New York, NY, USA, March 4-7, 2006. Proceedings 3*, pages 285–304. Springer, 2006.
- [DFLS22] S. De Capitani di Vimercati, S. Foresti, G. Livraga, and P. Samarati. Towards owner-controlled data sharing. In P. Nicosopolitidis, S. Misra, and L. Yang, editors, *Advances in Computing, Informatics, Networking and Cybersecurity*. Springer, 2022.
- [DFT05] J. Domingo-Ferrer and V. Torra. Ordinal, continuous and heterogeneous  $k$ -anonymity through microaggregation. *Data Mining and Knowledge Discovery*, 11(2):195–212, 2005.
- [DFT08] J. Domingo-Ferrer and V. Torra. A critique of  $k$ -anonymity and some of its enhancements. In *Proc. of ARES 2008*, Barcelona, Spain, March 2008.
- [DG08] J. Dean and S. Ghemawat. MapReduce: Simplified data processing on large clusters. *CACM*, 51(1):107–113, 2008.
- [DH22] W. Diffie and M. E. Hellman. New directions in cryptography. In *Democratizing Cryptography: The Work of Whitfield Diffie and Martin Hellman*, pages 365–390. 2022.



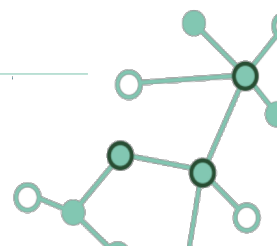


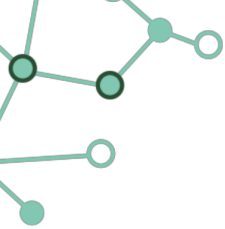
- [DJ01] I. Damgård and M. Jurik. A generalisation, a simplification and some applications of paillier's probabilistic public-key system. In *Public Key Cryptography: 4th International Workshop on Practice and Theory in Public Key Cryptosystems, PKC 2001 Cheju Island, Korea, February 13–15, 2001 Proceedings 4*, pages 119–136. Springer, 2001.
- [DJN10] I. Damgård, M. Jurik, and J. B. Nielsen. A generalization of paillier's public-key system with applications to electronic voting. *International Journal of Information Security*, 9:371–385, 2010.
- [DJW13] J. C. Duchi, M. I. Jordan, and M. J. Wainwright. Local privacy and statistical minimax rates. In *2013 IEEE 54th Annual Symposium on Foundations of Computer Science*, pages 429–438. IEEE, 2013.
- [DKM<sup>+</sup>06] C. Dwork, K. Kenthapadi, F. McSherry, I. Mironov, and M. Naor. Our data, ourselves: Privacy via distributed noise generation. In *Advances in Cryptology-EUROCRYPT 2006: 24th Annual International Conference on the Theory and Applications of Cryptographic Techniques, St. Petersburg, Russia, May 28-June 1, 2006. Proceedings 25*, EUROCRYPT, pages 486–503. Springer, 2006.
- [DR14] C. Dwork and A. Roth. The algorithmic foundations of differential privacy. *Foundations and Trends in Theoretical Computer Science*, 9(3-4):211–407, 2014.
- [DW10] S. Diesburg and A. Wang. A survey of confidential data storage and deletion methods. *ACM Computer Surveys*, 43(1), December 2010.
- [Dwo01] M. Dworkin. Recommendation for block cipher modes of operation, methods and techniques. Technical Report NIST Special Publication 800-38A, National Institute of Standards and Technology, 2001.
- [Dwo04] M. J. Dworkin. Sp 800-38c. Recommendation for block cipher modes of operation: The CCM mode for authentication and confidentiality. Technical report, National Institute of Standards & Technology, Gaithersburg, MD, United States, 2004.
- [Dwo06a] C. Dwork. Differential privacy. In *Proc. of ICALP 2006*, Venice, Italy, July 2006.
- [Dwo06b] C. Dwork. Differential privacy. In *Automata, Languages and Programming: 33rd International Colloquium, ICALP 2006, Venice, Italy, July 10-14, 2006, Proceedings, Part II 33*, ICALP, pages 1–12. Springer, 2006.
- [DYL<sup>+</sup>13] X. Ding, Q. Yu, J. Li, J. Liu, and H. Jin. Distributed anonymization for multiple data providers in a cloud system. In *Proc. of DASFAA 2013*, Wu Han, China, April 2013.
- [EKR<sup>+</sup>18] D. Evans, V. Kolesnikov, M. Rosulek, et al. A pragmatic introduction to secure multi-party computation. *Foundations and Trends® in Privacy and Security*, 2(2-3):70–246, 2018.
- [EIG85] T. ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE transactions on information theory*, 1985.



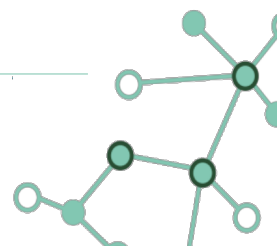


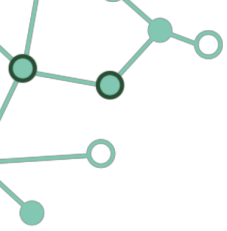
- [EPK14] U. Erlingsson, V. Pihur, and A. Korolova. Rappor: Randomized aggregatable privacy-preserving ordinal response. In *Proceedings of the 21st ACM Conference on Computer and Communications Security*, Scottsdale, Arizona, 2014.
- [EU18] EU. General data protection regulation, 2018.
- [FJR15] M. Fredrikson, S. Jha, and T. Ristenpart. Model inversion attacks that exploit confidence information and basic countermeasures. In *Proceedings of the 22nd ACM SIGSAC conference on computer and communications security*, pages 1322–1333, 2015.
- [FKK06] K. Fu, S. Kamara, and Y. Kohno. Key regression: Enabling efficient key distribution for secure distributed storage. In *Proc. of NDSS*, San Diego, CA, USA, February 2006.
- [FTPSH20] 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*, 15:3035–3050, 2020.
- [FY92] M. Franklin and M. Yung. Communication complexity of secure computation. In *Proceedings of the twenty-fourth annual ACM symposium on Theory of computing*, pages 699–710, 1992.
- [FYDX21] S. Fei, Z. Yan, W. Ding, and H. Xie. Security vulnerabilities of sgx and countermeasures: A survey. *ACM Computing Surveys (CSUR)*, 54(6):1–36, 2021.
- [Gen09] C. Gentry. *A fully homomorphic encryption scheme*. Stanford university, 2009.
- [GESM17] J. Götzfried, M. Eckert, S. Schinzel, and T. Müller. Cache attacks on intel sgx. In *Proceedings of the 10th European Workshop on Systems Security*, pages 1–6, 2017.
- [GMW19] O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game, or a completeness theorem for protocols with honest majority. In *Providing Sound Foundations for Cryptography: On the Work of Shafi Goldwasser and Silvio Micali*, pages 307–328. 2019.
- [GPSW06] V. Goyal, O. Pandey, A. Sahai, and B. Waters. Attribute-based encryption for fine-grained access control of encrypted data. In *Proc. of CCS*, Alexandria, VA, USA, October–November 2006.
- [GPW<sup>+</sup>20] S. Ghayyur, P. Pappachan, G. Wang, S. Mehrotra, and N. Venkatasubramanian. Designing privacy preserving data sharing middleware for internet of things. In *Proc. of the 3rd International SenSys+BuildSys Workshop on Data:Acquisition to Analysis (DATA 2020)*, November 2020.
- [GSB<sup>+</sup>22] C. Ge, W. Susilo, J. Baek, Z. Liu, J. Xia, and L. Fang. Revocable attribute-based encryption with data integrity in clouds. *IEEE TDSC*, 19(5):2864–2872, 2022.
- [GX15] S. Goryczka and L. Xiong. A comprehensive comparison of multiparty secure additions with differential privacy. *IEEE transactions on dependable and secure computing*, 14(5):463–477, 2015.



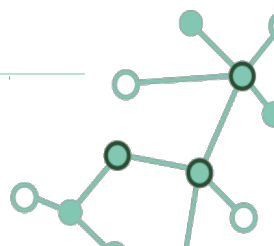


- [Has23a] Hasura. Authentication and authorization. <https://hasura.io/docs/latest/auth/how-it-works/>, 2023.
- [Has23b] Hasura. Hasura graphql engine documentation. <https://hasura.io/docs/latest/index/>, 2023.
- [Has23c] Hasura. Hasura graphql engine github page. <https://github.com/hasura/graphql-engine>, 2023.
- [HJM07] B. Hore, R. C. Jammalamadaka, and S. Mehrotra. Flexible anonymization for privacy preserving data publishing: A systematic search based approach. In *Proc. of SIAM SDM 2007*, Minneapolis, MN, USA, April 2007.
- [HK23] F. Hartmann and P. Kairouz. Distributed differential privacy for federated learning, 2023.
- [HKD15] I. Hang, F. Kerschbaum, and E. Damiani. ENKI: Access control for encrypted query processing. In *Proc. of SIGMOD*, Melbourne, Australia, May 2015.
- [HLK<sup>+</sup>17] M. Heikkilä, E. Lagerspetz, S. Kaski, K. Shimizu, S. Tarkoma, and A. Honkela. Differentially private bayesian learning on distributed data. *Advances in neural information processing systems*, 30, 2017.
- [HN11] J. Hur and D. Noh. Attribute-based access control with efficient revocation in data outsourcing systems. *IEEE TPDS*, 22(7):1214–1221, July 2011.
- [HSX<sup>+</sup>12] C. Huang, H. Simitci, Y. Xu, A. Ogus, B. Calder, P. Gopalan, J. Li, and S. Yekhanin. Erasure coding in windows azure storage. In *Proc. of USENIX ATC*, Boston, MA, USA, June 2012.
- [JAC<sup>+</sup>17] D. Julkowska, C. P. Austin, C. M. Cutillo, D. Gancberg, C. Hager, J. Halftermeyer, A. H. Jonker, L. Lau, I. Norstedt, A. Rath, et al. The importance of international collaboration for rare diseases research: a european perspective. *Gene therapy*, 24(9):562–571, 2017.
- [JE19] B. Jayaraman and D. Evans. Evaluating differentially private machine learning in practice. In *USENIX Security Symposium*, USENIXSec, 2019.
- [JLLK17] Y. Jang, J. Lee, S. Lee, and T. Kim. Sgx-bomb: Locking down the processor via rowhammer attack. In *Proceedings of the 2nd Workshop on System Software for Trusted Execution*, pages 1–6, 2017.
- [JMRO22] J. Jin, E. McMurtry, B. I. Rubinstein, and O. Ohrimenko. Are we there yet? timing and floating-point attacks on differential privacy systems. In *2022 IEEE Symposium on Security and Privacy*, SP, pages 473–488. IEEE, 2022.
- [JWEG18] B. Jayaraman, L. Wang, D. Evans, and Q. Gu. Distributed learning without distress: Privacy-preserving empirical risk minimization. *Advances in Neural Information Processing Systems*, 31, 2018.

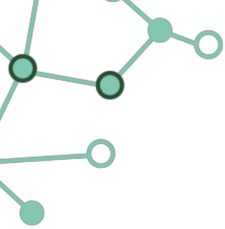




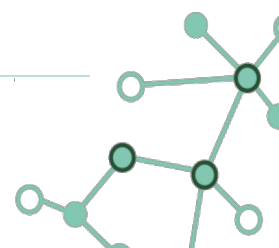
- [JX09] P. Jurczyk and L. Xiong. Distributed anonymization: Achieving privacy for both data subjects and data providers. In *Proc. of DBSec 2009*, Montreal, Canada, July 2009.
- [KCY21] F. Kato, Y. Cao, and M. Yoshikawa. Preventing manipulation attack in local differential privacy using verifiable randomization mechanism. In *Data and Applications Security and Privacy XXXV: 35th Annual IFIP WG 11.3 Conference, DBSec 2021, Calgary, Canada, July 19–20, 2021, Proceedings 35*, pages 43–60. Springer, 2021.
- [Kel20] M. Keller. Mp-spdz: A versatile framework for multi-party computation. Cryptology ePrint Archive, Paper 2020/521, 2020.
- [KKP01] S. Kotz, T. Kozubowski, and K. Podgórski. *The Laplace distribution and generalizations: a revisit with applications to communications, economics, engineering, and finance*. Number 183. Springer Science & Business Media, 2001.
- [KLN<sup>+</sup>11] S. P. Kasiviswanathan, H. K. Lee, K. Nissim, S. Raskhodnikova, and A. Smith. What can we learn privately? *SIAM Journal on Computing*, 40(3):793–826, 2011.
- [KLS21] P. Kairouz, Z. Liu, and T. Steinke. The distributed discrete gaussian mechanism for federated learning with secure aggregation. In *International Conference on Machine Learning, ICML*, pages 5201–5212. PMLR, 2021.
- [KPEK14] F. Kohlmayer, F. Prasser, C. Eckert, and K. A. Kuhn. A flexible approach to distributed data anonymization. *Journal of Biomedical Informatics*, 50:62–76, 2014.
- [KRM20] K. Kapusta, M. Rambaud, and G. Memmi. Revisiting shared data protection against key exposure. In *Proc. of ASIACCS*, Taipei, Taiwan, 2020.
- [KSLC17] G. O. Karame, C. Soriente, K. Lichota, and S. Capkun. Securing cloud data under key exposure. *IEEE TCC*, 7(3):838–849, 2017.
- [Law] A. M. Law. *Simulation modeling and analysis*, volume 3.
- [LDR05] K. LeFevre, D. DeWitt, and R. Ramakrishnan. Incognito: Efficient full-domain  $k$ -anonymity. In *Proc. of SIGMOD 2005*, Baltimore, MA, USA, June 2005.
- [LDR06] K. LeFevre, D. DeWitt, and R. Ramakrishnan. Mondrian multidimensional  $k$ -anonymity. In *Proc. of ICDE 2006*, Atlanta, GA, USA, April 2006.
- [Leg18] C. S. Legislature. California consumer privacy act, 2018.
- [LJJ<sup>+</sup>17] J. Lee, J. Jang, Y. Jang, N. Kwak, Y. Choi, C. Choi, T. Kim, M. Peinado, and B. B. Kang. Hacking in darkness: Return-oriented programming against secure enclaves. In *26th USENIX Security Symposium (USENIX Security 17)*, pages 523–539, 2017.
- [LK17] S. Lee and T. Kim. Leaking uninitialized secure enclave memory via structure padding. *arXiv preprint arXiv:1710.09061*, 2017.

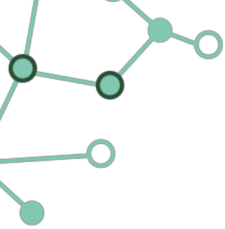




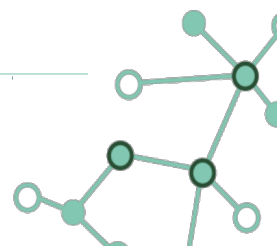


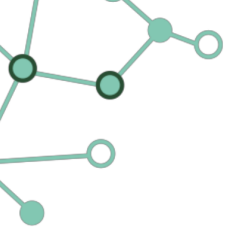
- [LR88] M. Luby and C. Rackoff. How to construct pseudorandom permutations from pseudorandom functions. *SIAM J. Comp.*, 17(2):373–386, April 1988.
- [McK10] W. McKinney. Data structures for statistical computing in Python. In *Proc. of SciPy 2010*, Austin, TX, USA, July 2010.
- [McS09] F. D. McSherry. Privacy integrated queries: an extensible platform for privacy-preserving data analysis. In *Proceedings of the 2009 ACM SIGMOD International Conference on Management of data*, pages 19–30, 2009.
- [MGK06] A. Machanavajjhala, J. Gehrke, and D. Kifer.  $\ell$ -diversity: Privacy beyond  $k$ -anonymity. In *Proc. of ICDE 2006*, Atlanta, GA, USA, April 2006.
- [MGSB22] G. Munilla Garrido, J. Sedlmeir, and M. Babel. Towards verifiable differentially-private polling. In *Proceedings of the 17th International Conference on Availability, Reliability and Security*, pages 1–11, 2022.
- [Mir12] I. Mironov. On significance of the least significant bits for differential privacy. In *Proceedings of the 2012 ACM conference on Computer and communications security, CCS*, pages 650–661, 2012.
- [Mir17] I. Mironov. Rényi differential privacy. In *2017 IEEE 30th computer security foundations symposium, CSF*, pages 263–275. IEEE, 2017.
- [MKGV07] A. Machanavajjhala, D. Kifer, J. Gehrke, and M. Venkatasubramanian.  $\ell$ -diversity: Privacy beyond  $k$ -anonymity. *ACM TKDD*, 1(1):3:1–3:52, 2007.
- [MMR<sup>+</sup>17] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas. Communication-efficient learning of deep networks from decentralized data. In *Artificial intelligence and statistics*, pages 1273–1282. PMLR, 2017.
- [MÖJC23] M. Mansouri, M. Önen, W. B. Jaballah, and M. Conti. Sok: Secure aggregation based on cryptographic schemes for federated learning. *Proceedings on Privacy Enhancing Technologies*, 1:140–157, 2023.
- [MPBB19] V. Mugunthan, A. Polychroniadou, D. Byrd, and T. H. Balch. Smpai: Secure multi-party computation for federated learning. In *Proceedings of the NeurIPS 2019 Workshop on Robust AI in Financial Services*, 2019.
- [MR18] P. Mohassel and P. Rindal. Aby3: A mixed protocol framework for machine learning. In *Proceedings of the 2018 ACM SIGSAC conference on computer and communications security*, pages 35–52, 2018.
- [MRTZ17] H. B. McMahan, D. Ramage, K. Talwar, and L. Zhang. Learning differentially private recurrent language models. *arXiv preprint arXiv:1710.06963*, 2017.
- [MT07] F. McSherry and K. Talwar. Mechanism design via differential privacy. In *Annual IEEE Symposium on Foundations of Computer Science, FOCS*, 2007.



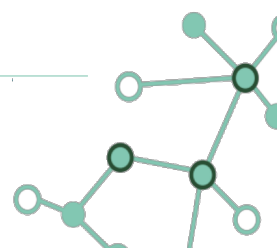


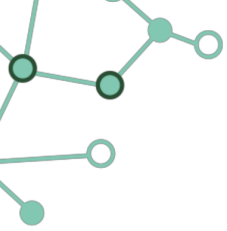
- [Nef03] C. A. Neff. Verifiable mixing (shuffling) of elgamal pairs. *VHTi Technical Document, Vote-Here, Inc*, page 112, 2003.
- [opa23] open-policy agent. gatekeeper-library. <https://github.com/open-policy-agent/gatekeeper-library>, 2023.
- [Org23a] C. Organization. Casbin. <https://casbin.org/>, 2023.
- [Org23b] C. Organization. Casbin – cloud native middlewares. <https://casbin.org/docs/cloud-native/>, 2023.
- [Org23c] C. Organization. Casbin github page. <https://github.com/casbin/casbin>, 2023.
- [Ory23] Ory. Ory keto – permission and role management. <https://www.ory.sh/keto/>, 2023.
- [OS23a] I. Oso Security. Oso: Authorization-as-a-service. <https://www.osohq.com/>, 2023.
- [OS23b] I. Oso Security. Relationship-based access control (rebac). <https://www.osohq.com/academy/relationship-based-access-control-rebac>, 2023.
- [Pai99] P. Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *Advances in Cryptology—EUROCRYPT'99: International Conference on the Theory and Application of Cryptographic Techniques Prague, Czech Republic, May 2–6, 1999 Proceedings 18*, pages 223–238. Springer, 1999.
- [PCB<sup>+</sup>19] R. Pang, R. Caceres, M. Burrows, Z. Chen, P. Dave, N. Germer, A. Golynski, K. Graney, N. Kang, L. Kissner, J. L. Korn, A. Parmar, C. D. Richards, and M. Wang. Zanzibar: Google's consistent, global authorization system. In *2019 USENIX Annual Technical Conference (USENIX ATC '19)*, Renton, WA, 2019.
- [Per23] Permify.co. Permify. <https://permify.co/>, 2023.
- [PHK<sup>+</sup>23] N. Ponomareva, H. Hazimeh, A. Kurakin, Z. Xu, C. Denison, H. B. McMahan, S. Vassilvitskii, S. Chien, and A. Thakurta. How to dp-fy ml: A practical guide to machine learning with differential privacy. *arXiv preprint arXiv:2303.00654*, 2023.
- [Pip23] D. Piper. Data protection laws of the world. <https://www.dlapiperdataprotection.com/index.html?t=world-map&c=AL>, 2023.
- [Pri23] S. J. Prince. *Understanding Deep Learning*. MIT Press, 2023.
- [PRM<sup>+</sup>22] S. Pentyala, D. Railsback, R. Maia, R. Dowsley, D. Melanson, A. Nascimento, and M. De Cock. Training differentially private models with secure multiparty computation. *arXiv preprint arXiv:2202.02625*, 2022.
- [PRR10] M. Pathak, S. Rane, and B. Raj. Multiparty differential privacy via aggregation of locally trained classifiers. *Advances in neural information processing systems*, 23, 2010.
- [PSM<sup>+</sup>18] N. Papernot, S. Song, I. Mironov, A. Raghunathan, K. Talwar, and Ú. Erlingsson. Scalable private learning with pate. *arXiv preprint arXiv:1802.08908*, 2018.



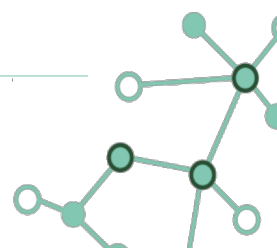


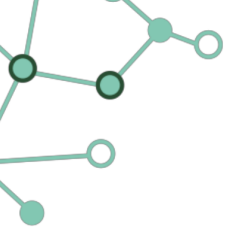
- [QKL<sup>+</sup>19] H. Qiu, K. Kapusta, Z. Lu, M. Qiu, and G. Memmi. All-or-nothing data protection for ubiquitous communication: Challenges and perspectives. *Information Sciences*, 502:434–445, 2019.
- [R<sup>+</sup>20] J. Reback et al. pandas-dev/pandas: Pandas, February 2020. <https://doi.org/10.5281/zenodo.3509134>.
- [Rab05] M. O. Rabin. How to exchange secrets with oblivious transfer. *Cryptology ePrint Archive*, 2005.
- [Red23a] RedHat. Keycloak – authorization services guide. [https://www.keycloak.org/docs/latest/authorization\\_services/index.html](https://www.keycloak.org/docs/latest/authorization_services/index.html), 2023.
- [Red23b] RedHat. Keycloak – open source identity and access management. <https://www.keycloak.org/>, 2023.
- [Reg09] O. Regev. On lattices, learning with errors, random linear codes, and cryptography. *Journal of the ACM (JACM)*, 56(6):1–40, 2009.
- [reg20] Census regions and divisions of the united states, 2020.
- [RFB<sup>+</sup>21] M. Rossi, D. Facchinetti, E. Bacis, M. Rosa, and S. Paraboschi. Seapp: Bringing mandatory access control to android apps. In *Proc. of the 30th USENIX Security Symposium (USENIX Security 21)*. USENIX Association, Aug. 2021.
- [RFG<sup>+</sup>20] S. Ruggles, S. Flood, R. Goeken, J. Grover, E. Meyer, J. Pacas, and M. Sobek. IPUMS USA: Version 10.0 [dataset], 2020. <https://doi.org/10.18128/D010.V10.0>.
- [Riv97] R. Rivest. All-or-nothing encryption and the package transform. In *Proc of FSE*, Haifa, Israel, January 1997.
- [RS60] I. Reed and G. Solomon. Polynomial codes over certain finite fields. *Journal of the Society for Industrial and Applied Mathematics*, 8(2):300–304, June 1960.
- [RXF<sup>+</sup>23] W. Ruan, M. Xu, W. Fang, L. Wang, L. Wang, and W. Han. Private, efficient, and accurate: Protecting models trained by multi-party learning with differential privacy. In *44th IEEE Symposium on Security and Privacy*. IEEE, 2023.
- [SA15] M. J. Saarinen and J.-P. Aumasson. The BLAKE2 cryptographic hash and message authentication code (MAC). RFC 7693, RFC Editor, 11 2015.
- [SA17] U. Sopaoglu and O. Abul. A top-down  $k$ -anonymization implementation for Apache Spark. In *Proc. of IEEE Big Data 2017*, Boston, MA, USA, December 2017.
- [Sam01] P. Samarati. Protecting respondents' identities in microdata release. *IEEE TKDE*, 13(6):1010–1027, November/December 2001.
- [SBM20] S. Sharma, A. Burtsev, and S. Mehrotra. Advances in cryptography and secure hardware for data outsourcing. In *Proc. of ICDE*, Dallas, TX, USA, April 2020.



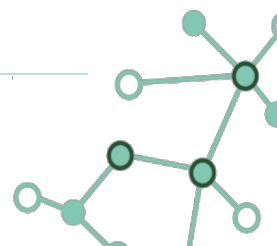


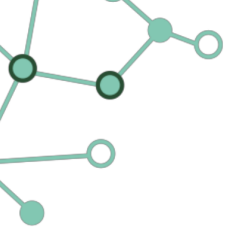
- [SCD<sup>+</sup>21] G. Sun, Y. Cong, J. Dong, Q. Wang, L. Lyu, and J. Liu. Data poisoning attacks on federated machine learning. *IEEE Internet of Things Journal*, 9(13):11365–11375, 2021.
- [SCR<sup>+</sup>11] E. Shi, H. Chan, E. Rieffel, R. Chow, and D. Song. Privacy-preserving aggregation of time-series data. In *Annual Network & Distributed System Security Symposium (NDSS)*. Internet Society., 2011.
- [SCS13] S. Song, K. Chaudhuri, and A. D. Sarwate. Stochastic gradient descent with differentially private updates. In *2013 IEEE global conference on signal and information processing*, pages 245–248. IEEE, 2013.
- [Sha79] A. Shamir. How to share a secret. *Communications of the ACM*, 22(11):612–613, 1979.
- [Sha23] S. Sharma. Support your organization’s gdpr initiatives by securing the digital workspace. <https://blogs.vmware.com/euc/2017/09/accelerate-towards-gdpr-compliance.html>, 2023.
- [SSSS17] R. Shokri, M. Stronati, C. Song, and V. Shmatikov. Membership inference attacks against machine learning models. In *2017 IEEE symposium on security and privacy (SP)*, pages 3–18. IEEE, 2017.
- [SSV<sup>+</sup>22] T. Stevens, C. Skalka, C. Vincent, J. Ring, S. Clark, and J. Near. Efficient differentially private secure aggregation for federated learning via hardness of learning with errors. In *31st USENIX Security Symposium (USENIX Security 22)*, pages 1379–1395, 2022.
- [Ste98] J. P. Stern. A new and efficient all-or-nothing disclosure of secrets protocol. In *Advances in Cryptology—ASIACRYPT’98: International Conference on the Theory and Application of Cryptology and Information Security Beijing, China, October 18–22, 1998 Proceedings*, pages 357–371. Springer, 1998.
- [Sty23a] Styra. Integrating OPA Documentation. <https://www.openpolicyagent.org/docs/latest/integration/><https://www.openpolicyagent.org/docs/latest/>, 2023.
- [Sty23b] Styra. Open policy agent. <https://www.openpolicyagent.org/>, 2023.
- [Sty23c] Styra. Open policy agent documentation. <https://www.openpolicyagent.org/docs/latest/>, 2023.
- [Sty23d] Styra. Overview and Architecture. <https://www.openpolicyagent.org/docs/latest/envoy-introduction/>, 2023.
- [Sty23e] Styra. Policy language. <https://www.openpolicyagent.org/docs/latest/policy-language/>, 2023.
- [Sty23f] Styra. Write Policy in OPA. Enforce Policy in SQL. <https://blog.openpolicyagent.org/write-policy-in-opa-enforce-policy-in-sql-d9d24db93bf4>, 2023.



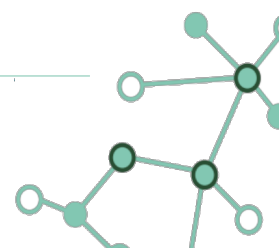


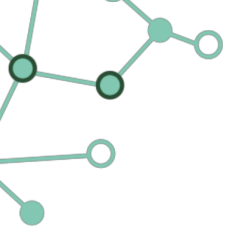
- [SWG<sup>+</sup>17] M. Schwarz, S. Weiser, D. Gruss, C. Maurice, and S. Mangard. Malware guard extension: Using sgx to conceal cache attacks. In *Detection of Intrusions and Malware, and Vulnerability Assessment: 14th International Conference, DIMVA 2017, Bonn, Germany, July 6-7, 2017, Proceedings 14*, pages 3–24. Springer, 2017.
- [TB20] F. Tramer and D. Boneh. Differentially private learning needs better features (or much more data). *arXiv preprint arXiv:2011.11660*, 2020.
- [TBA<sup>+</sup>19] S. Truex, N. Baracaldo, A. Anwar, T. Steinke, H. Ludwig, R. Zhang, and Y. Zhou. A hybrid approach to privacy-preserving federated learning. In *Proceedings of the 12th ACM workshop on artificial intelligence and security*, pages 1–11, 2019.
- [TCG15] TCG storage security subsystem class: Opal, August 2015.
- [Tea17] A. D. P. Team. Learning with privacy at scale, 2017.
- [TG12] T. Tassa and E. Gudes. Secure distributed computation of anonymized views of shared databases. *ACM TODS*, 37(2):1–43, 2012.
- [TJGE16] L. Tian, B. Jayaraman, Q. Gu, and D. Evans. Aggregating private sparse learning models using multi-party computation. In *NIPS Workshop on Private Multi-Party Machine Learning*, 2016.
- [TP17] J. Tomcy and M. Pankaj. *Data Lake for Enterprises*. Packt Publishing, May 2017.
- [TTP18] F. Taigel, A. K. Tueno, and R. Pibernik. Privacy-preserving condition-based forecasting using machine learning. *Journal of Business Economics*, 88:563–592, 2018.
- [TZJ<sup>+</sup>16] F. Tramèr, F. Zhang, A. Juels, M. K. Reiter, and T. Ristenpart. Stealing machine learning models via prediction apis. In *USENIX security symposium*, volume 16, pages 601–618, 2016.
- [VBWK<sup>+</sup>17] J. Van Bulck, N. Weichbrodt, R. Kapitza, F. Piessens, and R. Strackx. Telling your secrets without page faults: Stealthy page {Table-Based} attacks on enclaved execution. In *26th USENIX Security Symposium (USENIX Security 17)*, pages 1041–1056, 2017.
- [VC02] J. Vaidya and C. Clifton. Privacy preserving association rule mining in vertically partitioned data. In *Proc. of ACM KDD 2002*, Edmonton, Alberta, Canada, July 2002.
- [VGSR18] P. Vepakomma, O. Gupta, T. Swedish, and R. Raskar. Split learning for health: Distributed deep learning without sharing raw patient data. *arXiv preprint arXiv:1812.00564*, 2018.
- [WCLJ18] J. Wang, Y. Cheng, Q. Li, and Y. Jiang. Interface-based side channel attack against intel sgx. *arXiv preprint arXiv:1811.05378*, 2018.
- [WDJ<sup>+</sup>22] J. Wang, R. Das, G. Joshi, S. Kale, Z. Xu, and T. Zhang. On the unreasonable effectiveness of federated averaging with heterogeneous data. *arXiv preprint arXiv:2206.04723*, 2022.





- [WHMM21] S. Wagh, X. He, A. Machanavajjhala, and P. Mittal. Dp-cryptography: marrying differential privacy and cryptography in emerging applications. *Communications of the ACM*, 64(2):84–93, 2021.
- [WJS20] L. Wang, R. Jia, and D. Song. D2p-fed: Differentially private federated learning with efficient communication. *arXiv preprint arXiv:2006.13039*, 2020.
- [WMK16] X. Wang, A. J. Malozemoff, and J. Katz. EMP-toolkit: Efficient MultiParty computation toolkit. <https://github.com/emp-toolkit>, 2016.
- [XMW<sup>+</sup>21] S. Xie, M. Mohammady, H. Wang, L. Wang, J. Vaidya, and Y. Hong. A generalized framework for preserving both privacy and utility in data outsourcing. *IEEE TKDE*, 2021. pre-print.
- [XNH<sup>+</sup>22] S. Xu, J. Ning, X. Huang, Y. Li, and G. Xu. Untouchable once revoking: A practical and secure dynamic EHR sharing system via cloud. *IEEE TDSC*, 19(6):3759–3772, November–December 2022.
- [XT06] X. Xiao and Y. Tao. Anatomy: Simple and effective privacy preservation. In *Proc of VLDB 2006*, Seoul, South Korea, September 2006.
- [XWP<sup>+</sup>06] J. Xu, W. Wang, J. Pei, X. Wang, B. Shi, and A.-C. Fu. Utility-based anonymization for privacy preservation with less information loss. *ACM SIGKDD Explorations Newsletter*, 8(2):21–30, 2006.
- [Yao86] A. C.-C. Yao. How to generate and exchange secrets. In *27th annual symposium on foundations of computer science (Sfcs 1986)*, pages 162–167. IEEE, 1986.
- [YGFJ18] S. Yeom, I. Giacomelli, M. Fredrikson, and S. Jha. Privacy risk in machine learning: Analyzing the connection to overfitting. In *2018 IEEE 31st computer security foundations symposium (CSF)*, pages 268–282. IEEE, 2018.
- [YWRL10] S. Yu, C. Wang, K. Ren, and W. Lou. Attribute based data sharing with attribute revocation. In *Proc. of ASIACCS*, Beijing, China, April 2010.
- [ZDX<sup>+</sup>20] Y. Zhang, R. Deng, S. Xu, J. Sun, Q. Li, and D. Zheng. Attribute-based encryption for cloud computing access control: A survey. *ACM Computing Surveys*, 2020. (to appear).
- [ZDX<sup>+</sup>21] Y. Zhang, R. H. Deng, S. Xu, J. Sun, Q. Li, and D. Zheng. Attribute-based encryption for cloud computing access control: A survey. *ACM Computing Surveys*, 53(4):83:1–83:41, 2021.
- [ZLD<sup>+</sup>16] X. Zhang, C. Leckie, W. Dou, J. Chen, R. Kotagiri, and Z. Salcic. Scalable local-recoding anonymization using locality sensitive hashing for big data privacy preservation. In *Proc. of CIKM 2016*, Indianapolis, IN, USA, October 2016.
- [ZQD<sup>+</sup>22] X. Zhang, L. Qi, W. Dou, Q. He, C. Leckie, R. Kotagiri, and Z. Salcic. MRMondrian: Scalable multidimensional anonymisation for big data privacy preservation. *IEEE TBD*, 8(1):125–139, 2022.





- [ZXD<sup>+</sup>18] Z. Zheng, S. Xie, H.-N. Dai, X. Chen, and H. Wang. Blockchain challenges and opportunities: A survey. *International journal of web and grid services*, 14(4):352–375, 2018.
- [ZYL13] X. Zhang, L. T. Yang, C. Liu, and J. Chen. A scalable two-phase top-down specialization approach for data anonymization using MapReduce on cloud. *IEEE TPDS*, 25(2):363–373, 2013.
- [ZZX<sup>+</sup>12] J. Zhang, Z. Zhang, X. Xiao, Y. Yang, and M. Winslett. Functional mechanism: regression analysis under differential privacy. *arXiv preprint arXiv:1208.0219*, 2012.

