Privacy-Preserving Distributed Data Mining

by

Josenildo Costa da Silva, M.Sc.

Saarbrücken, 2022

Privacy-Preserving Distributed Data Mining

A dissertation submitted towards the degree
Doctor of Engineering (Dr.-Ing.) of the
Faculty of Mathematics and Computer Science
of Saarland University

by

Josenildo Costa da Silva, M.Sc.

Saarbrücken, 2022

Kolloquium	7th December, 2022
Dean of the Faculty	Univ.-Prof. Dr. Jürgen Steimle
Chair of the Committee	Prof. Dr.-Ing. Philipp Slusallek
Academic Assistant	Dr. Elena Gloria Jaramillo
Reporters	
First reviewer:	PD. Dr. rer. nat. Matthias Klusch
Second reviewer:	Prof. Dr.-Ing. Wolfgang Maaß
External reviewer:	Prof. Dr. Stefano Lodi (University of Bologna, Italy)

À Lene, por tudo.

Abstract

This thesis is concerned with privacy-preserving distributed data mining algorithms. The main challenges in this setting are inference attacks and the formation of collusion groups. *The inference problem* is the reconstruction of sensitive data by attackers from non-sensitive sources, such as intermediate results, exchanged messages, or public information. Moreover, in a distributed scenario, malicious insiders can organize *collusion groups* to deploy more effective inference attacks. This thesis shows that existing privacy measures do not adequately protect privacy against inference and collusion. Therefore, in this thesis, new measures based on information theory are developed to overcome the identified limitations. Furthermore, a new distributed data clustering algorithm is presented. The clustering approach is based on a kernel density estimates approximation that generates a controlled amount of ambiguity in the density estimates and provides privacy to original data. Besides, this thesis also introduces the first privacy-preserving algorithms for frequent pattern discovery in a distributed time series. Time series are transformed into a set of n-dimensional data points and finding frequent patterns reduced to finding local maxima in the n-dimensional density space. The proposed algorithms are linear in the size of the dataset with low communication costs, validated by experimental evaluation using different datasets.

Summary

This thesis is concerned with privacy-preserving distributed data mining algorithms (PPDDM) in environments where a group of insiders can try to deploy inference attacks against other peers in the mining group. In a distributed data context, any participant may try to infer sensitive information about data owned by other parties from intermediate results and other messages exchanged during the mining session. The reconstruction of sensitive data by attackers from non-sensitive sources is known as *the inference problem* and was first studied in the field of databases and subsequently by the data mining community. In a distributed scenario, the inference problem is even more challenging since malicious insiders can organize *collusion groups* to deploy more efficient inference attacks. Often, mathematical properties allow for reconstructions, like filtering out random noise or inverting a given data transformation.

This thesis shows that existing privacy measures do not satisfy all required properties for privacy protection in distributed data settings with inference and collusion. Moreover, we propose new and improved measures and apply them to some representative privacy-preserving algorithms. The new measures are based on information theory and can detect vulnerabilities of selected algorithms to collusion in different attack scenarios.

This thesis analyzes an existing distributed data clustering algorithm, the KDEC algorithm, to understand how it performs under an inference attack. KDEC follows a density-based clustering approach and uses a kernel function to estimate the density of a dataset. As the analysis pointed out, a possible inference attack against KDEC is achieved by inverting the kernel function to reconstruct original data. To overcome this class of attacks, we developed the KDEC-S algorithm. In KDEC-S, the original kernel is substituted by a kernel approximation function. As a result, a controlled ambiguity level

is introduced during the density estimation phase. KDEC-S provides more privacy than KDEC keeping the same mining quality. It is linear in the number of data points in the dataset and has low communication costs, validated through several experiments using different datasets.
Furthermore, in this thesis, we present the first privacy-preserving algorithms for frequent pattern discovery in distributed time series: DPD-TS, DPD-HE, and DPD-FS. All algorithms first transform time series into a set of n-dimensional data points and, then, reduce the problem of finding frequent patterns to the problem of finding local maxima in the n-dimensional density landscape. Additionally, the n-dimensional data points are discretized to form strings from a given alphabet. The density is then computed on discrete data points, which are generally much smaller than the original dataset size. Different approaches to density estimation were investigated, using heuristics to prune the number of data points generated from the original time series. Their privacy properties considering different inference attack scenarios were also investigated, and it was shown that the proposed algorithms provide parameter-controlled privacy levels. The algorithms are linear in the size of a time series and have low communication costs. The approach is validated by experimental evaluation using different datasets.

Zusammenfassung

Diese Arbeit befasst sich mit vertraulichkeitsbewahrendem Data Mining in verteilten Umgebungen mit Schwerpunkt auf ausgewählten N-Agenten-Angriffsszenarien für das Inferenzproblem im Data-Clustering und der Zeitreihenanalyse. Dabei handelt es sich um Angriffe von einzelnen oder Teilgruppen von Agenten innerhalb einer verteilten Data Mining-Gruppe oder von einem einzelnen Agenten außerhalb dieser Gruppe. Zunächst werden in dieser Arbeit zwei neue Privacy-Maße vorgestellt, die im Gegensatz zu bislang existierenden, die im verteilten Data Mining allgemein geforderte Eigenschaften zur Vertraulichkeitsbewahrung erfüllen und bei denen sich der gemessene Grad der Vertraulichkeit auf die verwendete Datenanalysemethode und die Anzahl von Angreifern bezieht.

Für den Zweck eines vertraulichkeitsbewahrenden, verteilten Data-Clustering wird ein neues Kernel-Dichteabschätzungsbasiertes Verfahren namens KDEC-S vorgestellt. KDEC-S verwendet eine Approximation der originalen, lokalen Kernel-Dichteschätzung, so dass die ursprünglichen Daten anderer Agenten in der Data Mining-Gruppe mit einer höheren Wahrscheinlichkeit als einem hierfür vorgegebenen Wert nicht mehr zu rekonstruieren sind. Das Verfahren ist nachweislich sicherer als Data-Clustering mit generativen Mixture Modellen und SMC-basiert sicherem k-means Data-Clustering.

Zusätzlich stellen wir neue Verfahren, namens DPD-TS, DPD-HE und DPD-FS, für eine vertraulichkeitsbewahrende, verteilte Mustererkennung in Zeitreihen vor, deren Komplexität und Sicherheitsgrad wir mit den zuvor erwähnten neuen Privacy-Maßen analysieren. Dabei hängt ein von einzelnen Agenten einer Data Mining-Gruppe jeweils vorgegebener, minimaler Sicherheitsgrad von DPD-TS und DPD-FS nur von der Dimensionsreduktion der Zeitreihenwerte und ihrer Diskretisierung ab und kann leicht überprüft werden. Einen noch besseren Schutz von sensiblen Daten bietet das Verfahren DPD-

HE mit Hilfe von homomorpher Verschlüsselung. Neben der theoretischen Analyse wurden die experimentellen Leistungsbewertungen der entwickelten Verfahren mit verschiedenen, öffentlich verfügbaren Datensätzen durchgeführt.

Acknowledgments

First of all, I would like to thank my advisor PD. Dr. rer. nat. Matthias Klusch. It is impossible to adequately express my gratitude for the clear guidance and for providing me with a stimulating research environment.

I would also like to thank all of my colleagues at the Multi-Agent Systems group (MAS), DFKI, where I spent much of my research discussing, exploring, and learning. Furthermore, I thank the people with whom I had the pleasure of cooperating. I am particularly grateful for the insightful discussions with Chris Giannella, Stefano Lodi, William Cheung, and Hillol Kargupta.

I also thank CAPES/Brazil for the scholarship, while DAAD also deserves my thanks as they provide me with a 6-month German course at Goethe Institut in Mannheim. It helped a lot to learn this *verdammt schwierige Sprache*.

I received an incredible amount of support from my friends, both in Germany and Brazil. Thank you guys very much for all the good times we experienced together. Also, thanks to all my friends at Computer Science Department (DComp/IFMA) for your motivation, discussions, and all that coffee.

Undoubtedly, none of this would have been possible without the support and love of my family. In particular, I would like to express my gratitude to my parents, who showed me that only concrete actions make dreams become a reality. I am also deeply thankful to my wife for her love and patience over all these years. I am very grateful to God for providing all of the random events that made this work possible.

Thank you all!

Josenildo C. da Silva

Publication List

Portions of the material reported in this thesis appeared in the following publications[1]:

1. J. C. da Silva, G. H. B. S. Oliveira, S. Lodi, and M. Klusch (2017): Clustering Distributed Short Time Series with Dense Patterns. Proceedings of the 16th IEEE International Conference on Machine Learning and Applications. Cancun, Mexico. IEEE CS Press. [Qualis B1]

2. J. C. da Silva, M. Klusch and S. Lodi (2016): Privacy-Awareness of Distributed Data Clustering Algorithms Revisited. Proceedings of 15th International Symposium on Intelligent Data Analysis (IDA). LNCS vol. 9897. Stockholm, Sweden. Springer. [CORE A]

3. J. C. da Silva, O. A. C. Cortes, G. H. B. S. Oliveira and M. Klusch (2012): Density-Based Pattern Discovery in Distributed Time Series. Proceedings of 21st Brazilian Symposium on Artificial Intelligence (SBIA). LNAI vol. 7589, pp. 62–71. Curitiba, PR, Brazil. Springer. [Qualis B2]

4. J. C. da Silva and M. Klusch (2007): Privacy-Preserving Discovery of Frequent Patterns in Time Series. Proceedings of 7th Industrial Conference on Data Mining (ICDM), LNCS vol. 4597, pp. 318-328. Leipzig, Germany. Springer.

5. J. C. da Silva and M. Klusch (2007): Privacy-Preserving Pattern Discovery in Distributed Time Series. Proceedings of 3rd International

[1] The conference or journal ranking is given in square brackets. The CORE ranking was created by Australian deans and the Australian Computing Research and Education Association of Australasia. The Qualis conference ranking has been published by the Brazilian ministry of education and uses the H-index as performance measure for conferences.

Workshop on Privacy Data Management (PDM), in conj. with 23rd Conference on Data Engineering (ICDE), pp. 207-214. Istanbul, Turkey. IEEE CS Press.

6. J. C. da Silva and M. Klusch, S. Lodi and G. L. Moro (2006): Privacy-Preserving Agent-Based Distributed Data Clustering. *Web Intelligence and Agent Systems*, vol. 4(2), pp. 221 - 238. IOS Press. [Qualis B3]

7. J. C. da Silva and M. Klusch (2006): Inference in Distributed Data Clustering. *Engineering Applications of Artificial Intelligence*, vol 19(4), pp. 363-369, June. [Qualis B1, CORE B]

8. J. C. da Silva and M. Klusch (2005): Inference on Distributed Data Clustering. Proc. of the 4th Intl. Conf. on Machine Learning and Data Mining (MLDM), LNCS vol. 3587, pp. 610-619. Leipzig, Germany. Springer. [Qualis B2]

9. J. C. da Silva, C. Giannella, R. Bhargava, H. Kargupta and M. Klusch (2005): Distributed Data Mining and Agents. *Engineering Applications of Artificial Intelligence*, vol. 18(7), pp. 791–807, October. [Qualis B1, CORE B]

10. J. C. da Silva, M. Klusch, S. Lodi and G. L. Moro (2004): Inference Attacks in Peer-to-peer Homogeneous Data Mining. Proc. of European Conference on Artificial Intelligence (ECAI), pp. 450-454. Valencia, Spain. IOS Press [Qualis A2, CORE A]

Contents

Summary iii

Zusammenfassung v

Acknowledgments vii

Publication List ix

1 Introduction 1

1.1 Motivation 1

1.2 Research Questions 4

1.3 Contributions 4

1.4 Thesis Overview 7

2 Background 9

2.1 Data Mining 9

2.2 Distributed Data Mining 12

2.2.1 Distributed Data Clustering 14

2.2.2 Distributed Time Series Mining 17

2.2.3 Distributed Data Mining Systems 18

2.3 Privacy Issues in Distributed Data Mining 20

2.3.1 Privacy, Confidentiality and Sensitive Data 22

2.3.2 The Inference Problem 25

2.3.3 Privacy-Preserving Approaches for Distributed Data . 32

2.3.4 Secure Multi-Party Computation 33

2.3.5 Distributed Model Aggregation 44

2.3.6 Perturbation-Based Approaches 52

2.4 Summary 54

3 Privacy Measures Revisited **57**
3.1 Privacy: Properties and Notations 58
3.2 Analysis of Existing Privacy Measures 60
3.2.1 Private Computation Measure for SMC 60
3.2.2 Likelihood-Based Measure 61
3.3 New Privacy Measures for Clustering and Time Series Mining 64
3.3.1 Privacy Measures for Distributed Clustering 64
3.3.2 Privacy Measure for Time Series Mining 73
3.4 Privacy Analysis with Inference and Collusion Attack Scenarios 77
3.4.1 Secure Multi-Party k-Means Clustering 78
3.4.2 Elliptic Curves-Based k-Means 83
3.4.3 Generative Models for Privacy-Preserving Clustering . 84
3.4.4 Information Theoretical Approach to DDC 88
3.4.5 Average Consensus-Based Clustering 89
3.5 Summary 91

4 Privacy-Preserving Distributed Data Clustering **95**
4.1 Clustering Distributed Sensitive Data 96
4.2 KDEC Algorithm 97
4.2.1 Algorithm Overview 97
4.2.2 Complexity Analysis 102
4.2.3 Inference and Collusion Attacks Analysis 104
4.3 KDEC-S Algorithm 114
4.3.1 Algorithm Overview 115
4.3.2 Complexity Analysis 120
4.3.3 Inference and Collusion Attacks Analysis 121
4.3.4 Experimental Evaluation 126
4.4 Related Work and Discussion 130
4.5 Summary 136

5 Privacy-Preserving Distributed Time Series Mining **139**
5.1 Distributed Pattern Discovery in Time Series 141
5.2 DPD-TS Algorithm 148
5.2.1 Algorithm Overview 149
5.2.2 Complexity Analysis 156
5.2.3 Inference and Collusion Attacks Analysis 157
5.2.4 Experimental Evaluation 162

5.3 DPD-HE Algorithm . . . 170
5.3.1 Algorithm Overview . . . 172
5.3.2 Complexity Analysis . . . 174
5.3.3 Inference and Collusion Attacks Analysis . . . 176
5.3.4 Experimental Evaluation . . . 178
5.4 DPD-FS Algorithm . . . 179
5.4.1 Algorithm Overview . . . 181
5.4.2 Complexity Analysis . . . 184
5.4.3 Inference and Collusion Attacks Analysis . . . 185
5.4.4 Experimental Evaluation . . . 188
5.5 Application to Genomic Data . . . 194
5.6 Related Work and Discussion . . . 194
5.7 Summary . . . 198

6 pp-Expert: Evaluation Tool 201
6.1 Overview of pp-Expert . . . 201
6.1.1 Installation . . . 201
6.1.2 Features . . . 202
6.2 Architecture . . . 203
6.2.1 pp-Expert Layers . . . 203
6.2.2 Class Interactions . . . 207
6.3 Running an Experiment on pp-Expert . . . 208
6.4 Datasets . . . 210
6.5 Availability . . . 213

7 Conclusion and Outlook 215
7.1 Summary of Contributions . . . 216
7.2 Further Directions . . . 218

Bibliography 221

Index 247

List of Figures

2.1 KDD phases showing its sub-steps. 10
2.2 Data mining tasks. 11
2.3 Homogeneous data distribution 13
2.4 Heterogeneous data distribution 14
2.5 Distributed data clustering scenario 15
2.6 Global clusters 15
2.7 Distributed data mining scenario 30
2.8 Inference attack in a distributed scenario 30
2.9 Insider attack scenarios 31
2.10 Outsider attack scenario 32
2.11 Mixed attack scenario 32
2.12 Two Gaussians fitted from dataset D (cf. Example 2.8) 45
2.13 Different models for a given dataset 48

3.1 Example of a one-dimensional cluster 66

4.1 Density estimates with Gaussian kernels 98
4.2 LDE at site j 100
4.3 Sampled LDE at site j 100
4.4 Sampled GDE at the Helper site 100
4.5 Reconstructed GDE at Site j 100
4.6 Original and reconstructed data 108
4.7 $\psi_{f,\mathbf{v}}$ of the Gaussian function. 116
4.8 Ambiguity of transformation 117
4.9 Gaussian dataset 127
4.10 Polar dataset 127
4.11 S1 dataset 127
4.12 Spiral dataset 127

4.13 Four clusters found in `Gaussian` dataset. 128
4.14 Clusters found in `Polar` dataset 128
4.15 Clusters found in `S1` dataset 128
4.16 Clusters found in `Spiral` dataset 128
4.17 Error rate in `Gaussian` dataset 129
4.18 Error in `Polar` dataset 129
4.19 Error in `S1` dataset 129
4.20 Error in `Spiral` dataset 129
4.21 Running time as a function of dataset size 130
4.22 Running time as a function of privacy parameter τ 130

5.1 Occurrence set relative to given subsequences in a time series. 143
5.2 All subsequences from T that are inside a ball radius r 144
5.3 Trivial matches for a given subsequence Q 145
5.4 A toy time series with pattern occurrences in highlight. 146
5.5 Time series dimension reduction 151
5.6 Reduced subsequence is composed of w average values. 152
5.7 Breakpoints for a 4-symbol alphabet 152
5.8 4-symbol discretization example 153
5.9 Excerpt of power data. 163
5.10 Frequent patterns found by DPD-TS in different datasets. . . 165
5.11 Running time as a function of time series size 166
5.12 Running time as a function of subsequence size 167
5.13 Running time as function of pattern size w 168
5.14 Privacy level as a function of alphabet size 170
5.15 Information loss as a function of subsequence size n. 171
5.16 Running time as a function of security parameter mod length. 179
5.17 Time as a function of dataset size for fixed mod length 180
5.18 Example of pattern densities 183
5.19 Frequent patterns found by DPD-FS in different datasets. . . 190
5.20 Running time vs. time series size. 191
5.21 Running time as function of subsequence size. 192
5.22 Running time as function of pattern size w 193

6.1 pp-Expert's graphical user interface. 203
6.2 Overview of layers and packages 204
6.3 Elements of pp-Expert GUI 205

6.4 pp-Expert's Results Tab 206
6.5 Sequence diagram of KDEC 207
6.6 `Gaussian` dataset 211
6.7 `Polar` dataset 211
6.8 `S1` dataset 211
6.9 `Spiral` dataset 211
6.10 Time series datasets. 212

List of Tables

2.1 Comparison of DDM Systems. 21
2.2 SMC-based Privacy-preserving Clustering Approaches 43
2.3 Summary of privacy measures for distributed data mining . . 56

3.1 Summary of privacy measures 64
3.2 Summary of New Privacy measures 91
3.3 Summary of privacy-preserving DDC algorithms 92

4.1 Summary of PP-DDC algorithms 138

5.1 Running time (in sec.) for different time series sizes. 164
5.2 Minimum interval between breakpoints 169
5.3 Running time data for different time series sizes. 189
5.4 Summary of PP-DDC algorithms 199

List of Algorithms

3.1 Secure Multi-Party k-Means 78
3.2 SMC Closest Centroid 79
3.3 Distributed Data Clustering with Generative Models 85
4.1 DECluster 101
4.2 DECluster's Auxiliary Functions 102
4.3 KDEC: Arbitrary Site 103
4.4 KDEC: Helper Site 103
4.5 Pointhunt 106
4.6 Pointhunt's Auxiliary Functions 106
4.7 KDEC-S: Local Peer 119
4.8 KDEC-S: Helper 120
5.1 DPD-TS: Initiator 150
5.2 DPD-TS: Arbitrary Party 151
5.3 DPD-HE: Initiator 172
5.4 DPD-HE: Arbitrary Party 173
5.5 DPD-FS: Initiator 181
5.6 DPD-FS: Arbitrary Party 182

Chapter 1

Introduction

This thesis investigates how to extract valuable knowledge from distributed data sources without compromising the privacy requirements of sensitive data during the mining process. This question is part of the field known as privacy-preserving distributed data mining and has been an active area of research for many decades now. The main challenge in this setting is that even if the original data are not directly disclosed, a skilled attacker may still manage to reconstruct original sensitive data to some extent via inference attacks. Moreover, the attackers may be one of the mining partners and may organize themselves in collusion groups to improve their chance at reconstructing sensitive data. This is an important line of investigation with several potential applications in many fields. We discuss details of our motivation, research questions, and main contributions in the following.

1.1 Motivation

Distributed data mining (DDM) is a research field concerned with developing algorithms to extract knowledge from distributed data sources. In many real-world scenarios, datasets are intrinsically distributed across different companies, governments, or organizations, with loosely coupled sites connected by a network. Good examples are the biomedical informatics fields [162] and health care. Data from millions of patients have already been collected and stored in an electronic format [225], e-Health mobile applications [11, 33, 85], large scale medical studies [142], mobile crowdsensing [39] and internet of medical things [177, 180]. Smart grids applications [223] and

the Internet of things (IoT) are also interesting application scenarios with distributed data and huge economic potential [115, 209].

Distributed data scenarios raise many challenges concerning the privacy preservation of sensitive data [2, 32, 122]. For example, various countries have laws and regulations to control how data should be collected and distributed among different parties, be it institutions or companies. Privacy-preserving distributed data mining (PPDDM) addresses the question of how to extract meaningful knowledge from distributed data sources without jeopardizing the privacy of sensitive data.

The main approaches to PPDDM are secure multi-party computation (SMC), distributed model aggregation (DMA) mining and local differential privacy (LDP). In SMC-based data mining, the idea is to devise secure protocols for data mining tasks adapting well known algorithms like decision trees [124, 37], clustering [8, 27, 80, 185], expectation maximization clustering [129]. DMA approach addresses PPDDM by building local models that are aggregated into a global model. This approach has been used for classification [107, 149, 243] and clustering [22, 108, 118, 184]. Local differential privacy protects privacy by adding noise to original data before using it to build a model [40, 227]. LDP has been applied to a wide variety of models and mining tasks including classification [228, 107], clustering [193, 227] and time series mining [19, 224]. Actually, many works apply differential privacy together with SMC or DMA [229] as an extra layer of protection.

A critical issue in PPDDM is that participants may learn about sensitive data owned by other parties during the protocol. The reconstruction of sensitive data by attackers from non-sensitive sources is known as **the inference problem** and was first studied in the field of databases [70, 201] and subsequently by the data mining community [14, 90, 215, 187]. Additionally, in a distributed scenario, malicious insiders can organize **collusion groups** to deploy even more efficient inference attacks. We discuss this problem in further details in Section 2.3.2 (p. 25).

SMC-based approaches, for instance, protect sensitive information owned by each participant from direct disclosure via cryptography but do not address inferences from the protocol output [109, 132]. For example, in a protocol where three parties compute the sum of numbers in a secure multiparty protocol, the process does not leak directly any input information. However, when a subset of parties collude, they can subtract their contribution and

learn the input of the remaining party. Unlike SMC, distributed model aggregation approaches do not utilize cryptography and avoid sensitive data disclosure by exchanging only partial models of local datasets. However, in some circumstances, the privacy of single points may be compromised. For example, in [149] the dataset privacy is based on the average privacy of all points. Some points will, of course, have a privacy level much lower than the average privacy level. In this case, an inference attack by an insider might be able to reconstruct these low privacy points with high accuracy.

This investigation focuses on privacy-preserving distributed data mining algorithms in environments where a group of insiders can try to deploy inference attacks against other peers in the mining group. In particular, this research concentrates on distributed data clustering (DDC) and pattern discovery in distributed time series mining (DTS) algorithms. The goal is to provide DDC and DTS with low network traffic, good data mining quality, and high privacy preservation. We assume a network of peers, each of which owns a local dataset with access denied to other peers.

This thesis investigates current privacy-preserving measures for SMC and distributed model aggregation approaches and identifies their limitations when applied to collusion groups and inference attacks scenarios. We then propose a set of formal properties to capture the requirements a privacy measure needs to fulfill and show that the current privacy measures fail to meet at least one of these requirements. From this analysis, new and improved measures for DDC and DTS are developed. Besides, the new measures are applied to some representative privacy-preserving distributed algorithms making explicit their vulnerabilities to inference and collusion.

Further, new algorithms for DDC and DTS are proposed and their privacy-preserving properties analyzed concerning the new measures. The proposed algorithms follow a model-based approach, first computing local density estimates at each peer and then generating a global model at a distinguished peer, making it available to all peers in the mining group. Our central hypothesis in this investigation is that a sample of density estimates can be utilized as surrogate data to perform DDC and DTS. Density estimation is a non-parametric approach to compute a probability density function given a dataset. A key observation is that density estimates are additive. Therefore, we can compute a global sample of the estimates from local estimates. With this approach, we reduce the bandwidth requirements and avoid publishing

sensitive data. The mining step works with samples of density estimates to build the mining results.

As the main result, this investigation shows that density estimates effectively attain all three goals: reduce bandwidth, provide good mining results and guarantee privacy to sensitive data. Additionally, our algorithms are linear in the size of datasets and scale well on the number of parties in the mining session.

1.2 Research Questions

In the context of the discussion above, the investigation presented in this thesis is driven by the following research questions in privacy-preserving distributed data mining, particularly DDC and DTS:

1. **Privacy Measures for DDC and DTS Mining**

   How to define and formalize the concept of privacy and corresponding privacy measure in a distributed environment taking into account inference attacks and collusion of malicious insiders? What kind of attacks to sensitive data owned by each participant in a distributed data mining setting can take place?

2. **Privacy-Preserving Distributed Mining Algorithms**

   How to develop an algorithm that provides the desired privacy level of sensitive data, particularly in DDC and DTS, during the mining sessions? What kind of data transformations or surrogate data can be used instead of the original sensitive data while maintaining the quality of mining results to the desired level? Can an algorithm be privacy-preserving even without cryptography-based protocols? Can the algorithm be privacy-preserving while being scalable in the number of parties and the size of datasets?

1.3 Contributions

The main contributions of this thesis are as follows:

1. **Privacy Measures for DDC and DTS Mining**

Different existing formalizations of privacy, its assumptions, and limitations are investigated. We start with a set of formal properties and show that existing privacy measures do not satisfy all required properties for privacy measures in distributed environments with collusion. Therefore, new and improved measures are developed and applied to some representative privacy-preserving algorithms [51]. The new measures are based on information theory, using the concept of entropy as a measure of uncertainty. These measures model privacy as the size of an interval from where values of a given random variable can be drawn. Some identified benefits from the new measures are detecting the vulnerabilities of selected algorithms to collusion in different attack scenarios and detecting point-level privacy breaches. The new measures are used in subsequent chapters (Ch. 4 and 5).

This work also discusses two main threats to privacy (Ch. 2, Sec. 2.3): *inference attacks* and *collusion groups*. Inference attacks allow the attacker to reconstruct sensitive data from any piece of data exchanged among the parties during a mining session. A collusion group consists of malicious peers who cooperate to improve their attacks. A successful inference attack reconstructs a given sensitive dataset with little or no distortion. Often, mathematical properties reveal possible reconstructions, like filtering out random noise [111, 219], or inverting a given data transformation (see, e.g., Sec.4.2.3). We assume that inference attacks are more powerful when performed by insider agents, which know all parameter values. Therefore, inference attacks scenarios with different degrees of available knowledge are developed [47, 48, 52]. Our privacy analysis follows this framework to assess the privacy flaws of a given algorithm. Different threats to data privacy in distributed data mining environments are investigated in Chapters 4 and 5, focusing on clustering and time series mining, respectively.

This work has been in part published in [47, 48, 51, 52]

2. **Privacy-Preserving Distributed Mining Algorithms**

Distributed Clustering. This thesis investigates an existing density-based clustering algorithm, the KDEC scheme, to understand how it performed under an inference attack using only information exchanged during a KDEC session. Since KDEC uses kernel density estimation, a

possible attack is based on computing the inverse of the kernel function. The KDEC-S algorithm for distributed data clustering is developed to handle the inverse kernel inference attack (Chapter 4). The main idea is to replace the original kernel function with a kernel approximation at the estimation phase. Therefore, a controlled amount of ambiguity is added to the density estimates. The subsequent results show that KDEC-S provides more privacy than KDEC keeping the same mining quality (cf. Theorem 4.7). Moreover, it is linear in the number of data points in the dataset and has low communication costs. The approach is validated through several experiments using different datasets. This work has been in part published in [46, 47, 52, 53]

Pattern Discovery in Time Series. The thesis proceeds to develop the first privacy-preserving algorithms for frequent pattern discovery in distributed time series: DPD-TS, DPD-HE and DPD-FS (Chapter 5). The main idea is to transform time series in a set of n-dimensional data points and, then, reduce the problem of finding frequent patterns to the problem of finding local maxima in the n-dimensional density landscape. Different approaches to computing the density estimates are investigated. DPD-TS generates contiguous non-overlapping subsequences of the original series. However, it may miss some patterns if they are not aligned with the subsequence size n. DPD-FS generates non-overlapping subsequences but uses a heuristic to prune the number of data points generated. DPD-HE adds a security layer with homomorphic encryption. The privacy properties of all algorithms considering different inference attack scenarios are also investigated. It is shown that the algorithms provides a controlled privacy levels via clear defined privacy parameters (cf. Theorems 5.4 and 5.12). DTSCluster is yet another algorithm spun off the main results of this thesis (also in Chapter 5). We applied the idea of pattern discovery to the problem of clustering short time series from genomic experiments. As it turned out, the approach produced very consistent clusters of short time series with linear time complexity both in dataset size and the number of parties in the mining session. The proposed algorithms are linear in the size of time series and have low communication costs. The approach is validated by experimental evaluation using different datasets. This work has been in part published in [45, 49, 50, 54]

1.4 Thesis Overview

The remainder of this thesis is organized as follows.

Chapter 2 introduces the reader to the basics of knowledge discovery, data mining, distributed data mining algorithms, and systems. This chapter presents an overview of privacy-preserving distributed data mining and discusses its privacy definitions, highlighting its assumptions, applicability, and limitations. Furthermore, this chapter introduces the inference problem, which occurs when unauthorized agents learn sensitive data during a mining session. The inference problem poses a general question as to how data mining can potentially jeopardize privacy since data mining is inherently a learning framework. We propose several general inference attack scenarios and discuss the limitations of current privacy-preserving data mining solutions to handle said inference attacks. [52]

Chapter 3 introduces a set of privacy properties and applies it to the privacy measures discussed in the previous chapter. It shows that existing privacy measures do not satisfy all required properties for privacy measures in distributed environments with inference collusion groups. New privacy measures for distributed data clustering and time series, which address inference and collusion, are also introduced. The new measures are then applied to some representative distributed privacy-preserving algorithms.

Chapter 4 presents the privacy-preserving data mining problem reformulated for the particular case of distributed data clustering. We analyze a distributed clustering algorithm, KDEC, concerning its privacy-preserving properties through attack scenarios and propose a new algorithm, KDEC-S, which improves the privacy-preserving aspects without compromising mining quality.

Chapter 5 presents the general problem applied to the case of distributed time series mining. We formulate the problem of privacy-preserving pattern discovery in time series and propose three algorithms to solve it, DPD-TS, DPD-FS, and DPD-HE, together with an analysis of their privacy and performance properties. This chapter also introduces the

DTSCluster algorithm, which is applied to genomic short time series data clustering.

Chapter 6 discusses the architecture and implementation of the proposed algorithms. It also presents an evaluation environment in which the experiments in this thesis were run. A description of all datasets used in this thesis is also provided.

Chapter 7 presents the main conclusions and discusses future research directions.

Chapter 2

Background

> "Advanced analysis of data for extracting useful knowledge is the next natural step in the world of ubiquitous computing."
> (H. Kargupta in *ACM SIGKDD Tutorial, 2001*)

This chapter introduces fundamental notions of distributed data mining and privacy issues. Readers familiar with knowledge discovery in databases (KDD), data mining algorithms (DM), distributed data mining (DDM), and data mining systems, may skip all or parts of this chapter. It also introduces privacy-preserving distributed data mining (PPDDM), its central concepts, problems, approaches, and limitations. It also introduces the main threats to privacy in a distributed data setting: inference attacks and collusion groups.

2.1 Data Mining

Data mining and knowledge discovery are commonly used to denote finding new knowledge from large databases. In this work, however, a clear distinction is made between the two terms, as discussed in the following sections.

Knowledge Discovery Process Knowledge discovery in databases (KDD) is a process that aims to automatically find new pieces of useful information from vast amounts of data [93]. Its primary motivation is to automatically discover patterns and relations between data stored in large datasets that could not feasibly be discovered manually. The process is essentially an attempt to cope with the massive amount of information that one must manage, developing complex analysis tools to reach this objective. In KDD,

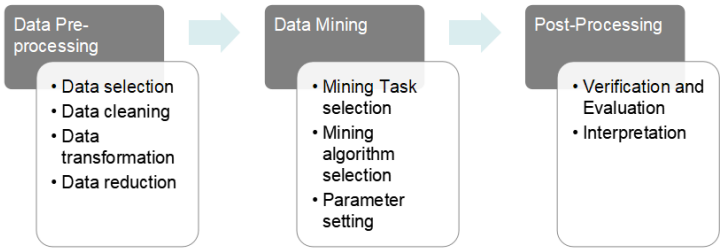


Figure 2.1: *KDD phases showing its sub-steps.*

the term "huge" usually denotes Terabytes of information. Examples of such gigantic databases are satellite image databases, medical information repositories, market information databases, social network data, to name a few.

KDD is an interdisciplinary field and can be seen as the intersection of the research in machine learning, statistics, information theory, and databases [238]. Today, the term KDD is used as a synonym for data mining (DM), and, indeed, one can find many research papers that use the term DM to represent the entire KDD process. Kargupta and Chan remark, on a "nostalgic note", that the term KDD was used even before the term DM was coined [110]. Throughout this thesis, the term *data mining* will be referred to as one of the phases of the KDD process.

The KDD process consists of four main phases [93]: *problem definition, data preprocessing, data mining*, and *data post-processing*. These phases can be further decomposed into sub-phases, as shown in Figure 2.1 (cf. [72, 238, 242] and [94]).

Data Mining Phase The previous section shows that data mining (DM) is just one of several steps in the KDD workflow. This section presents some of the main issues addressed by DM.

DM is commonly regarded as data analysis of large datasets. Data analysis shares with DM the goal of searching patterns in data, and it has been used for a reasonable amount of time by statisticians, economists, biologists, meteorologists, to name just a few. DM, however, is related to the *automated search* in massive datasets, where data is electronically stored and usually takes the form of a set of examples.

Data mining has two high level goals[1] [72]: *prediction* and *description* (cf.

[1] This classification is object of discussion in the data mining community.

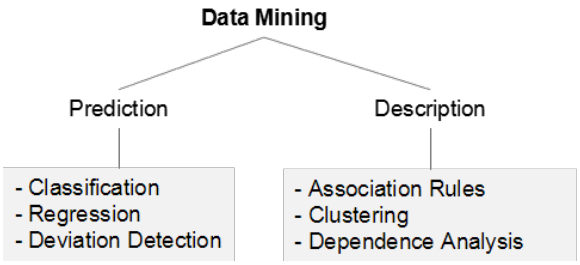


Figure 2.2: *Data mining tasks.*

Fig. 2.2).

Prediction The main goal of prediction tasks is to build models (and patterns) that can predict unknown values of a target attribute from the known predictor attributes. Examples of prediction tasks are classification, regression, and deviation detection.

Description The main goal of description tasks is to summarize the data that can be used to recognize, for example, how the data is distributed, whether or not the attributes are correlated, and so forth. Examples of description tasks include data clustering, association rules, and dependence analysis.

The pieces of information discovered by DM algorithms are represented by any well-defined mathematical structure, two of the most well-known of which are decision trees and association rules. A variety of DM techniques have been developed in the last decades, including cluster analysis [3], decision-tree based classification [119], and mining of association rules [73, 141]. The reader may refer to [174] for introductory material on data mining tasks and algorithms.

DM also involves computational issues. For example, in DM, one has to address the scalability of the pattern searching algorithms, data distribution issues, and memory spaces utilization. The early research addressed these issues and produced several algorithms for different mining tasks.

One critical dimension of DM pertains to how exactly data is stored, and it can be stored in a single central repository (e.g., a data warehouse) or distributed among different locations. Distribution implies a set of new challenges to DM, which are discussed in the following section.

2.2 Distributed Data Mining

One of the most widely used approaches to mining data from distributed sources is to apply traditional DM techniques to a collection of integrated data from the sources in a central repository [76]. This approach presents several limitations, such as network bandwidth limitation and poor algorithm scalability.

Most datasets in a distributed setting are too large to be transferred. For instance, health care clinical data [177, 180], smart grids data generated from smart meters [223],distributed genomic studies [28, 90], are examples of huge datasets generated at different locations. Moreover, downloading data to a central site is impractical when the network is based on a low bandwidth connection. The downloading of sensitive data may also not be allowed due to the local security policies of any given data owner. The centralized approach with classical DM algorithms does not scale well on the size of datasets or the number of different data sources since they need to access raw data, requiring many rounds of messages among the sites. For all these reasons, the centralized approach typically represents the main bottleneck for mining distributed databases. Distributed Data Mining (DDM) addresses these limitations, investigating how to enable data mining with extremely large distributed data sources with limited network bandwidth.

Another challenge posed by a distributed environment is how to handle the heterogeneity of data. Heterogeneity comes into play in two different circumstances [76]: (i) when data is represented in different ways on different sites; (ii) when data is split in a non-trivial way among sites. Different data representations and distribution possibilities are presented in the following.

There are three main data representation possibilities according to the data structure (or lack of it):

- *Structured data*: In this category, data is represented by a rigid, regular schema, such as a relational database.
- *Semi-structured data*: In this category, data follows no rigid schema. Moreover, it can have an irregular and implicit structure. Commonly, the distinction between schema and data in semi-structured data is blurred. Examples include XML, BibTex, and JSON data format.
- *Unstructured data*: In this category, data has no structure at all, e.g.,

	f1	f2	f3
X1	2,3	1,6	0,2
X2	5,2	7,8	0,3
...			

Site 1

	f1	f2	f3
X3	4,5	5,3	0,9
X4	5,1	3,2	0,7
...			

Site 2

Figure 2.3: *Homogeneous data distribution: each site stores the* same set of features *(e.g., f_1, f_2, f_3), but has a different set of data objects. In the example, Site 1 stores objects x_1, x_2 while Site 2 stores objects x_3, x_4.*

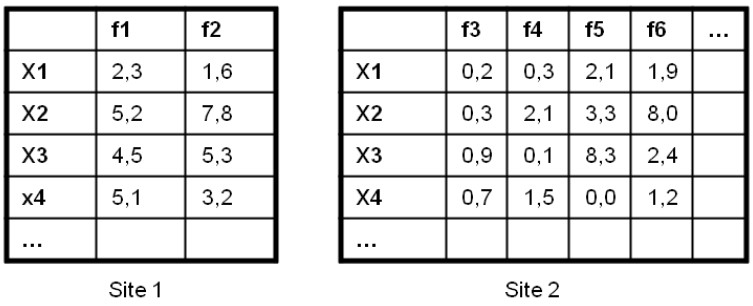
plain text, images, sound, genomic data.

With regards to data distribution, data may be spread through distributed datasets in at least two different ways:

- *Homogeneous data distribution*: In this case, data objects are distributed across the sites in such a manner that each location stores the same features of the data objects. It can be said that the data objects are grouped by site (cf. Fig. 2.3).
- *Heterogeneous data distribution*: Each site stores a different set of attributes for each data object, possibly with one or more features in common among the sites, meaning that each location has only partial information concerning the data object distributed over the local sites (cf. Fig. 2.4).

The terminology *horizontal* and *vertical* is often used in the literature to describe a particular case of homogeneous and heterogeneous distribution, respectively, when data is structured and stored in just one table [76].

A general scheme to solve distributed data mining (DDM) problems is to perform data mining algorithms at each site to build partial local models of datasets stored at each location, e.g., [212, 240]. These partial models are then combined to create a global model. Some data samples may be



	f1	f2
X1	2,3	1,6
X2	5,2	7,8
X3	4,5	5,3
x4	5,1	3,2
...		

Site 1

	f3	f4	f5	f6	...
X1	0,2	0,3	2,1	1,9	
X2	0,3	2,1	3,3	8,0	
X3	0,9	0,1	8,3	2,4	
X4	0,7	1,5	0,0	1,2	
...					

Site 2

Figure 2.4: *Heterogeneous data distribution: each site refers to the same data objects (e.g., $x_1, x_2, x_3, x_4, \ldots$), but stores a partial view (*different set of features*) of them. In the example, Site 1 stores feature f_1 and f_2 while Site 2 stores features $f_3, f_4, f_5, f_6 \ldots$.*

exchanged among the sites to improve the global model. The DDM approach presents better scalability and less communication overhead than the centralized approach.

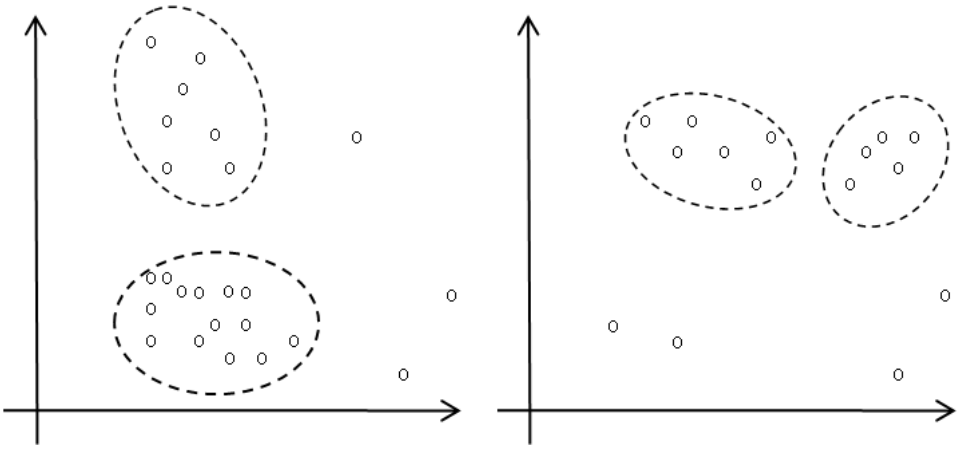
The following sections overview two DDM tasks and discuss their main challenges, followed by strategies to solve them. A survey on the field of distributed data mining can be found in [76, 214].

2.2.1 Distributed Data Clustering

The distributed data clustering (DDC) problem is informally stated as the following: *to find a partition over distributed data such that every data point in local datasets is assigned to a global cluster.* Global clusters are only discovered when all datasets are considered.

Example 2.1 *Figure 2.5 shows two sites and their respective local cluster map. There are three clusters if we combine the datasets, as shown in Fig. 2.6.*

Distributed clustering algorithms can be classified into two subcategories. The first consists of methods requiring multiple rounds of message passing, which requires a significant amount of synchronization [9, 185]. The second subcategory consists of methods that build local clustering models and transmit them to a central site (asynchronously) [126]. The central site forms a combined global model. These methods require only a single round of message passing, thus resulting in modest synchronization requirements.



(a) Site 1 (b) Site 2

Figure 2.5: *A distributed data clustering scenario. Local data clustering may not find all clusters.*

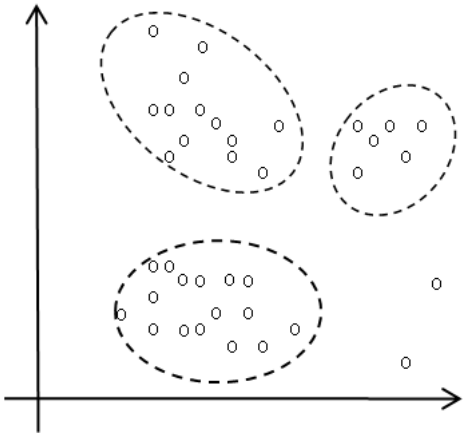


Figure 2.6: *Global clusters*

When it comes to network architecture, there are many forms in which peers may get organized: (i) distributed network with central node; and (ii) distributed network without central node [9]. The peer-to-peer model is an example of the first architecture, and MapReduce is an example of the second model.

MapReduce [57] is a framework defining a shared-nothing memory architecture implemented over a network of several loosely coupled machines. Nonetheless, it is more restrictive than peer-to-peer, requiring all the machines to be under the same trust domain. In MapReduce, one node act as coordinator or master, and all the other node act as workers or slaves.

Several MapReduce implementations of clustering algorithms have been proposed. Yu et al. [237] proposes Cludoop, an efficient and load-balanced distributed density-based clustering for big data on the Hadoop platform. The distributed algorithm incorporates a proposed serial clustering with c-cluster as a plug-in on mapper and a Merging-Refinement-Merging 3-step framework to merge c-cluster on the reducer. Experiments on large-scale real-world and synthetic data show that Cludoop exhibits better scalability and efficiency when compared with its predecessors. Tsapano et al. [210] improve the performance of Kernel k-Means using a kernel matrix-trimming algorithm. All small entries in the kernel matrix are discarded to reduce the kernel matrix size, and the resulting sparse matrix is stored as adjacency lists instead of the entire matrix. The approach follows the MapReduce programming model and consists of 3 stages: the kernel matrix computation, kernel matrix trimming method, and the Kernel k-Means clustering algorithm. Results showed that the proposed approach performs just as the Kernel k-means with the whole matrix. Heidari and colleagues developed a version of MR-VDBSCAN in MapReduce to address the problem of varied densities in a dataset [100]. It is based on DBSCAN, but MR-VDBSCAN computed a local density list to help identify points located on different levels of densities.

Peer-to-peer architecture does not rely on a central node but requires more coordination among all peers. Altilio et al. [9] addressed the problem of clustering data distributed in a peer-to-peer network. They developed a consensus-based expectation-maximization algorithm (CEM). Computation is split among the parties, where local EM produces a local temporary solution. A consensus protocol is run at each step to define the global EM

parameters. It is an iterative algorithm running until a convergence criterion is reached. A similar approach is explored by Qin et al. [170] who propose a consensus algorithm applied to cluster distributed data from a wireless sensors network. The base model is a multiclass logistic regression model built iteratively. The first partition is generated by k-means in the first round to ensure the algorithm's convergence. The authors point out that the algorithm can find the appropriate number of clusters. On the other hand, the proposed approach is limited to linear boundaries between clusters. Rosato et al. [175] proposes the V-DEC algorithm for a decentralized architecture. In this approach, each peer generates a local cluster and collaborates to find k-means centroids.

2.2.2 Distributed Time Series Mining

Distributed time series analysis and mining encompass a wide variety of problems and applications, such as smart meter applications [222], collaborative forecasting among different companies [86, 87] public health and clinical research or participatory sensing applications discussed in [172]. In this setting, participants contribute various time-series data to get useful information such as road congestion patterns, micro-weather, load profiles. We discuss a few examples of research directions on distributed time series in the following.

Forecasting is one classical problem in time series. Galicia et al. developed a scalable approach for multi-step forecasting [75]. The problem is to predict m steps in the future, given past w values in the time series. The original time series was split into m subseries and trained in parallel to build an ensemble of m models to predict forecasts with different time horizons. The approach was presented as a centralized method but could be implemented as a distributed scheme. Talavera et al. [195] also studied the m step forecasting problem. They use parallel computing to find k nearest time series from a distributed dataset previously split among several nodes. The average value of the most similar time series is used as the forecast. We refer the reader to [164] for an overview of distributed forecasting approaches.

Baldán and Benítez proposed a distributed time series **classification** algorithm based on MapReduce model [18]. The time series are split between nodes to distribute the processing load. A time series is transformed via SAX to a discrete version at each node, and a set of best shaplets is built collabo-

ratively with information from all nodes. Then, the minimum distance from each time series to each shaplets is computed, and this information is used to train a random forest model.

Gong et al. [83] studied **clustering** of distributed time series and proposed the DBPEC algorithm. DBPEC first split the original time series into several partitions. Using the Apache Spark framework, DBPEC computes the centroids in parallel and finally produces a global cluster mapping aggregating information from each partition. Corizzo et al. introduced DENCAST [44]. DENCAST is implemented in Apache Spark and splits the time series among several nodes. The algorithm uses a distributed local sensitive hash (LHS) to find an approximate solution that clusters the time series. DENCAST uses the discovered cluster mapping to make predictions using the average values from time series in a given cluster.

Indexing is a core function for many data mining tasks, like clustering, motif discovery, and classification. Yagoubi et al. [231] proposed DPiSAX, a parallel solution to index and query billions of time series. The authors implemented with Apache Spark framework. The DPiSAX splits the time series datasets into partitions where local indexes are created and combined into a global index. The solution includes both approximate and exact searches.

2.2.3 Distributed Data Mining Systems

In this section, a couple of DDM systems that have been successfully applied to real-world cases are reviewed. The main features of these systems – the mining task, data mining algorithms used, network type, and data distribution – are all given an overview. Table 2.1 is provided to summarize the systems reviewed.

Kensington system architecture was developed by Chattratichat et al. [38]. Kensington uses several Enterprise Java Beans to provide data mining, object management, and storage management for local and remote clients. The client is implemented as Java Beans and can connect to the server through the RMI protocol. The system can operate in a TCP/IP Internet, Intranet, or virtual private network. Additionally, the system incorporates connection management and secure communication through secure sockets (SSL). Each node in the system communicates mining results, database schema, data samples, and even full datasets on demand. Kensington was demonstrated at the Terabyte Challenge '98, performing data mining in a

distributed setting, including a Kensington server in London, another in Chicago, and a client in Orlando.

Papyrus [89] is a Java-based system addressing wide-area DDM over clusters of heterogeneous data sites and meta-clusters, supporting different task and predictive models, including C4.5. It uses mobile agents to move data, intermediate results, and models between clusters. All computation is performed locally to reduce network load, and a central root produces the final result. Each cluster has one distinguished node, which acts as its cluster's access and control point for the agents. The coordination of the overall clustering task is either carried out by a central root site or distributed. Papyrus describes the models and metadata by using a particular markup language. The authors do not address privacy in Papyrus.

PADMA was developed by Kargupta et al. [112] and deals with the problem of DDM with homogeneous data sites. Partial data cluster models are first computed by stationary agents locally at different locations. All local models are collected at a central site that performs a second-level clustering algorithm to generate the global cluster model. That is followed by individual agents carrying out hierarchical clustering in text document classification and web-based information visualization. Partial concept graphs are subsequently exchanged among agents. It is worth noting that raw data is also exchanged to perform parallel join operations. PADMA uses Parallel Portable File System (PPFS) as its core infrastructure, developed in C++. No privacy issues are discussed in PADMA.

BODHI was developed by Kargupta and his colleagues based on the Collective Data Mining (CDM) framework [113]. This system was developed in Java, uses mobile agents, and intends to be a communication system and runtime environment used in collective data mining. It is not bound to any specific platform, learning algorithm, or knowledge representation. The messages are in KQML with embedded data or commands, and the security is based on RSA.

JAM was proposed by Prodromidis, Chan and Stolfo [192, 168] and is an agent-based meta-learning system for large-scale data mining applications. JAM performs high-level classifiers called meta-classifiers and can operate in heterogeneous data. The system exchanges mining results as classifiers agents sent as serializable Java objects. The authors do not, however, describe the privacy features.

EMADS (Extendible Multi-Agent Data Mining System) [6] is a hybrid peer-to-peer agent-based system for distributed data mining. It was implemented in JADE and includes data agents, mining agents, task agents, user agents, and JADE-specific agents for mediation and coordination. EMADS considers three mining tasks: classification, clustering, and association rules mining, though it is not bound to any specific algorithm. Wrappers provide extensibility, adapting data sources or existing mining systems to operate as an EMADS agent. Agents communicate partial models and control information. Privacy is not particularly well described in EMADS. However, only local mining agents have access to local datasets, never revealing sensitive data to non-local mining agents.

JaCa-DDM (Jason and CArtAgO for DDM) follows an agent and artifacts approach [127]. It defines complex distributed data mining workflows as agents interactions and data mining algorithms as artifacts. New algorithms can be added to JaCa-DDM via artifacts, and workflows are defined as Jason agent programs. Agents in JaCA-DDM are full belive-desire-intention (BDI). Communication is based on KQML speech acts. JaCA-DDM was implemented in Java (Jason agents and Weka artifacts). Concerning privacy and security, the authors mention it as a feature in JaCa-DDM, but it is not explored in detail.

2.3 Privacy Issues in Distributed Data Mining

Protecting data privacy in today's era of digital information processing is a challenging task. Data breaches are frequently happening as intentional attacks, not accidental disclosures. Yahoo was attacked in 2014, LinkedIn in 2012 and 2021, Adobe in 2013, to name a few. The attackers look for credit card information, user ids, emails, and passwords. Typically, the stolen information is sold later on the dark web [155]. Nevertheless, there are other threats to privacy beyond data disclosure.

There have been increasing concerns that data mining is a potential threat to privacy [187]. A well-known data privacy scandal involved Facebook and Cambridge Analytica in 2018 [98]. Cambridge Analytica used data from Facebook users to produce political profiles during the presidential run in the United States. In this case, sensitive data was not stolen, released to the public, or sold by hackers. Nevertheless, it raised ethical and political

	Papyrus	**Kensignton**	**PADMA**	**BODHI**	**JAM**	**EMADS**	**JaCA-DDM**
Reference	[89]	[38]	[112]	[113]	[192, 168]	[6]	[127]
Language	Java	Java (EJB)	Java (GUI) and C++ (core)	Java	Java with support to native code	Java (JADE)	Java
Network type	WAN of clusters and meta-clusters	LAN / CORBA over TCP/IP	Cluster of nodes	TCP/IP based network	LAN	Peer-to-peer	TCP/IP
Framework	Distributed Data Clustering	Knowledge Probing	Distributed Data Clustering	Collective Data Mining	Meta-learning	Distributed Data Mining	Distributed Data Mining
Tasks Supported	Text clustering	Classification	Clustering	Any DM task	Classification	Association Rules and Classification	Any DM task
DDM Algorithms	C4.5	C4.5, CN2, Naive Bayes, Neural Nets, Apriori	Hierarchical Clustering	Not bound to a specific algorithm. Extensible to any DM algorithm	Any classifier including Ripper, CART, ID3, C4.5, and Bayes	Extensible to any DM algorithm	Any DDM algorithm
Data distribution	Heterogeneous	Homogeneous	Homogeneous	Heterogeneous	Heterogeneous	Any	Homogeneous
Information transferred	Data, intermediate results, and models	Mining Results, DB schema, samples, and full datasets	Partial Models (concept graphs) and raw data	KQML Messages with embedded data or commands	Mining results (learners agents)	Partial models and control information	Results, logs, partial models
Mining Description Language	A special markup language is used	PMML	N/A	Not bounded to specific language	N/A	N/A	N/A
Agents	Mobile	N/A	Stationary	Mobile	Mobile (learners/classifiers)	Stationary	N/A
Coordination	Central Root or Distributed	Task Engine EJB	Facilitator (Module)	Facilitator (Agent)	JAM Engine	Task Agent	ag1 (contact person)
Data Privacy	not discussed	Secure Sockets in RMI (SSL)	Not discussed	RSA encryption	Not discussed	Data is handled by local mining agents only	Not discussed

Note: N/A stands for *not addressed*

Table 2.1: *Comparison of DDM Systems.*

concerns about how to use and distribute sensitive data. In 2011, a similar debate emerged around the issue of mining drug prescription data as some deemed it a violation of privacy [226]. While privacy advocates fear the consequences of privacy breaches, such as civil rights infringement, the industry wants to keep profiting from the knowledge discovered from as many data sources as possible.

The privacy breach is evident in many cases – personal information was disclosed. However, generally speaking, what is a privacy breach? Moreover, what if just a summary of data was released? What if only data mining results were made public? Does it threaten privacy? These are increasingly pertinent questions. One of the main difficulties involved in this debate pertains to the very definition of privacy itself and how to guarantee it is being protected in the context of data mining.

2.3.1 Privacy, Confidentiality and Sensitive Data

Privacy is an elusive concept; it provokes many interpretations and, as such, there is plenty of literature attempting to shed further light on its complexities [65, 147, 182, 204, 206, 216, 236]. After the *Universal Declaration of Human Rights* defined privacy as a right [199, Art. 12] it is well-accepted that it must be protected.

The term "privacy" itself prompts the intuitive response that it pertains to a particular individual. Therefore, all information about a person is subject to the notion of privacy. As a consequence, only an individual has the right to determine how her or his data should or should not be used [20]. Of course, the question of what is to be considered private can be interpreted differently depending on one's culture and legislation, but it is the individual who should have the ultimate decision on the matter [1].

Well-known privacy regulations define any piece of data that is related to a natural individual and which may be used to identify said individual as *sensitive* information. In the USA, there is no federal law covering all aspects of data privacy. There are a large number of laws regulating different topics and sectors [163]. The Health Insurance Portability and Accountability Act (HIPAA) establishes regulations for the use and disclosure of Protected Health Information (PHI). HIPAA defines PHI as any information concerning health status, provision of healthcare, or payment for health care that

can be linked to an individual[2]. The disclosure of personal information is also regulated by many different federal statutes, such as the Fair Credit Reporting Act of 1970 (credit records), the Video Privacy Protection Act of 1988 (video rental records), the Family Education Rights and Privacy Act of 1974 (educational records) and the Employee Polygraph Protection Act of 1988 (employee polygraph records). California went further and passed the California Consumers Privacy Act (CCPA), 2018. For instance, CCPA provides California residents the right to know what is collected about them, refuse to sell personal data, or request a company to delete information about them, among other rights. CCPA defines responsibilities, sanctions, and remedies.

In the European Union, privacy is regulated by the General Data Protection Regulation (GDPR), enforced by each EU country since May, 2018[3]. In contrast to regulation in the USA, GDPR applies to all business sectors and defines how individuals are to be protected when data is processed and transferred. GDPR requires businesses to communicate their user agreement in simple terms and to allow users to have their data removed when requested [232]. Companies violating the regulation will face severe sanctions.

China [163] created a privacy regulation in 2018 "that is stricter than the US but not as much as the EU" [163]. In Brazil the National Congress passed the *Lei Geral de Proteção de Dados Pessoais* (LGPD) in August, 2018[4]. Brazil's LGPD substitutes previous fragmented legislation about data privacy and is, in many respects, similar to the EU's GDPR.

According to all privacy regulations, any piece of data concerning a natural individual is to be considered private and cannot be made available for any other purpose beyond the one for which it was first collected. The fundamental idea behind every regulation is that any information that could identify a person in the real world needs to be protected from unauthorized access.

One related aspect of privacy is the preservation of *sensitive knowledge*, i.e. sensible mining results [147]. In this thesis, the term *knowledge* essentially denotes models and patterns, i.e., typical mining results, that could

[2]The complete suite of HIPAA Administrative Simplification Regulations can be found at 45 Code of Federal Regulations (C.F.R.), Parts 160, 162, and 164.

[3]Regulation (EU) 2016/679 of the European Parliament and of the Council of 27 April 2016. EU countries had to transpose it into their national law by 6 May 2018.

[4]Lei nº 13.709, August 2018. Accessible from `http://www.planalto.gov.br/ccivil_03/_ato2015-2018/2018/lei/l13709.htm`

reveal sensitive information (such as individual buying habits, a particular health condition, or even about a company's financial situation).

From a more security-minded point of view, privacy is closely related to the concept of confidentiality. Ensuring that only authorized agents have access to sensitive data is one of the objectives of secure systems. Data may fall under many different levels of confidentiality, such as *public, restricted, secret, top-secret*. Privacy is one such case of confidentiality where sensitive data pertains to a person. Every individual has the right to define a desired level of privacy to its data, and other agents in the system should comply. Sensitive data, through this thesis, is referred to as a general term, denoting any piece of data that an agent wants to keep secret from any other agents to any desired level. If it is not meant to be public, it is sensitive.

Independent of the domain, sensitive data can be roughly categorized into two main classes in data mining: (i) sensitive inputs; and (ii) sensitive outputs. Sensitive inputs are the starting point in the data analysis procedure. Sensitive outputs may consist of statistical summaries, classifiers, cluster mappings, regression models, and the like. The outputs themselves may represent a valuable asset or compromise input privacy if used to infer sensitive input data.

The same idea applies to business data in a business-to-business scenario. In some scenarios, information about the data collector is, in itself, sensitive. Consider, for example, a survey on the rate of deaths in hospitals; it is likely that hospitals would want to keep their identity secret if they are to partake in the study.

Following the ideas discussed thus far in this chapter, we, rather informally, define:

Sensitive data is a piece of data to which a privacy level is assigned for a given privacy measure. An example of sensitive data is information about individuals in the context of medical data.

Privacy is the right that an agent has to keep any given piece of information hidden from other agents.

Privacy measure is a quantitative indication of how hidden a piece of data is from being accessed by unauthorized agents.

The following section discusses privacy threats to datasets and privacy

threats in the context of data mining. Later in this chapter, we return to the topic of privacy formalization (cf. Section 2.3.3), and in Chapter 3 we introduce new privacy measures for distributed data clustering and time series mining.

2.3.2 The Inference Problem

Several organizations, like retail stores, hospitals, clinics, census bureaus, collect and maintain large collections of personal information, for instance, purchase transactions, medical records, and census data [236]. These data collections are valuable for research, marketing, fraud detection, to name a few. However, as discussed in the previous section, data collectors in many fields (medicine, for example) cannot publicize data about individuals due to privacy regulations. In a business-to-business context, allowing a third party to access a dataset may disclose valuable information that competitors could use as a business advantage.

The main concern is not direct disclosure of sensitive data, which is addressed by access control mechanisms such as mandatory access control (MAC), but rather the threat of indirect disclosures based on inferences that can be drawn from queries against the database or released data [194]. This problem is known as *the inference problem* and appeared first in the literature on Statistical Databases in the mid-1970s [201]. It would now be helpful to examine various well-known instances of this problem in different contexts, all of which involve sensitive data.

Inference Problem in Databases

Access to statistics about groups is permitted in a statistical database (SD), but the data concerning individual entities is not released to preserve confidentiality. However, an attacker might disclose confidential information about an individual entity by posing queries on aggregate statistics and performing arithmetic operations on the answers received, using information about the size and nature of the sets of individuals involved, as shown in Example 2.2. Inference control in a statistical database has been extensively studied, and several *inference control mechanisms* were developed, including cell suppression, table restriction, and record perturbation [58, 70].

Example 2.2 (from [58]) *Suppose there is only one female professor in a*

particular department. If statistics reveal the total salary of all professors in the department and the total salary of all the male professors, the female professor's salary can be obtained by subtraction.

A multilevel secure database (MLSDB) is a system in which every user and each piece of data has a security classification label. Classification labels form a mathematical lattice structure defining a partial order among the labels [30]. MLSDB ensures that data at a security classification above the user's level will be invisible to that particular user [152]. An inference attack in a multilevel secure database allows an attacker to use low classified data to infer data classified at higher levels. As illustrated in the following example, a significant class of inference attacks is based on meta-data such as integrity constraints, functional, multivalued, or join dependencies.

Example 2.3 (adapted from [30]) *Assume security classification labels* `public` $<$ `confidential` $<$ `top-secret`*, where the relation* $<$ *induces a partial order among the labels, with* $x < y$ *meaning* x *is less secret than* y*. Let a table* T*, holding information on employees, be defined as* $T(name, rank, salary, experience)$*. Assume that tuples containing both name and salary are classified at top-secret level and that tuples with name and rank are classified as confidential. Further, consider that salary is determined by the employee's rank, i.e.* $rank \rightarrow salary$*. An attacker with clearance to handle only confidential data but not allowed to access data at higher levels, e.g. top-secret, may issue two queries listing* $R_1(name, rank)$ *and* $R_2(rank, salary)$*, both at confidential level. However, with* R_1 *and* R_2 *a user may use the fact that rank determine salary and disclose the relation* $R_3(name, salary)$*, which is at top-secret level.*

Several approaches to handling inference in MLSDB have been proposed, including constraint-based security, conceptual structures, and logic-based approaches. For a survey on MLSD, we refer the reader to [201].

The inference problem is not limited to statistical or multi-level databases. Research concerning general-purpose databases has been a topic of investigation that has produced several significant contributions. The main goal is to protect sensitive information from indirect disclosure, but there are no explicit classification labels to guide a disclosure control mechanism.

Example 2.4 (adapted from [194]) *Consider a table defined by the re-*

lation T(physician, patient, medication). A query listing physicians and patients, i.e., the relation $R_1(physician, patient)$, may not be sensitive. Similarly, a query on medications prescribed by each physician may also be non-sensitive. However, consider a query associating patients with their prescribed medications, i.e., $R_2(patient, medication)$. This query may be sensitive since medications typically correlate with diseases. Although there is no direct disclosure of any relation like $R_3(patient, disease)$, an attacker may use public data $T_{pub}(medication, disease)$ to infer that a given patient may suffer from a given disease.

Approaches to inference in databases typically focus on issues like minimal classification updating, partial disclosure, classifying existing data repositories and how to prevent inferences via knowledge discovery [23, 31, 55, 103, 157, 202]. For a survey on this area, the reader can refer to [66].

Inference Problem in Data Mining

The inference problem has also been investigated by the data mining community (e.g., [14, 99, 121, 144, 187, 200]). Classical data mining algorithms typically require access to raw data and, if the miner is a malicious agent, it may use the queries' answers to infer sensitive information. Countermeasures to this problem are already discussed in previous paragraphs in the context of databases. However, data mining opens up new inference possibilities. With data mining techniques, the inference problem has become even worse. According to Thuraisingham [200], privacy threats in data mining can be viewed as a variation of the inference problem in databases:

> "[The inference problem] has been discussed a lot over the past two decades. However, data mining makes this problem worse. Users now have sophisticated tools that they can use to get data and deduce patterns that could be sensitive. Without these data mining tools, users would have to be fairly sophisticated in their reasoning to deduce information from posing queries to the databases. That is, data mining tools make the inference problem quite dangerous. (...) we are beginning to see many parallels between the inference problem and what we now call the privacy problem." – *B. Thuraisingham*

Data mining results inherently elicit information regarding the data collection [14, 187]. Cluster maps, for instance, describe the overall distribution of data points in a dataset. Similarly, association rules (AR) reveal how items co-occur in a shopping basket; a rule about drugs used to treat a given disease, for example, may disclose sensitive health information about individuals. Classifiers also offer potential privacy threats. To illustrate the problem, imagine that a health insurance company builds a classifier allowing it to identify people with an HIV infection. In this case, if HIV-positive individuals attempt to attain life insurance from this company, they will have their health condition disclosed by the classifier. Therefore, data mining results may represent a threat to privacy.

The above discussion implicitly assumes that we have a centralized setting with a central dataset being queried by a data miner. To summarize, we state that in a centralized data mining setting, the inference problem may pose the following threats:

Threat 1 A malicious miner agent could try to access sensitive information from a set of queries to the central dataset (or a privacy-preserving version of them).

Threat 2 A malicious agent may learn sensitive mining results (patterns or models), which can be used to infer either the identity or sensitive data stored in the central dataset.

Inference and Collusion in Distributed Data Mining

In a distributed data mining setting, data is spread across different sites, each of which represents a different party (such as an institution, a company, or a clinic). Each party owns a local dataset and is unwilling to disclose it to other parties. In a distributed data mining algorithm, the previous threats are still present. Further, intermediate results, partial models, and pieces of sensitive knowledge (or model) may be linked to a specific party. Therefore, we rewrite the previous threats as follows:

Threat 3 A malicious miner agent may try to infer sensitive data or identity from queries to datasets (or a privacy-preserving version of it), intermediate mining results, or messages exchanged during the mining process;

Threat 4 A malicious miner agent may learn sensitive mining results (patterns or models), which can be used to infer either the identity or sensitive data stored in datasets owned by other agents.

Threat 5 A malicious miner agent may learn which party produced a given partial model or local mining result. A partial model could reveal sensitive information about a specific party.

Inference and Collusion Scenarios in Distributed Data Mining

This section endeavors to define a general framework that describes inference attacks due to the distributed data mining process. The following section outlines definitions and assumptions used throughout this thesis.

In this work, the term *agent* is assumed to denote a generic piece of software able to: (i) communicate with other agents, (ii) perceive its environment, and (iii) carry out a set of activities on behalf of the user it represents [178]. A distributed data mining algorithm is defined as a distributed process taking a distributed dataset as input and giving a mining model or pattern as output. A distributed mining algorithm is started by a finite group of agents organized, according to the algorithm at hand, into a temporary coalition called a *mining group*.

Two roles, in particular, are considered; the *data holder* (or *data collector*) and the *data miner*, which are played by agents in the system. Data holders are in charge of the datasets, which contain sensitive information about individuals, trade secrets, or business strategies. Data miners start and coordinate mining sessions on behalf of their users. In any given mining group, each agent may act at the same time as *data miner* and *data holder* as well.

A typical distributed data mining session is illustrated in Figure 2.7. Starting with the original dataset D, data holders produce intermediate results I from the original datasets D, which can be local models or statistical information about D. Then, data miners collaboratively produce and publicize the mining results M to all parties in the mining group.

A *malicious agent* is an agent that uses information exchanged among the participants of the mining group to feed a secret reconstruction process, aiming to disclose sensitive data points from other agents' datasets in the mining group.

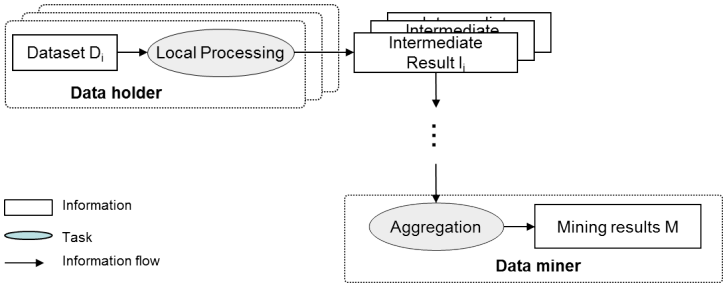


Figure 2.7: *Distributed data mining scenario. Each data holder produces a local model (or any intermediate result) of its datasets, from which a miner agent produces the final mining result.*

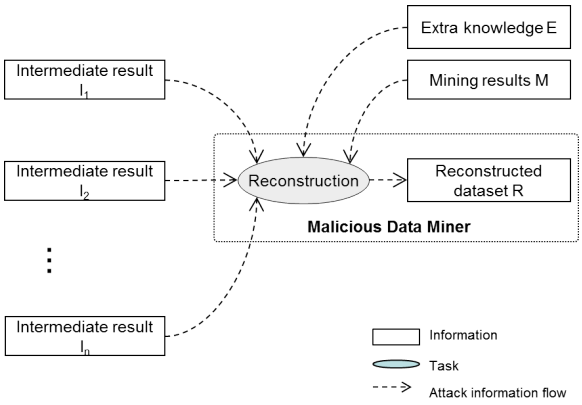
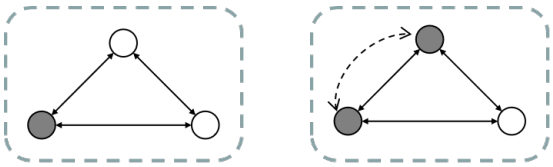


Figure 2.8: *Inference attack in a distributed data mining scenario. Possible inference attacks occur under the control of a malicious miner agent without being detected by the data holders.*

Furthermore, we define an *insider agent*, given a mining group $\mathcal{L}$, as an agent which takes part in the activities of this group. Conversely, an *outsider agent* is an agent that is not part of a given mining group. Note that the terms *insider* and *outsider* are relative to a specific mining group.

In this thesis, we refer to the **inference problem** as the threat to data privacy posed by inference attacks, as stated in the following definition.

Definition 2.1 (Inference Attack) *An inference attack in a distributed data mining setting is a reconstruction algorithm executed in the background by a malicious miner agent beyond the steps dictated by the mining protocol (cf. Figure 2.8) using all information available to him (including partial mining results, intermediate computations, parameter values, and extra knowledge) to reconstruct sensitive information residing on the data space of*



(a) Single peer attack (b) Collusion attack.

Figure 2.9: *Insider attack scenarios. (a) Single peer attack: one or more peers try to learn sensitive information from the other mining peers, but attackers work alone, i.e., independent from each other. (b) Collusion attack: attackers work together, exchanging extra messages (dashed line) to improve the confidence of the disclosed sensitive data.*

other peers in a mining group. □

Inference attacks in distributed settings may represent threats 3, 4, and 5 discussed earlier in this section. In other words, inference attacks could breach the privacy of sensitive information even when the data holder does not directly disclose sensitive data to miner agents.

An inference attacks can be further separated the following categories: *insider* (possibly with collusion), *outsider* and *mixed* attacks.

Insider attack scenario. In this scenario, one member of the mining group performs the attack. This scenario can be further separated into two sub-scenarios: (a) single peer attack and (b) collusion attack. In the single peer attack scenario (cf. Fig. 2.9(a)), one or more peer tries to reconstruct sensitive information from the other mining peers, but each attacker works alone without contacting any other attackers. Conversely, in the collusion attack scenario (cf. Fig. 2.9(b)), there are messages exchanged between (a subset of) attackers. In this scenario, we assume that the attacker knows the values of the parameters used in the mining session.

Outsider attack scenario. In this scenario, attacks are performed by peers that are not members of the mining group – such a peer will henceforth be referred to as outsider throughout this work (cf. Fig. 2.10). An outsider attempts to infer sensitive information from data exchanged between mining peers. It is assumed that the outsider eavesdrops on the communication among peers in the mining session but forms no collusion with insiders, i.e., the attacker gets no extra information from insiders other than what is

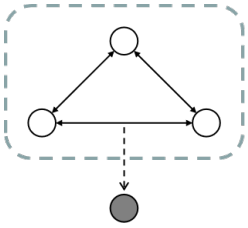


Figure 2.10: *Outsider attack scenario. An outsider who eavesdrops on the communication amongst the peers in a mining group, trying to infer sensitive information from data contained in the messages it manages to intercept.*

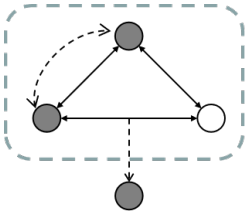


Figure 2.11: *Mixed attack scenario. A collusion of insiders and an outsider attacker try to infer sensitive information about other peers.*

defined in the mining protocol being used.

Mixed attack scenario. In real-world applications, it is possible to have mixed attack scenarios. However, we do not explore this scenario further on this thesis (cf. Fig. 2.11). For the sake of simplicity, we assume that outsiders know less information about the mining group than any insider. If an outsider knows everything about the mining session, just as an insider, we reduce this situation to an insider attack (single or collusion).

In the next section, we discuss the approaches that can be found in the literature to counter the privacy threats when mining sensitive data, a research area known as *privacy-preserving distributed data mining*.

2.3.3 Privacy-Preserving Approaches for Distributed Data

When data is distributed among multiple parties, privacy and data ownership play a major role, which calls for a privacy-preserving solution. Privacy regulations may control how data is used beyond the original purpose for which it was collected – a situation that applies to sensitive data such as healthcare, telecommunication, or customer data [22].

Another critical point to consider is competition; even if the mining re-

sults offer some benefits, giving competitors business data may not be a viable option. For example, if a group of banks wants to learn typical fraud patterns, they may have to address the privacy regulation on customers' data and be sure that the competitors learn nothing that could be used as a business advantage.

Therefore, data integration or aggregation in a distributed data mining process introduces concerns regarding **inference attacks** as a potential privacy threat. The main question here is whether the data mining process or data mining results may compromise the privacy of sensitive information in a distributed setting, even when obtained utilizing privacy-preserving techniques.

To address such concerns, three main approaches have emerged in the field of privacy-preserving distributed data mining: *secure multi-party computation* (Sec. 2.3.4), *distributed model aggregation* (Sec. 2.3.5) and *pertubation-based approaches* (Sec. 2.3.6). With the secure multi party computation (SMC) approach, all computations are performed by the group following a given protocol and using cryptographic techniques to ensure that only the final results will be revealed to the participants. In the distributed model aggregation approach, each site computes a (partial) local model from the local dataset, and, in a second step, all local models are aggregated to produce a global model, which is shared with the participants. The perturbation-based approach adds noise to original data or model parameters to hide sensitive data against disclosure.

In the following discussion, we assume a set $\mathcal{L} = \{L_i\}_{i=1}^{P}$ of agents, each L_i residing at site S_i. Each agent may communicate with several other agents forming a pure peer-2-peer architecture. We assume that only L_i has access to local data set D_i. The size of collusion of malicious agents is denoted c, with $c \leq |\mathcal{L}| - 1$. We often write "site L_i" as a simplification to denote "the agent L_i residing at site S_i".

2.3.4 Secure Multi-Party Computation

Secure multi-party computation (SMC) aims to compute a function in such a distributed fashion that information sharing is minimized, expecting to add only a small overhead in the overall time complexity. In SMC, only the final result can be shared while intermediate information must be kept hidden from the parties [81, 68].

Secure Two-Party Computation was first proposed by Yao [233, 234] with the Millionaires' Problem. The problem is that two millionaires would like to know who is richer without revealing their net worth to one another. Yao [233] presented a solution to this comparison problem, generalizing it to any computable function in the two parties setting. Subsequently, the problem was generalized to multi-party computation in [82].

In general, an SMC scenario is described by P parties, private information $x_1, x_2, \ldots, x_P$ from each party, a public function $f(x_1, \ldots x_P)$ that needs to be computed from the shared data without any of the parties revealing their private information. Additionally, there is no trusted central server in an SMC setting. Otherwise, they could send their data to the central server and wait for the computation result.

A simple example of SMC is secure sum [43]. Consider P sites denoted L_i, each of them holding a private value x_i. The goal is to secretly compute:

$$v = \sum_{i=1}^{P} x_i$$

Assume that the value of v, to be computed, is known to lie in the interval $[0, m]$. One site is chosen to be the master and is numbered 1. Remaining sites are numbered $2, \ldots, P$. Site L_1 generates a random number r, uniformly chosen from $[0, m]$. Site L_1 sends $v_1 = (r + x_i)\ mod\ m$ to site s_2. In the sequel, every site L_l, with $l = 2, \ldots, P$, receives the partial sum $v_{l-1} = r + \sum_{i=1}^{l-1} x_i$ and adds its local value x_l and sends $v_l = (x_l + v_{l-1})\ mod\ m$ to the next site L_{l+1}. The last site sends the last partial v_P to the master site, L_1. Since L_1 knows the value of r, it can compute $v = v_P - r$, which is the output of the secret sum.

Example 2.5 (Secure Sum) *Let $P = 3$ and $x_1 = 0$, $x_2 = -5$ and $x_3 = 7$. Let $m = 20$. Assume $r = 17$, uniformly chosen from $[0, 20]$. L_1 computes $v_1 = 0 + 17 = 17$. L_2 receives v_1 and computes $v_2 = (-5 + v_1)\ mod\ 20 = (-5 + (0 + 17))\ mod\ 20 = 12$. L_3 computes $v_3 = (7 + v_2)\ mod\ 20 = (7 + (-5 + 0 + 17)\ mod\ 20 = 19$ and sends it to L_1. Finally, L_1 computes $v = v_3 - r = 2$, which is the actual result.*

Generic SMC protocols are not efficient for large inputs [130]. Therefore, there was an increased interest in finding efficient SMC protocols for specific applications of secure computations [62, 88, 133].

SMC has been applied to privacy-preserving data mining in several different settings, starting with the work of Lindell and Pinkas [130, 131]. They proposed a modified version of the ID3, called $ID3_\delta$, which gives an approximation of the results generated by the original algorithm. They assume two parties, holding two horizontally partitioned datasets D_1 and D_2, respectively. The basic idea is to find the attribute that maximizes information gain, which reduces to finding the attribute that minimizes the conditional entropy. In ID3, if two attributes have similar entropy levels, the two different trees resulting from choosing one attribute or the other are expected to have similar predicting accuracy. $ID3_\delta$ does not choose the best attribute but chooses any attribute whose entropy differs less than δ from the best one. The conditional entropy of D given an attribute A for two parties $H_Y(D|A)$ can be approximated as a sum of the expressions:

$$(v_1 + v_2) \cdot \log(v_1 + v_2)$$

where v_1 is computed by party L_1 and v_2 is computed by party L_2. The problem lies in how to compute this equation securely. The authors combine secure log, secure polynomial evaluation, and secure comparison subprotocols to evaluate this expression and show how to use this function to build $ID3_\delta$. A multi-party version of ID3 is proposed later by Pinkas [165].

Privacy Measure for SMC

SMC protects the exact values of the inputs from being disclosed to the remaining parties in the group. The protection is achieved via cryptographic methods, e.g., secret shares, homomorphic encryption, oblivious transfers, to name a few. SMC does not measure privacy levels directly but provides analytical proof that a given protocol does not disclose input information during the protocol execution.

Privacy in SMC is, informally speaking, the equivalent of having a trusted third party perform the computation and erasing all of the input data after the computation. An SMC protocol is said to preserve privacy if we can prove that no party learns anything but the final results, as would be the case with a trusted third party in the setting. The above notion of privacy is known as the *simulation paradigm* [81] and is used to define privacy for SMC protocols formally.

Before a formal definition of privacy is put forth, it is necessary to define some basic concepts.

Definition 2.2 (Negligible Function [82]) *A function* $\mu : \mathbb{N} \to \mathbb{R}$ *is* negligible *if for every positive polynomial* p*, and all sufficiently large* n*'s,*

$$\mu(n) < 1/p(n)$$

□

A negligible function decreases faster than the reciprocal of any polynomial. For example, $2^{-\sqrt{n}}$ and $n^{-\log_2 n}$ are negligible (as functions in n).

Definition 2.3 (Indistinguishability [132]) *Let* X *and* Y *be two random variables and a parameter* n *defining the size of inputs. We say that* X *and* Y *are computationally indistinguishable, denoted* $X \stackrel{c}{\equiv} Y$*, if for every non-uniform polynomial-time algorithm* $\mathcal{A}$ *there exists a function* $\mu(\cdot)$ *that is negligible in* n *such that,*

$$|\mathsf{Pr}[\mathcal{A}(X) = 1] - \mathsf{Pr}[\mathcal{A}(Y) = 1]| < \mu(n)$$

where $\mathsf{Pr}[a]$ *denotes the probability of event* a*.* □

Since X and Y cannot be distinguished by a polynomial-time algorithm $\mathcal{A}$, they are the same for all practical purposes. Typically, X and Y will denote the output vectors of the parties in the real (with SMC protocol) and ideal (with trusted third party) executions, respectively. The outputs are modeled as random variables since the operation of the parties is typically probabilistic [132]. Another way to interpret this definition is to say that the output of $\mathcal{A}$ is not significantly different for samples drawn from X or Y.

An essential aspect of SMC protocols is the adversary model. The simplest model is the semi-honest adversary model. **Semi-honest adversary** refers to an adversary who follows the protocols instructions but keeps a record of all messages, and intermediate computations [81]. Although semi-honest is a weak adversary model, it does guarantee that there is no inadvertent information leakage. The semi-honest based protocols are designed as the first step towards more secure ones.

Definition 2.4 (Private Computation from [81, 82]) *Assume* P *parties*

$L_1, L_2, \ldots L_P$. Let f be a random process that maps pairs of inputs into pairs of outputs, one for each party. We say that a given protocol π privately computes f in presence of semi-honest adversaries if there exist probabilistic polynomial-time algorithms S_i, $i \in \{1, 2, \ldots, P\}$, such that for every $x_i \in \{0,1\}^*$ and $\mathbf{x} = (x_1, x_2, \ldots, x_P)$, we have:

$$\{(S_i(x_i, f_i(x_i)), f(\mathbf{x}))\} \stackrel{c}{\equiv} \{(view_i^{\pi}(\mathbf{x}), output^{\pi}(\mathbf{x}))\} \tag{2.1}$$

□

In the above definition, S_i is a simulator used by party L_i. S_i receives as input x_i and $f_i(x_i)$ and must generate output that is (computationally) indistinguishable from the view of L_i in the protocol execution. We use $f(\mathbf{x})$ to represent the output generated from f with the input from all parties. Notice that $view_i^{\pi}(\mathbf{x})$, and $output_i^{\pi}(\mathbf{x})$ are related random variables with probability taken over the random tapes of all the parties [132].

Essentially, to prove that an SMC protocol is private, using the above definition, one needs to show that each party can simulate its view using the protocol output alone. If the simulation is not possible, there is an information leak, and the protocol is not private.

In a **multi-party** scenario it is possible that dishonest parties form a **collusion** (or coalitions) [82, ch. 7]. Members of a collusion group can exchange local inputs, intermediate messages, or local outputs to breach the privacy of the information held by honest members in the group. In the presence of a coalition, the main idea remains the same: a multi-party protocol privately computes a function f if any piece of information learned by a set (or a coalition) of semi-honest parties can be learned from the set of inputs and outputs of said parties.

We stress that the notion of privacy computation is essentially binary; a protocol is either private or not, and it cannot express intermediate degrees of privacy. For a discussion on proofs for SMC protocols, the reader may peruse the discussion in [132] and [82, ch. 7].

The following definition rewrites the original definition as a binary measure. We use the notation $\mathbf{PR}_{\mathcal{A}}$ to denote the privacy level of a given algorithm $\mathcal{A}$. To explicitly indicate privacy measure m in the evaluation of a given algorithm $\mathcal{A}$ we use the notation $\mathbf{PR}_{\mathcal{A}}^{m}$.

Definition 2.5 *For a given dataset D and an algorithm $\mathcal{A}$ and private computation as in Definition 2.4, let $\mathbf{PR}^{PC}_{\mathcal{A}}(D) = 1$ denote that $\mathcal{A}$ privately computes a given function f for all $x \in D$ and $\mathbf{PR}^{PC}_{\mathcal{A}}(D) = 0$ otherwise. The privacy against outsider attacks is denoted by $\mathbf{PR}^{PC}_{\mathcal{A}[0]}(D)$, to indicate that the attack has zero participation of insiders. The privacy in the presence of single inside attackers, without collusion groups, is indicated as $\mathbf{PR}^{PC}_{\mathcal{A}[1]}(D)$. Additionally, the privacy level in the presence of collusion group with at most c members is expressed as $\mathbf{PR}^{PC}_{\mathcal{A}[c]}(D)$.* □

Example 2.6 *Example 2.5 presented secure sum protocol where each party receives a random number in the interval $[0, 20]$. Each party can simulate its view by drawing a random number from this interval. The simulated and actual views cannot be distinguished since both represent random numbers drawn from the same interval. Therefore, the protocol is private, i.e. $\mathbf{PR}^{PC}_{SMCsum[0]}(D) = 1$.*

Example 2.7 (from [132]) *Two parties run a protocol to decide if their two inputs are equal (assume that each is of length k). The protocol works by running a more straightforward protocol that compares two bits. The input to this simpler protocol is a pair of bits taken from the corresponding location in both inputs. The first comparison is of the most significant bits of both inputs and, afterward, successive bits are compared until a difference is found or it is decided that the two inputs are equal. If this protocol stops after i comparisons, the parties can safely conclude that the $i - 1$ most significant bits of their inputs are equal. This information cannot be deduced in the ideal model since the parties are only told if the inputs are equal or not equal. Consequently, the protocol cannot be simulated by one of the parties given her input and output alone. The protocol is not private according to the privacy definition above, i.e. $\mathbf{PR}^{PC}_{SMCBitEq[0]}(D) = 0$.*

Limitations. According to SMC literature, there are two kinds of information leaks in an SMC protocol [109, 132]: (i) the information leak from the function computed, irrespective of the process used to compute the function, and (ii) the information leak from the specific process of computing the function. Privacy breaches related to the first type of information leak are unavoidable in an SMC protocol as long as the function has to be computed [187, 208]. In this case, a privacy breach occurs when any party learns

more than the final result, as shown in Example 2.7. For example, when two parties sum their ages, they may compute each other's age by subtraction. Similarly, if two millionaires compare their net worth, both parties can have a lower or an upper bound on the other millionaire's net worth. The second kind of leak, however, is provably prevented. There is no information leak whatsoever due to the process, i.e., any outside attacker eavesdropping on the communications will not be able to learn the inputs from the participants.

In summary, the private computation measure $\mathbf{PR}^{PC}(D)$ gives us the privacy level from the outsider attackers' point of view. However, it does not indicate the privacy level when malicious insiders are organized as collusion of attackers or the critical number of malicious that may compromise the privacy. Moreover, by being binary, $\mathbf{PR}^{PC}(D)$ does not quantify the amount of privacy breach for a given SMC protocol, and it only indicates when a breach takes place.

Related Work on Secure Multi-Party Computation

SMC has been applied to many data mining techniques. In the following, we present a sample of the research work in this field.

Surveys, Tools, and Frameworks. Clifton et al. [43] described various SMC-based protocols, which can be combined for specific privacy-preserving data mining applications. As the authors pointed out, the proposed techniques are not secure because some information other than the final results is revealed to the parties. The authors presented secure protocols for sum, set union, set union size, and scalar product and showed how to apply them to mine association rules and clustering. Xu and Yi [230] surveyed the field of privacy-preserving distributed data mining and proposed a framework to synthesize and characterize existing SMC protocols. This framework analyzes privacy requirements to help developers design effective and efficient SMC protocols. They use the following dimensions to classify protocols: data partition, algorithm, secure communication model, privacy-preserving techniques.

Bogdanov et al. [26] proposes Sharemind, a toolkit allowing a data mining specialist with no cryptographic expertise to develop data mining algorithms with fundamental security guarantees. The building blocks needed to deploy a privacy-preserving data mining application are listed, and the

design decisions that make Sharemind applications efficient in practice are explained. The Sharemind architecture provides a secure server and a programming language SECRE-C for application developers, which provides explicit confidentiality types: public and private. All private values are secretly shared, and conversion to the public type requires an explicit call of the declassify operator. SECRE-C also automatically parallelizes vector and matrix operations. The architecture has been tested on real-world datasets with several algorithms implemented in the framework. Sharemind uses the secret-sharing cryptographic approach to provide security. Further, it assumes that no two miners will collude.

Teo et al. [197, 198] developed the DAG model: a set of secure operators (including $+$, $-$, $*$, log) that can be combined to produce various functions. The objective is to use these operators as building blocks to implement SMC protocols. To demonstrate the applicability of the DAG model, they implemented kernel regression and naive Bayes algorithm. Currently, DAG operators support only the 2-party case.

We refer the interested reader to [97] for a comprehensive comparison of ten general-purpose frameworks for secure multi-party computation.

Distributed Data Clustering. Vaidya et al. [213], in a seminal paper, proposed a protocol for computing distributed data clustering based on the k-means algorithm. The approach assumes vertically partitioned data and a multi-party scenario with three non-colluding parties. The authors proposed secure protocols to find the closest cluster for a given point, secure permutation, and secure comparison. During the secure comparison, homomorphic encryption ensures the privacy of each data point. Each site learns only its part of each cluster centroid and the cluster assignment of all points at each iteration. Doganay et al. [60] proposed a secret sharing approach to compute the closet cluster in k-means. They achieved better performance than Vaidya et al. [213] since secret sharing avoids expensive cryptographic computation. On the other hand, the authors assume four non-colluding parties. Lin et al. [129] presents a distributed clustering algorithm over horizontally partitioned data, based on the expectation-maximization (EM) algorithm. EM mixture clustering is an iterative process that produces a new set of cluster assignments at each iteration. Over time, these centroids converge to cluster centers. The proposed algorithm does not disclose individual data points,

and no information can be traced back to a specific site. The algorithm uses *secure sum protocol* to integrate the estimators across the different sites. The authors observe that the approach assumes no collusion. Otherwise, a collusion group could learn the centroids and variances of other parties. Gheid and Challal [80] proposed a distributed k-means algorithm based on a secure sum protocol. The mining group securely computes each cluster's sum and the number of points, iteratively, until convergence as in classical k-means. The idea is similar to Clifton's secure sum protocol [43] without the modulo operation. Experimental results indicate that this protocol is scalable with the size of the dataset and the number of parties. However, the security is based on the assumption that there are no collusion groups. Otherwise, the malicious may recover the centroids and number of points of an honest party. Shewale et al. [185] used elliptic curves for key sharing and authentication in a distributed version of the k-means algorithm. Each local party owns a local dataset and shares the sum and count information to other sites. This approach assumes a trusted third party that receives encrypted sums and counting from each party and computes global centroids at each iteration. The trusted third party also verifies the digital signature of each information received. Almutairi et al. [8] extended DBSCAN with homomorphic encryption. First, local parties produce a distance matrix and apply Pailler encryption to protect it before sending it to a central server. The server receives and aggregates all local matrices into a global distance matrix. The server runs with encrypted data to produce an encrypted cluster solution. Finally, local parties decrypt only the data relative to their data. Another secure version of DBSCAN was proposed by Bozdemir et al. [27]. In this case, the authors employed secret sharing to design a ppDBSCAN. Local parties use secret sharing to split original data before sending it to the non-colluding cluster servers. The servers run a 2-parties-computation secure protocol to privately compute distances and discover points in dense regions as in the original DBSCAN. Local parties receive either labels or centroids depending on the implementation.

Application of SMC's Measure to Related Work

Now lets us discuss how the SMC's *private computation* measure, $\mathbf{PR}^{PC}$, indicates the privacy level of SMC-based clustering solutions discussed earlier in this section. All subsequent analyses are derived directly from each

respective work. We consider only distributed clustering approaches (closely related to our work) for three or more parties, as collusion only makes sense with three or more parties. Table 2.2 presents a summary of SMC-based privacy-preserving clustering approaches. This table displays the number of parties necessary to run a given protocol, the number of trusted third parties assumed by the approach, the sensitive info exchanged during a mining session, the privacy level under various attack scenarios, and the smallest collusion group size that causes a privacy leak.

EM-based clustering by Lin et al. [129] works with three or more parties and no trusted third party. Single attackers are not able to learn anything beyond the output. However, when there is a collusion between the first and the last parties, the malicious parties can learn the arguments and variances of honest parties. This represents a breach because the leaked information cannot be learned from output alone. In this case, $\mathbf{PR}^{PC}_{Lin[2]}(D) = 0$.

Similarly, Vaidya's approach [213] and Doganay's [60] work with three or more parties without a trusted third party. Collusion of at least two parties in both schemes also discloses the centroids but not variances. Thus, $\mathbf{PR}^{PC}_{Vaidya[2]}(D) = 0$ and $\mathbf{PR}^{PC}_{Doganay[2]}(D) = 0$.

Gheid and Challal [80] assumes three or more parties and no collusion among them. The proposed secure sum protocol breaches privacy under the collusion of two parties and exposes centroids and several points to other potentially malicious parties. Therefore, $\mathbf{PR}^{PC}_{Gheid[2]}(D) = 0$.

The protocol proposed by Shewale et al. [185] works with three or more parties and needs a trusted third party. Single attackers cannot breach privacy because they need information held by a trusted third party to succeed. However, any collusion between any single malicious party and the trusted third party reveals other parties' sum and the number of points. In this case, $\mathbf{PR}^{PC}_{Shewale[2]}(D) = 0$.

Some key points to notice from this brief overview are as follows. First, SMC is very effective in protecting the inputs from outsiders eavesdropping to the communication. Indeed, all SMC-based data clustering approaches studied in this section have $\mathbf{PR}^{PC}_{[0]} = 1$, i.e., secure against outsider attacks. Furthermore, all approaches are secure against single attackers (no collusion), i.e., $\mathbf{PR}^{PC}_{[1]} = 1$. However, SMC was not designed to protect against malicious insiders. As already mentioned, in a scenario where parties may collude if the majority of parties are corrupted, then an SMC protocol may

Approach	# of Parties	# of Trusted 3rd. Parties	Sensitive info.	Privacy under attack			
				outsider	*single*	*collusion*	*collusion size*
Vaidya [213]	$P > 3$	3	centroids, labels	$\mathbf{PR}^{PC}_{[0]} = 1$	$\mathbf{PR}^{PC}_{[1]} = 1$	$\mathbf{PR}^{PC}_{[2]} = 0$	$c = 2$
Lin [129]	$P > 3$	0	centroids, variances	$\mathbf{PR}^{PC}_{[0]} = 1$	$\mathbf{PR}^{PC}_{[1]} = 1$	$\mathbf{PR}^{PC}_{[2]} = 0$	$c = 2$
Doganay [60]	$P > 4$	4	centroids, distances, labels	$\mathbf{PR}^{PC}_{[0]} = 1$	$\mathbf{PR}^{PC}_{[1]} = 1$	$\mathbf{PR}^{PC}_{[4]} = 0$	$c = 4$
Shewale [185]	$P > 2$	1	cluster sums, number of points	$\mathbf{PR}^{PC}_{[0]} = 1$	$\mathbf{PR}^{PC}_{[1]} = 1$	$\mathbf{PR}^{PC}_{[2]} = 0$	$c = 2$
Gheid [80]	$P > 2$	0	cluster sums, number of points	$\mathbf{PR}^{PC}_{[0]} = 1$	$\mathbf{PR}^{PC}_{[1]} = 1$	$\mathbf{PR}^{PC}_{[M]} = 0$	$c = M < P$

P is the number of parties,c is the collusion group size, $M < P$ is the number of malicious parties in a mining group, $R < P$ is the number of servers.

Table 2.2: *SMC-based Privacy-preserving Clustering Approaches*

fail to provide the correct output and even fail to guarantee that the result will be delivered to the honest parties [132].

Another important aspect to notice is that each algorithm resists collusion attacks up to a certain size of collusion groups. A small group of two malicious may be enough to breach privacy in a given scheme, yet another algorithm may resist a larger collusion group. For instance, VC-SMC protocol by Vaidya and Clifton [213] breaches privacy with $c = 2$ while Patel's ECC-based approach [160, 161] resists to a collusion attack of all against one ($c = P - 1$). The $\mathbf{PR}^{PC}$ measure, however, does not indicate if there is a critical number of malicious that breaks each protocol.

We also stress the binary nature of the $\mathbf{PR}^{PC}$ measure. The goal of the original private computation measure is to indicate if a given SMC protocol leaks information without any further indication on how much privacy was compromised. For example, there is no distinction between disclosing centroids, data points, or the number of points in a given cluster. Nevertheless, each piece of leaked information has a different harm potential to privacy. Leaking a centroid reveals less information than disclosing a raw data point. All leaks, however, are treated as equally harmful by $\mathbf{PR}^{PC}$ metric.

It is also worthy of note that k-means is the most investigated clustering algorithm in privacy-preserving distributed data scenarios. The choice of k-means over other approaches could be related to its simplicity. However, there is still the need to extend this line of research to other clustering approaches since k-means present several well-known drawbacks, e.g., the need to know the number of clusters a priori and the tendency to find ball-shaped clusters.

2.3.5 Distributed Model Aggregation

Model aggregation approaches work by producing local models, later aggregated into a global model. It was first proposed in central data settings to produce models with reduced variance. An aggregated model is computed from several models built from subsets of the data available [29, 61, 71, 173, 179]. Aggregation of models is often formulated as a finite mixture model $f_\lambda(x)$ with k components from a chosen family of models λ [190]:

$$f_\lambda(x) = \sum_{i=1}^{k} \beta_i f_{\lambda_i}(x) \tag{2.2}$$

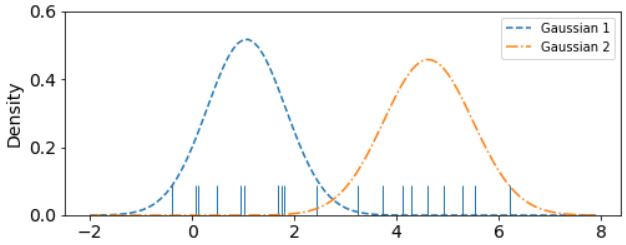


Figure 2.12: *Two Gaussians fitted from dataset D (cf. Example 2.8)*

where $\sum_{i=1}^{k} \beta_i = 1$ and f_{λ_i} denotes a individual component model described by parameters λ_i.

Example 2.8 (From [96](p. 272)) *Assume dataset* $D = \{-0.39,\ 0.12,\ 0.94,\ 1.67,\ 1.76,\ 2.44,\ 3.72,\ 4.28,\ 4.92,\ 5.53,\ 0.06\ ,\ 0.48,\ 1.01,\ 1.68,\ 1.80,\ 3.25,\ 4.12,\ 4.60,\ 5.28,\ 6.22\}$. *Consider the family* λ *of uni-modal Gaussians, each of which is described by a mean* μ *and variance* σ^2. *Let us model the probability distribution function (p.d.f) of this dataset as a mixture of two Gaussian, since it appears to be two different populations on dataset D. After fitting the mixture model to D, we have that each Gaussian*[5] *has the following parameters:* $\lambda_1 = \left(\mu_1 = 1.06,\ \sigma_1^2 = 0.77\right)$ *and* $\lambda_2 = \left(\mu_2 = 4.62,\ \sigma_2^2 = 0.87\right)$, *respectively (cf. Fig. 2.12). The mixture coefficients are* $\beta_1 = 0.45$ *and* $\beta_2 = 0.55$. *Therefore, the mixture model for D is:*

$$f_\lambda(x) = 0.45 f_{\lambda_1}(x) + 0.55 f_{\lambda_2}(x)$$

There are many ways to combine models, like averaging local parameters, averaging the output of local models, majority voting, or training of a meta-model from resampling of local models [179]. The most known families of models used as components are decision trees [71], uni-modal Gaussians and non-parametric kernel density estimates [190].

The idea was extended to the distributed data setting [12, 22, 149, 169, 184, 212, 240] to speed up computation and reduce communication costs,

[5] The p.d.f of a Gaussian with parameters $\lambda_i = (\mu_i, \sigma_i)$ is

$$f_{\lambda_i}(x) = \frac{1}{\sqrt{2\pi\sigma_i^2}} e^{-\frac{(x-\mu_i)^2}{2\sigma_i^2}}$$

as the size of local models exchanged is much smaller than the size of local datasets. The general setting is similar, but each component $f_{\lambda_i}(x)$ now represents a local model owned by a different site. Distributed Model Aggregation has been used for classification [107, 212, 228], clustering [175, 203, 240], manifold discovery and probabilistic models [91, 92, 136].

In the following sections, we first present privacy definitions for distributed model aggregation, and in the subsequent section, we discuss a sample of distributed model approaches found in the literature.

Privacy Measure for Distributed Model Aggregation

The distributed model aggregation approach seeks to protect the exact values owned by each participant from being disclosed to other parties in the group. The protection is achieved by exchanging only data models representing local datasets and not original data points. The *likelihood-based* measure has been proposed in the context of clustering and classification. This measure interprets privacy as the uncertainty that a given dataset was generated from a given probability model [149, 150]. For a dataset $\mathcal{D}$ with probability model $f_\lambda(x)$, the likelihood is given by:

$$L(\mathcal{D}, \lambda) = \prod_{x \in \mathcal{D}} f_\lambda(x)$$

Merugu and Ghosh exploit the fact that the uncertainty is related to the reciprocal of the likelihood. Consequently, when the likelihood is high, i.e., if the model accurately represents the dataset, privacy is low and vice-versa. It is important to remark that this measure pertains to a whole input dataset and not a single data point. Notice that the geometrical mean can be interpreted as the reciprocal of the average likelihood[6], for $\mathcal{D}$ and $f_\lambda(x)$. Thus, they define the privacy measure as:

$$\mathbf{PR}^{like}(\mathcal{D}) = \left(\prod_{x \in \mathcal{D}} f_\lambda(x) \right)^{\frac{-1}{|\mathcal{D}|}}$$

[6]Given the geometrical mean $G = \sqrt[n]{x_1 x_2 x_3 \dots x_n}$, it holds that $\log G = \frac{1}{n} \sum_{i=1}^{n} \log x_i$

Taking the log of both sides, the previous equation can be rewritten as:

$$\log_2(\mathbf{PR}^{like}(\mathcal{D})) = -\frac{1}{\mid \mathcal{D} \mid} \sum_{x \in \mathcal{D}} \log_2 f_\lambda(x)$$

Finally,

$$\mathbf{PR}^{like}(\mathcal{D}) = 2^{\left(-\frac{1}{|\mathcal{D}|} \sum_{x \in \mathcal{D}} \log f_\lambda(x)\right)}$$

Definition 2.6 ([149]) *Let $\mathcal{D}$ be a given dataset and $f_\lambda(x)$ be the probability density function associated with a given probabilistic model λ. The privacy $\mathbf{PR}^{like}$ of data set $\mathcal{D}$ given model λ is defined as:*

$$\mathbf{PR}^{like}(\mathcal{D}) = 2^{\left(-\frac{1}{|\mathcal{D}|} \sum_{x \in \mathcal{D}} \log f_\lambda(x)\right)} \tag{2.3}$$

□

From the above definition, it follows that a higher likelihood of generating a given dataset $\mathcal{D}$ from the model λ would result in a lower amount of privacy. For instance, a highly detailed model comprising a mixture of Gaussians with low variance and centered at each point provides no privacy. Conversely, a coarse model with few Gaussians and high variance has a low likelihood of generating a particular dataset and, therefore, provides a higher privacy level.

Example 2.9 *Given a dataset $\mathcal{D} = \{1, 4, 6, 9\}$ and three models λ_1, λ_2, and λ_3 consisting of a mixture of Gaussians centered at each point $x \in \mathcal{D}$, with variances $\sigma_1^2 = 0.1$, $\sigma_2^2 = 0.5$ and $\sigma_3^2 = 1$, respectively. The probability density functions of each model are denoted as $f_1(x)$, $f_2(x)$, and $f_3(x)$. Using Eq. 2.3 we have, for the first model:*

$$\mathbf{PR}^{like}(\mathcal{D}) = 2^{\left(-\frac{1}{|4|}(\log_2(f_1(1)) + log_2(f_1(4)) + log_2(f_1(6)) + log_2(f_1(9)))\right)} = 1.0027$$

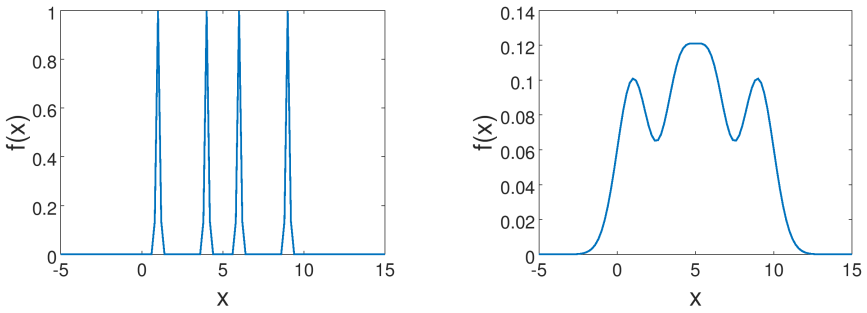
Similarly, we compute the privacy level for $\mathcal{D}$ using λ_2

$$\mathbf{PR}^{like}(\mathcal{D}) = 2^{\left(-\frac{1}{|4|}(\log_2(f_2(1)) + log_2(f_2(4)) + log_2(f_2(6)) + log_2(f_2(9)))\right)} = 5.0124$$

and λ_3

$$\mathbf{PR}^{like}(\mathcal{D}) = 2^{\left(-\frac{1}{|4|}(\log_2(f_3(1)) + log_2(f_3(4)) + log_2(f_3(6)) + log_2(f_3(9)))\right)} = 9.3126$$

Definition 2.6 was proposed in the context of distributed data classification and clustering [149]. In this setting, a dataset owner will publish a model λ instead of the raw data. Therefore, the question is which model should be made public, given the quality and privacy constraints. The choice is up to the dataset owner, who will balance the trade-off between the constraints.



(a) λ_1 is a mixture of 4 Gaussians, centered at each point, with variance $\sigma^2 = 0.1$

(b) λ_3 is also a mixture of 4 Gaussians, centered at each point, with variance $\sigma^2 = 1$

Figure 2.13: *Two different models λ_1 and λ_3 for dataset $\mathcal{D} = \{1, 4, 6, 9\}$, as defined in Example 2.9.*

Intuitively, lower values of $\mathbf{PR}^{like}(\mathcal{D})$ mean that an insider attacker could reconstruct the dataset $\mathcal{D}$ with high likelihood. Consider, for example, the model λ_1 in the previous example, which is a very detailed mixture model (cf. Fig. 2.13(a)). If we generate datasets R from λ_1 with a high likelihood they will be very similar to $\mathcal{D}$, since the variance is very small $\sigma^2 = 0.1$. Still worse, an attacker can choose the dataset generated, which has a maximum likelihood for the given model, optimizing the precision of this reconstruction attack. With higher variances and coarser models, as in the model shown in Fig. 2.13(b), generated datasets will be more dissimilar to the original dataset since the model may produce datasets with a high likelihood in a much wider region around the initial dataset.

This approach provides adequate protection against outsider attacks since it does not exchange original data points, only models. Assuming that outsiders have no information on the model or other relevant parameters, they cannot reconstruct the original dataset.

Limitations. One limitation of the likelihood measure is that it does not consider individual points. Thus, if only a few points are highly likely to be reconstructed with high precision, the overall measure may still indicate a low likelihood for the whole dataset. Therefore, privacy breaches at the data point level may not be detected. One better approach would enforce the minimum privacy level found, not the average level. Moreover, it does not address malicious insiders, which may expose the participants to a malicious central entity attack.

Related Work on Distributed Model Aggregation

Distributed Model Aggregation has been applied to many data mining techniques. In the following, we present a sample of the research work in this field. We grouped the work by the family of models used to represent underlying data.

Parametric representations. One common approach to represent data is via parametric models [92, 136, 239, 240]. Zhang et al. [240] describe how global data models could be learned from local ones, e.g., latent variable model from Gaussian Mixture Model for clustering and manifold discovery [239]. The method does not need resampling from local models and produces a hierarchy of abstracted models, from more fine-grained to a more coarse global model. Liu et al. [136] investigate how to learn probabilistic principal components analysis (PPCA) and Gaussian Mixture Models (GMM). The aggregation is obtained by KL-averaging the local models using Kullback-Leibler distance instead of linear averaging. The authors propose a simple bootstrap approach to generate samples from local models to compute the KL-averaging. Han et al [92] and Han [91] proposes methods to minimize noise when drawing bootstrap samples from local models. We refer the reader to [104] for a survey on distributed model aggregation with parametric models.

Non-parametric approaches. Local data can also be modeled with non-parametric models, e.g., kernel density estimates. Liang et al. [126] proposed REMOLD for distributed data clustering. REMOLD first computes local kernel density estimates and models them as Gaussian distributions, merged based on a connection value defined on the kernel nearest neighbor graph.

The globally merged Gaussian model is finally applied to all points to get a final cluster map. The approach was implemented in Spark on a cluster with one master and four slaves. Lodi et al [140] investigate distributed clustering in a peer-to-peer network. Initially, each peer computes a local kernel density estimate and then queries the value of the local maxima estimates computed at neighboring peers in the network. Each peer can label its local dataset considering all identified maxima in the density estimate.

Other representations. Adjacency graphs and discriminative functions can also be used to represent clusters. Scardapane et al. [181] proposes a distributed spectral clustering algorithm for peer-to-peer networks. First, an Euclidean Distance Matrix (EDM) is computed via a distributed gradient descent optimization algorithm. At each step, local sites exchange a small portion of their local data and a low-rank factorization of the local distance matrix during the computation. When the number of iterations is reached, each local site extracts its Laplacian Matrix from the EDM and applies k-means locally. Shen et al. [184] use high order statistics to improve the performance of distributed clustering. Clusters are modeled by discriminative clustering functions, which define the boundaries of each cluster with no assumption about the probability distribution of each cluster. The optimization problem is solved via distributed gradient descent to find the maximal mutual information for a given set of parameters describing the discriminative functions. Only mutual information is exchanged among peers during the process, and the approximate global mutual information is calculated by a linear combination of local mutual information. The process iterates for a user-defined number of times. Tong et al. [203] propose a distributed clustering algorithm based on boundary information. It first identifies boundary points from local datasets and performs local clustering with the selected points. A fusion step is conducted by a central party taking all local labeled boundaries to produce global boundaries, used locally to cluster local data points. The approach can work with various classic clustering algorithms on the local clustering step, e.g., DBSCAN or Spectral Clustering. It is worth noting that boundary points are not treated to avoid privacy leakage. Rosato et al. [175] follow an ensemble approach to cluster a distributed dataset spread over different sites with the V-DEC algorithm. Each site generates a local cluster, and in a second step, they exchange clusters similarity

index and centroids until convergence is reached. V-DEC can work with any clustering algorithm in the local phase, but the authors demonstrated the idea with k-means. The main goal is performance, not privacy, which is not explored in detail.

Privacy-preserving approaches. The study of privacy-preserving distributed model aggregation has been an active research field. Indeed, ensembles learning has received most of the attention from the data mining community (e.g. [22, 107, 123, 125, 228] and references therein).

Merugu and Gosh [149, 150] present a general framework for distributed learning with privacy. Assuming a given family of parametric models, each party builds a model of its local dataset and sends it to a central location where all models are aggregated. The central location first computes a mean model from local ones, and then artificial data is generated from the mean model. A final model is fitted on the artificial data. Both local and central location models are fitted with the EM algorithm run multiple times to pick the best solution. The authors also propose a privacy measure based on information theory to quantify the privacy of local datasets given the model. The proposed measure is such that if the local models are more detailed, generating a more accurate model, the privacy will decrease. On the other hand, privacy will increase with less detailed local models.

Xiang and colleagues [228] addresses the problem of combining local ensembles of classifiers (e.g., random forests) into a global ensemble with privacy-preservation. The authors presented differential-private versions of Random Forests (RFsDP) and AdaBoost (AdaBoostDP) algorithms to build local ensembles. The algorithm computes the weights of a global mixture of each local ensemble given the size of each local dataset and the accuracy of each local ensemble. Similar work was proposed by Jia et al. [107]. An ensemble of trees is computed via AdaBoost with differential privacy. However, the base learners are CART decision trees. This line of work could be seen as a hybrid of model aggregation and differential privacy.

Bhowmik et al. [22] define distributed learning approach when data is split into several disjoint subsets, and the learner has access only to aggregates. They instantiate the framework to learn Gaussian regression, binary classifiers, and generalized linear models using telecommunication and healthcare data. Each subset originates a weak learner, and the global

model is generated by combining the results from each learner via averaging. Raw data are used only once to compute the local models and never accessed again. The authors do not use any privacy measure to quantify privacy preservation of breaches. Li et al. [125] proposes an algorithm for semi-supervised learning based on a mixture-model, similar to Merugu and Ghosh's approach. They first compute local mixture models and then use a secure protocol to propagate data labels securely. SMC protocols guarantee privacy.

2.3.6 Perturbation-Based Approaches

As discussed earlier in this section (cf. Subsection 2.3.2 on page 28), the inference problem poses several threats in a distributed data mining scenario, even when applying SMC or DMA. SMC was designed to protect inputs from direct disclosure during a distributed computation. However, intermediate results and outputs may still reveal information via inference attacks [187, 208]. Model aggregation also offers an alternative to avoid data disclosure. However, several inference attacks can be deployed against a learned model [144, 229].

Data perturbation has long been investigated as an approach to provide privacy to sensitive data [34, 35]. The idea is to modify original sensitive data or even the mining results to avoid inference attacks. Numerous privacy metrics emerged from this line of research, e.g. *k-anonymity*, *l-diversity*, or *t-closeness*. However, differential privacy is the most popular metric to assess how much privacy is preserved by data perturbation algorithms [59].

Differential privacy is a perturbation-based approach proposed by Dwork [64] as a solution to the privacy-preserving data publishing problem. In this setting, a data holder answer queries issued by an untrusted miner. The data holder will add noise to the data to decrease the risk of disclosing membership of an individual [84]. The goal is to generate summaries that do not change significantly, even if an individual subject had opted out of the data collection. In other words, an algorithm is differentially private if and only if the inclusion of a single tuple in the dataset causes only statistically insignificant changes to the algorithm's output. The primary goal is to protect the so-called **membership privacy** of an individual, i.e., to keep private if a record about a given individual is in the dataset or not.

Definition 2.7 (ϵ-Differential Privacy [64]) *An algorithm A gives ϵ-differential privacy if for all data sets D_1 and D_2 differing in one entry, all outputs $O \in Range(A)$ satisfy*

$$Pr[A(D_1) \in O] \leq e^{\epsilon} Pr[A(D_2) \in O] \tag{2.4}$$

where $Pr[a]$ is the probability of event a. □

The above definition states that the probabilities of outputs from ϵ-differentially private algorithm are bound to a factor of e^{ϵ}. Consequently, the probability distribution of outputs from $A(D_1)$ and $A(D_2)$ are very similar. Some authors prefer to say that in this context $A(D_1)$ and $A(D_2)$ are ϵ-indistinguishable [59]. If ϵ is small enough, an adversary will not know if the output was computed over database D_1 or database D_2 [145]. Therefore, membership privacy is preserved.

The noise added to the data is commonly drawn from a Laplace distribution. The scale of the distribution is defined by the query's sensitivity, which is the maximum possible difference when applying the same query to D_1 and D_2 [63]. Higher sensitivity queries are more likely to reveal individual tuples, and thus they should receive more noise.

One essential aspect of a DP algorithm is the parameter ϵ, also known as the *privacy budget*. Each query reduces the privacy budget, and after the budget is consumed, no further queries can be answered. The challenge is to pick a value big enough to enable a sufficient number of queries before breaching privacy. However, the semantics of ϵ is not well understood and it is an active topic of research [24, 95, 98, 167, 196]. Additionally, there is no systematic way to chose the value (or the range of values) for ϵ [154, 211]. DP assumes an honest data holder. In practice, honest data holder is a strong assumption that motivated the research on distributed differential privacy.

In a **distributed scenario**, no trusted central data holder is assumed. This approach is called Local Differential Privacy (LDP), and each party is responsible for adding noise to their data before releasing it to other parties [227]. Choosing ϵ in LDP is even more challenging since the privacy budget is shared among all the parties and is added up to compose a global ϵ [221]. The downside of LDP is that it requires more significant amounts of noise than in central DP [40]. An intermediate trust model between the central and local is the shuffle privacy. Shuffle-privacy is an alternative definition that requires

less noise at the expense of increased computational cost [24, 41]. In this model, users apply noise to their data, as in the local model. However, each data point is transformed by a trusted shuffler to break the link between data points and individual users. Google's PROCHLO system [24] is a realization of this model, which is easier to achieve than an equivalent SMC protocol. For a comprehensive list of extensions and variations of differential privacy definition, we refer the reader to [59]. DP has also been combined with generative adversarial networks (GANs) in the distributed setting to produce synthetic data as a proxy for local sensitive data. The goal is to train global models with local synthetic data to avoid model inversion attacks [15, 207].

Limitations As we already mentioned, there are a couple of issues related to differential privacy. First, the choice of ϵ and its semantics is still an ongoing question [95, 98, 24, 196, 167, 211]. Second, in the distributed scenario the necessary amount of noise is much larger than in the centralized case [40, 221].

It is interesting to notice that differential privacy has been used in many works as a complementary line of defense to distributed model aggregation or SMC, e.g., [229]. The idea of hybrid approaches promises to reduce the amount of noise added by the differential mechanism. Therefore, we will focus our investigation on the core issues related to SMC and distributed model aggregation.

2.4 Summary

This chapter presented a brief review of knowledge discovery, data mining, its main tasks, and algorithms. Advances in communications technology enabled the development of data mining algorithms for distributed data scenarios. In this context, we presented some of the relevant distributed mining algorithms and discussed the main features of well-known distributed data mining systems. Data distribution poses new challenges to data mining practitioners and researchers, such as preserving data privacy when data mining is performed across the boundaries of different institutions with different authorities' domains.

This chapter also introduced important background concepts, such as privacy, sensitive data, and potential threats to privacy involved in a data

mining process (cf. Section 2.3.2). It has shown that the threat of direct sensitive data disclosure is well addressed by secure multi-party computation and distributed model aggregation. However, the other threats are harder to address since they involve the *inference problem*.

Inference attacks are not as easy to control as one wishes, and they are hard to detect and handle. Functional dependencies among data may reveal unexpected inference channels, and mathematical properties may allow accurate reconstructions even from data aggregation. Furthermore, malicious peers can organize themselves into *collusion groups* to get more accurate inference results against honest parties. This study defined a privacy framework to analyze distributed data mining algorithms for several general inference attack scenarios.

Numerous pieces of relevant existing studies on privacy-preserving distributed data mining have been considered, including both secure multi-party and distributed model aggregation approaches. The different privacy formalizations have also been presented, with their specific assumptions and limitations. Existing approaches protect against outsider attacks but may not be safe against insider attacks. Table 2.3 summarizes the privacy-preserving approaches and measures discussed in this chapter.

	SMC	Dist. Model Aggregation	Perturbation-based
Applications	Association rules, classification, clustering	Clustering, classification, manifold detection	Classification, clustering, etc.
Privacy measure	Private computation	Likelihood-based measure	Differential privacy
Protects	Exact values of inputs	Input dataset	Membership privacy
Interpretation	Privacy is achieved when real, and the ideal execution scenarios are indistinguishable	Privacy is viewed as the average likelihood of data being generated from a given Mixture Model	Privacy is achieved when the inclusion of a single tuple does not change probability distribution of outputs too much
Assumptions	Privacy proofs are based on simulation paradigm	Cluster map modeled as a model mixture	Noise only need to be proportional to queries sensitivity. Need to trust in the entity that applies noise.
Good against	Outsider attacks, due to cryptographic techniques	Outsider attacks, since only models are exchanged instead of data points	Insiders and outsiders, depending on who applies noise.
Limitations	(i) Does not consider malicious insider attacks; (ii) it is a binary measure that does not quantify the amount of privacy breach (p. 38)	(i) Does not consider malicious insider attacks; (ii) does not model privacy breaches at data point level, only dataset level breaches (p. 49)	(i)choice of ϵ and its semantics is still an ongoing question; (ii) distributed scenario requires more noise than the centralized case

Table 2.3: *Summary of privacy measures for distributed data mining*

Chapter 3

Privacy Measures Revisited

> "She went fairly often to the hut, in the morning or the afternoon, but he was never there. No doubt, he avoided her on purpose. He wanted to keep his privacy."
>
> (D. H. Lawrence in *Lady Chatterley's Lover*)

> "We all have a right to privacy," she said. "Nobody should have found this all out."
>
> (Thelma Arnold in *New York Times, 9 August 2006*)

The previous chapter presented the main approaches for privacy-preserving distributed data mining. It also pointed out the general limitations of current approaches, mainly related to inference attacks by malicious insiders.

This chapter introduces a set of privacy properties to capture the main features a privacy measure should have to avoid the limitations identified thus far. The goal is to formalize the limitations identified informally in the studied privacy measures allowing the domain experts to define new privacy measures that improve those limitations.

As the main threats in a distributed mining session come from malicious insiders trying to infer sensitive information, a privacy measure should consider the presence of collusion groups of malicious peers. Moreover, a privacy measure should detect the privacy of single data points. Further, this chapter introduces new privacy measures for distributed data clustering and time series, which address inference and collusion. The new measures are then applied to some representative distributed privacy-preserving algorithms.

3.1 Privacy: Properties and Notations

Classical understanding of **privacy** sees it as the right to avoid the access to sensitive information by other agents [20, 65, 147]. In this view of privacy, the privacy level is given as a binary variable: {`private, public`}. To say that some information is *private* means that other agents have no access to it. A **privacy breach** is the unauthorized disclosure of private information.

An alternative definition allows us to see privacy as a continuous variable. For example, Agrawal and Aggarwal [4] propose an information-theoretical approach to privacy as the uncertainty about sensitive information. According to this view, privacy is the right to avoid disclosing exact values to other agents. Consequently, other agents should not determine sensitive values down to a given interval. In this case, the privacy level is zero for public information and is some positive value for private information. Privacy of sensitive data is preserved if the privacy level does not decrease when an agent participates in a data mining protocol.

Example 3.1 *Assume a party P_1 holds a sensitive variable $x \in (20, 25)$ indicating the age of an individual. The variable x is private if other parties have no access to its exact value and only know that x is a number between* 20 *and* 25*. Privacy of x is preserved if, after executing a data mining protocol, other agents still only know the original interval, i.e.,* $(20, 25)$*.*

Now, the question is how to measure the amount of privacy in a distributed setting. In this study, we will regard a **privacy measure** as a function that, for a given distributed data mining algorithm, maps a dataset subset and the maximum size of collusion groups of parties to a real number and satisfies certain properties. We will call the value of such a measure a *privacy level*.

Throughout this study, we use the following notation. Let $L_1, \ldots, L_P$ be sites hosting one element of a partition of a dataset D each, and $\mathcal{A}$ be any distributed data mining algorithm running on $L_1, \ldots, L_P$. We will assume that up to $P - 1$ sites among $L_1, \ldots, L_P$ are malicious, in that they seek to infer objects of dataset D, or parts thereof, possibly in collusion groups of at most $c < P$ members, by either exchanging information or violating the protocol of $\mathcal{A}$, or both. When no party attempts to learn sensitive information held by other parties, i.e., are honest, there are no inside threats

and $c = 0$. To explicitly indicate a privacy measure m in the evaluation of a given algorithm $\mathcal{A}$ we use the notation $\mathbf{PR}^m_{\mathcal{A}}$. We indicate the privacy of a given singleton $\{x\}$, given an algorithm $\mathcal{A}$ and measure m, as $\mathbf{PR}^m_{\mathcal{A}}(\{x\})$, and overload the notation as $\mathbf{PR}^m_{\mathcal{A}}(x)$; for a dataset D we use $\mathbf{PR}^m_{\mathcal{A}}(D)$. To indicate presence of collusion groups of maximum size c we write $\mathbf{PR}^m_{\mathcal{A}[c]}(D)$. For the sake of simplicity, we omit algorithm, collusion size, measure, dataset, or data point, when they are implicit in a given context.

By *privacy measure* for $\mathcal{A}$ we mean a computable partial function

$$\mathbf{PR}_{\mathcal{A}} \colon (X, c) \in 2^D \times \{0, 1, 2, \ldots, P-1\} \to \mathbf{PR}_{\mathcal{A}[c]}(X) \in [0, \infty) \qquad (3.1)$$

which satisfies one or multiple of the following properties:

P1 (inference and collusion) $\mathbf{PR}_{\mathcal{A}[0]}(X) > \mathbf{PR}_{\mathcal{A}[c]}(X)$ when there are at most c malicious peers colluding, with $c \in \{1, 2, \ldots, P-1\}$, for all $X \subseteq D$;

P2 (point monotonicity) it is nonincreasing from singletons to dataset, i.e., $\mathbf{PR}_{\mathcal{A}[c]}(\{x\}) \geq \mathbf{PR}_{\mathcal{A}[c]}(D)$ for all $c \in \{0, 1, \ldots, P-1\}$.

P3 (interpretability): For a given data point $x \in D$, there must be a mapping from the privacy measure $\mathbf{PR}(\{x\})$ to some property of x or dataset D.

Property **P1** expresses the decrease of privacy level when inference attacks by malicious parties alone or in collusion groups take place. Note that $c = 0$ denotes only honest insiders, $c = 1$ expresses the presence of dishonest parties but no collusion, and $c > 1$ denotes collusion groups with more than one member. Property **P2** constrains $\mathbf{PR}_{\mathcal{A}}$ to behave as a worst-case measure: a greater privacy level than the one at singletons is not attainable for the dataset. We call this property *point-level* awareness. Property **P3** requires a natural interpretation from the privacy measure into the sensitive data domain. It should be natural for a domain expert to choose the desired amount of privacy to be preserved expressed with the expert's vocabulary. For instance, the measure could express the number of points in a subset of D or if x is present/absent in D. Additionally, Eq. (3.1) requires that privacy measures have non-negative codomain.

There is an increasing interest in the pertinent literature to define desirable properties for privacy measures explicitly. Zhao and Wagner [244] proposed four properties in the context of the vehicular communication domain. Bezzi [21] emphasizes the importance of point-level privacy compared to the prevailing view of dataset privacy protection. Pufferfish privacy [116, 117] and [59] presents a rigorous framework for privacy definitions in the context of data publishing. We share the same motivation, but we focus on distributed data scenarios.

3.2 Analysis of Existing Privacy Measures

In the following, private computation and likelihood-measure are analyzed concerning the set of properties proposed in the previous section [1]

3.2.1 Private Computation Measure for SMC

In the preceding chapter, it was shown that in the presence of collusion groups, a secure multi-party protocol (SMC) is likely to fail [132]. SMC fails because the semantic of privacy computation measure only gives the privacy level from the outsiders' point of view. Any malicious insiders will receive the correct output, from which they may try to reconstruct sensitive inputs owned by other parties [208] or information about these parties [187]. The private computation measure is non-negative definite and fulfills properties P2 and P3 but fails to satisfy P1, as discussed below.

$\mathbf{PR}^{PC}(D)$ is non-negative. According to the definition of private computation (Def. 2.4), a computation is private if no party learns each other's input. $\mathbf{PR}^{PC}(D)$ is a binary measure, indicating if any leak occurred or not, without any indication of the degree of the privacy breach. Indeed, Def. 2.5 explicitly uses 0 to indicate that a leak occurred during the protocol's execution and 1 to indicate that the protocol is private. Therefore, $\mathbf{PR}^{PC}(D) \geq 0$.

$\mathbf{PR}^{PC}(D)$ does not address inference or collusion ($\neg$P1). The private computation measure, $\mathbf{PR}^{PC}(D)$, was designed to detect leaks from the protocol and not from outputs. In an SMC protocol, information that is

[1] As we mentioned previously, we will not discuss perturbation-based measures further in this thesis.

leaked from the results is unavoidable as long as the function has to be computed [109]. For that reason, the private computation measure $\mathbf{PR}^{PC}(D)$ concerns only with leaks due to the process of computing the function. This measure does not address inferences from the output, intermediate messages, or any advantages due to collusion among the parties.

Example 3.2 *Consider example 2.5 where three parties compute the sum of numbers in an SMC protocol. The process does not leak any information, as shown in example 2.6. Therefore, with* 0 *malicious parties,*

$$\mathbf{PR}^{PC}_{SMCSum[0]}(x) = 1$$

Now, suppose that two parties collude. In this case, they can subtract their input and learn the input of the third party. However, privacy computation does not care about inferences after the process, and

$$\mathbf{PR}^{PC}_{SMCSum[2]}(x) = 1$$

meaning that even with two colluders, the process of computation does not leak information. Notice that $\mathbf{PR}^{PC}_{SMCSum[0]}(x) = \mathbf{PR}^{PC}_{SMCSum[2]}(x)$ *when* $\mathbf{PR}^{PC}_{SMCSum[2]}(x)$ *should be* 0 *to indicate that the privacy is not preserved in the presence of 2 malicious parties working in collusion.*

$\mathbf{PR}^{PC}(D)$ is point level (P2). By definition, if any point $x \in D$, the dataset of inputs of a given party, is leaked, the protocol is considered not private, i.e. $\forall x \in D : \mathbf{PR}^{PC}(x) = 0 \rightarrow \mathbf{PR}^{PC}(D) = 0$. Therefore, $\forall x \in D : \mathbf{PR}^{PC}(x) \geq \mathbf{PR}^{PC}(D)$.

$\mathbf{PR}^{PC}(D)$ has clear interpretation (P3). The interpretation of the private computation measure in SMC is that for any input $x_i \in D$ belonging to party p_i, no party p_j learns x_i, or anything about x_i, with $i \neq j$, directly from protocol execution. A privacy breach occurs when any information about x_i is leaked during the process.

3.2.2 Likelihood-Based Measure

In the previous chapter, we discussed the likelihood-based measure introduced by Merugu et al. [149]. The likelihood-based measure is non-negative and fulfills property P1. However, as discussed below, it does not fulfill

properties P2 or P3.

$\mathbf{PR}^{like}(D)$ is non-negative. By definition of likelihood-based privacy measure (Def. 2.3 and Eq. (2.3)), $\mathbf{PR}^{like}(D) \geq 0$ because it is defined as a power of 2.

$\mathbf{PR}^{like}(D)$ addresses inference and collusion (P1). It addresses inside attacks, which may expose the participants to a malicious central entity attack.

Example 3.3 *Consider a dataset $D = \{1, 4, 6, 9\}$ and a model formed by the mixture of two Gaussian. Let the first Gaussian be centered at x_0, i.e. it has mean $\mu_1 = 1$ with variance $\sigma_1^2 = 0.1$. The second Gaussian models the three remaining points, i.e. has mean $\mu_2 = 6.33$ and variance $\sigma_2^2 = 1.0$. With probability density function of the mixture model denoted by $f(x)$, using Eq. (2.3) we have:*

$$\mathbf{PR}^{like}_{[0]}(D) = 2^{\left(-\frac{1}{|4|}(\log_2(f(1))+log_2(f(4))+log_2(f(6))+log_2(f(9)))\right)} = 13.7326$$

If attackers know the mixture model, it is possible to compute the privacy level for a given reconstructed set, for example, $R = \{1\}$:

$$\mathbf{PR}^{like}_{[1]}(R) = 2^{\log_2 f(1)} = 0.5013$$

The privacy measure of the original dataset is 13.7326 but drops to 0.5013 when the reconstructed dataset consists of points close to the mean of first Gaussian.[2] *Therefore, a malicious central entity, which receives the mixture model, may reconstruct the point x_0 by choosing a set of points R that minimizes $\mathbf{PR}^{like}(R)$.*

$\mathbf{PR}^{like}(D)$ is not point-wise ($\neg$P2). When only a few points have a high likelihood of being reconstructed with high precision, $\mathbf{PR}^{like}(D)$ measure will still indicate high privacy protection. Consequently, privacy breaches at the data point level may not be detected. The breach is a consequence of $\mathbf{PR}^{like}(D)$ being defined as the geometrical mean, as illustrated in the following example.

Example 3.4 *Consider a dataset $D = \{1, 4, 6, 9\}$ and a mixture of two*

[2] The same behavior occurs if the reconstructed dataset is $R = \{6\}$, but in this case, $\mathbf{PR}^{like}(R)$ drops to 5.299.

Gaussian as in the previous example. With probability density function of the mixture model denoted by $f(x)$, using Eq. (2.3) we have:

$$\mathbf{PR}^{like}(D) = 2^{\left(-\frac{1}{|4|}(\log_2(f(1))+log_2(f(4))+log_2(f(6))+log_2(f(9)))\right)} = 13.7326$$

Let's examine the privacy measure $\mathbf{PR}^{like}(x)$ for each point $x \in D$:

$$\mathbf{PR}^{like}(1) = 2^{-\log_2 f(1)} = 0.5013$$
$$\mathbf{PR}^{like}(4) = 2^{-\log_2 f(4)} = 76.272$$
$$\mathbf{PR}^{like}(6) = 2^{-\log_2 f(6)} = 5.299$$
$$\mathbf{PR}^{like}(9) = 2^{-\log_2 f(9)} = 175.500$$

Notice that the geometrical mean in the privacy measure smoothed out the measure for $x_0 = 1$, masking a possible privacy breach. In that case,

$$\mathbf{PR}^{like}(1) = 0.5013 < \mathbf{PR}^{like}(D) = 13.7326$$

Therefore, $\mathbf{PR}^{like}(D)$ does not fulfill property P3.

$\mathbf{PR}^{like}(D)$ has no clear interpretation ($\neg$P3). For a given (sensitive and hidden) dataset D and a public model λ, $\mathbf{PR}^{like}(D)$ is the reciprocal of the geometrical average likelihood of points being generated from the given model. As a consequence, when the model is a good fit for the data, $\mathbf{PR}^{like}(D)$ is small. In this case, the model is said to leak too much information about the original data. Conversely, if the model is not a good fit for the data, $\mathbf{PR}^{like}(D)$ is large, and the model is said to preserve information. The problem with this approach is that the average likelihood does not translate well to the data domain; it is unclear how much protection is provided in the original data space, i.e., in terms known to a domain expert.

Table 3.1 presents a summary of all studied privacy measures and their properties.

	Reference	Infer. and Collusion (P1)	Point monotonicity (P2)	Interpretability (P3)
$\mathbf{PR}^{PC}$	[82]	no	yes	yes
$\mathbf{PR}^{like}$	[149]	yes	no	no
$\mathbf{PR}^{range}$	(Def. 3.1)	yes	yes	yes
$\mathbf{PR}^{rec}$	(Def. 3.2)	yes	yes	yes
$\mathbf{PR}^{BK}$	(Def. 3.3)	yes	yes	yes
$\mathbf{PR}^{TBK}$	(Def. 3.6)	yes	yes	yes

Table 3.1: *Summary of privacy measures*

3.3 New Privacy Measures for Clustering and Time Series Mining

As discussed in previous sections, the general idea behind privacy measures involves formalizing the intuitive notion regarding the protection, or lack thereof, of a piece of sensitive information from unauthorized access. This section proposes new privacy measures and uses them to analyze specific distributed data mining algorithms in subsequent chapters. These measures assume that the attacker may be a member of the mining group. Furthermore, these measures also focus on point-level privacy, not only the dataset level.

3.3.1 Privacy Measures for Distributed Clustering

Due to its inherent descriptive nature, any cluster map reveals information about the data. Indeed, this is one of the primary purposes of clustering: to provide an initial idea of the overall data distribution. From a privacy-focused point of view, the interesting question is: how does one measure the amount of privacy loss caused by a given cluster map?

We define cluster privacy as the size of the range of values in a given dimension. For instance, a cluster of data points over the dimension annual income ranging from US$ 100 000 to US$ 150 000 reveals the value of each data point with a maximal absolute error of US$ 50 000 and mean absolute error of US$ 25 000, assuming uniform distribution[3]. Consequently, if it is known that a specific person is modeled as a point in this cluster, then it is known, with 100% certainty, this person has an annual income of US$

[3]The maximal absolute error happens when an attacker guesses a point to be in one extreme when it is actually in the other extreme of the range.

125 000 ± 25 000. This cluster, then, is said to have a privacy level of 25 000 dimension units, US$ in this case.

Definition 3.1 (Cluster range measure) *Given a dataset of reals, i.e. $D \subset \mathbb{R}$ and a cluster map $\mathcal{C} = \{C_k\} \subseteq 2^D$, whose elements C_k are pairwise disjoint. We define the cluster privacy of a given point x in a given cluster $C_k \in \mathcal{C}$ as:*

$$\mathbf{PR}^{range}(x) = \max C_k - \min C_k \tag{3.2}$$

Extending to the whole dataset:

$$\mathbf{PR}^{range}(D) = \min\{\mathbf{PR}^{range}(x) : x \in D\} \tag{3.3}$$

$$= \min\{\max C_k - \min C_k : k = 1, \ldots, |\mathcal{C}|\} \tag{3.4}$$

□

$\mathbf{PR}^{range}$ serves as a reminder that the cluster map itself gives away information on the original data. The smaller a given cluster range is, the greater the amount of information given away about the location of points. To avoid (or control) this information leakage, privacy-preserving algorithms must adopt a mechanism that allows the user to control the minimum range of each cluster.

Assuming the attackers only know the boundary of clusters (revealed by the cluster map and known by the members of the mining group), they cannot learn anything new except for the range of values covered by each cluster. This metric is based on the assumption that points inside a cluster follow a uniform distribution from the attackers' point of view.

Example 3.5 *Consider Figure 3.1, representing a dataset with by 6 points, $D = \{0, 0.3, 0.6, 1.0, 2.5, 2.7\}$. The points in this dataset form two clusters: $C_1 = \{0, 0.3, 0.6, 1.0\}$ and $C_2 = \{2.5, 2.7\}$. When the cluster map is made public after the mining session, attackers know that points in cluster C_1 range from 0 to 1 and that points in cluster C_2 range from 2.5 to 2.7. For all points x_i in cluster C_1 we have:*

$$\mathbf{PR}^{range}(x_i) = 1 - 0 = 1$$

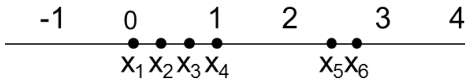


Figure 3.1: *Example of a dataset with two clusters.*

Similarly, for all points x_j in cluster C_2 we have:

$$\mathbf{PR}^{range}(x_j) = 2.7 - 2.5 = 0.2$$

Therefore,

$$\mathbf{PR}^{range}(D) = \min\{1, 0.2\} = 0.2$$

This result indicates that at least one region of D may be vulnerable to reconstruction within an interval of size 0.2.

This study proposes another measure, one based on reconstruction. If a reconstruction method is known, it is possible to measure how close the reconstructed data gets to the original sensitive data.

Definition 3.2 (Reconstruction based measure) *Let $R \subset \mathbb{R}$ denote a set of reconstructed data points such that each $r_i \in R$ is a reconstructed version of $x_i \in D \subset \mathbb{R}$. We define the privacy level, given a reconstruction method, by:*

$$\mathbf{PR}^{rec}(x_i) = \mid x_i - r_i \mid \tag{3.5}$$

Extending to the whole dataset:

$$\mathbf{PR}^{rec}(D) = \min\{\mathbf{PR}^{rec}(x_i) : x_i \in D, r_i \in R, 1 \leq i \leq |D|\} \tag{3.6}$$

where $|D|$ is the size of the dataset $D \subset \mathbb{R}$. □

Roughly speaking, $\mathbf{PR}^{rec}$ indicates the precision with which a data object $\mathbf{x}_i$ may be *reconstructed* for a given cluster map and a reconstruction method. Note that the reconstruction method does not need to be an exact function; it can be a heuristic-based method for partially reconstructing the data. With regards to privacy preservation, even partial reconstructions need to be considered.

Example 3.6 *Assume that we compute the density estimate from the cluster in Fig. 3.1 using triangle kernel (which is a bounded kernel). Assuming*

that the attackers know the density estimates, they can locate data points by locating the kernel borders [4]*. The error in this scenario is down to machine precision, assuming the attackers know the estimation parameters, which, as discussed in the previous chapter, is a very reasonable assumption when the attacker is an insider. Therefore, with triangle kernel, in an inside attack this cluster has:*

$$\mathbf{PR}^{rec}(D) \approx 0$$

A similar privacy metric is the *relative error*, introduced by Lyu et al. [143]. *Relative error* uses 2-norm to evaluate how different is the recovered dataset compared to the original data, while $\mathbf{PR}^{rec}(D)$ uses absolute error. Lyu and colleagues proposed *relative error* metric in the context of data perturbation approach for central data setting.

A general definition of privacy proposed in the centralized data mining setting is the *bounded knowledge* measure [4], which defines privacy as the length of the interval from which a random variable X is generated. This measure can be expressed in terms of the entropy of X, as follows.

Definition 3.3 (Bounded Knowledge) *Given a random variable X with probability density function f_X and domain Ω_X, the privacy of X is given by its bounded knowledge is:*

$$\mathbf{PR}^{BK}(X) = 2^{h(X)} \tag{3.7}$$

where the differential entropy $h(X)$ is given by

$$h(X) = -\int_{\Omega_X} f_X(x) log_2 \left[f_X(x)\right] dx$$

□

Example 3.7 *A random variable X uniformly distributed between 20 and 70, abbreviated $X \sim U(20, 70)$, has probability density function given by:*

$$f_X(x) = \begin{cases} \frac{1}{50} & \text{for } 20 \leq x \leq 70 \ , \\ 0 & \text{otherwise.} \end{cases}$$

[4] See Section 4.2.3 for a discussion on single insider attacks using density estimates.

The entropy of X is $h(X) = log_2(50)$. Then, the privacy provided by X according to bounded knowledge measure is $\mathbf{PR}^{BK}(X) = 2^{log_2(50)} = 50$.

This definition only assumes that the sensitive attribute can be modeled as a random variable, i.e., it assumes we can compute the probability distribution of the sensitive attribute. This assumption is general enough to be used in different data mining contexts, such as cluster analysis and association rules [20]. This measure is also known as Inherent Privacy [216] and denote the number of bits describing each element in the domain of the X.

For a given datapoint $x \in C_i$, a cluster C_i in the cluster map $\mathcal{C}$ induced from dataset $D \subset \mathbb{R}$, a random variable X_i with domain C_i and probability density function $f_{X_i}(x)$ being zero outside C_i, let:

$$\mathbf{PR}^{BK}(x) = \mathbf{PR}^{BK}(X_i) = 2^{h(X_i)} \tag{3.8}$$

In the case of a cluster map, we are interested in the smallest interval size in the said map. Therefore,

$$\mathbf{PR}^{BK}(D) = \min\{\mathbf{PR}^{BK}(x)\} = \min\{2^{h(X_i)}\} \tag{3.9}$$

It is interesting to note that the $\mathbf{PR}^{range}$ and bounded knowledge measure $\mathbf{PR}^{BK}$ (cf. Eq. (3.9)) are related. Indeed, $\mathbf{PR}^{range}$ is a special case of the bounded knowledge measure $\mathbf{PR}^{BK}$ when the probability density function $f(x)$ is assumed to be the uniform distribution. Therefore, the privacy level indicated by bounded knowledge measure, $\mathbf{PR}^{BK}$, tends to be tighter than the privacy level indicated by $\mathbf{PR}^{range}$ because the former uses more information to compute the privacy level.

When there is no known reconstruction method, the reconstruction error is bounded to the entire domain of possible values, giving no information to the attacker. In this case we denote $\mathbf{PR}^{rec} = \infty$. Furthermore, if the range of clusters is not known, we denote it as $\mathbf{PR}^{range} = \infty$. Similarly, with no distribution known, we have $\mathbf{PR}^{BK} = \infty$.

The following definition extends each of the previously defined measures to include collusion groups.

Definition 3.4 (Distributed Cluster Privacy) *Let $\mathcal{A}$ be a distributed data clustering algorithm, $D \subset \mathbb{R}$ be a dataset, and a privacy measure $m \in \{rec, range, BK\}$, with collusion groups containing at most c attackers. We*

define:

$$\mathbf{PR}^{DCP}_{\mathcal{A}[c]}(D) = \min\{\mathbf{PR}^{m}_{\mathcal{A}[i]}(D) : \ 0 \leq i \leq c\} \tag{3.10}$$

□

The function $\mathbf{PR}^{DCP}_{\mathcal{A}[c]}(D)$ represents the minimum privacy level provided to dataset D when the collusion groups have at most c peers. For example, $\mathbf{PR}^{BK}_{\mathcal{A}[2]}(D)$ denotes the smallest privacy level provided by algorithm $\mathcal{A}$ to dataset D when collusion groups contain at most two malicious peers. The focus on the smallest level represents the idea that the weakest scenario is the most dangerous one[5]. When all parties are honest, we have $c = 0$. In summary, $\mathbf{PR}^{range}$ measures the privacy level achieved by the disclosure of the cluster map, assuming that the attacker knows nothing else. Therefore, $\mathbf{PR}^{range}$ is suited for measuring the privacy level in the single inside attack scenarios. In a more general setting, $\mathbf{PR}^{BK}$ gives the privacy level when the distribution of data is known to the attacker or collusion group, i.e., $\mathbf{PR}^{BK}$ can be used in a single or collusion attack scenario. When a reconstruction function is known, the $\mathbf{PR}^{rec}$ measure gives a more accurate privacy level. $\mathbf{PR}^{rec}$ can be used in single or collusion attack scenarios. Therefore, deciding which measure is the most appropriate relies on the information that is assumed to be available to the attacker or collusion group: cluster map only, data distribution, or reconstruction function.

Properties Analysis of $\mathbf{PR}^{DCP}_{\mathcal{A}[c]}(D)$

In the following, we state that $\mathbf{PR}^{DCP}_{\mathcal{A}[c]}(D)$ is non-negative (Theorem 3.1) and has the following properties:

P1 (inference and collusion): theorem 3.2;

P2 (point level privacy): theorem 3.3;

P3 (interpretation): theorem 3.4.

Lemma 3.1 $\mathbf{PR}^{range}(D) \geq 0$, *for all dataset* $D \subset \mathbb{R}$.

Proof. $\mathbf{PR}^{range}(D)$ is defined as $\min\{\max C_k - \min C_k \ \}$. Notice that the definition uses the difference between the max and min values in a given

[5]This notion comes from the popular idea in computer security that defines the security level of a system as the level of its *weakest link*.

cluster. It follows that,

$$\forall C_k \in \mathcal{C} : \max C_k - \min C_k \geq 0$$

□

Lemma 3.2 $\mathbf{PR}^{rec}(D) \geq 0$, *given dataset* $D \subset \mathbb{R}$ *and reconstructed set* $R \subset \mathbb{R}$.

Proof. $\mathbf{PR}^{rec}(D)$ is defined as $\min\{|\mathbf{x}_i - \mathbf{r}_i| : \mathbf{x}_i \in D, \mathbf{r}_i \in R, 1 \leq i \leq N\}$. Since $|\mathbf{x}_i - \mathbf{r}_i| \geq 0$ by definition of the absolute function, it follows that $\mathbf{PR}^{rec}(D) \geq 0$ □

Lemma 3.3 $\mathbf{PR}^{BK}(X) \geq 0$, *for all random variable* X.

Proof. $\mathbf{PR}^{BK}(X)$ is defined as $2^{h(X)}$. Since $h(X) \in \mathbb{R}$ and $2^a \geq 0$ for all $a \in \mathbb{R}$, it follows that $2^{h(X)} \geq 0$. □

Theorem 3.1 $\mathbf{PR}^{DCP}_{\mathcal{A}[c]}(D) \geq 0$, *for all dataset* $D \subset \mathbb{R}$ *and privacy measures* $m \in \{range, BK, rec\}$.

Proof. $\mathbf{PR}^{DCP}_{\mathcal{A}[c]}(D)$ is defined over basic measures $\mathbf{PR}^{range}$, $\mathbf{PR}^{BK}$, and $\mathbf{PR}^{rec}$. By lemmas 3.1, 3.3 and 3.2, it holds that $\mathbf{PR}^{range} \geq 0$, $\mathbf{PR}^{BK} \geq 0$, and $\mathbf{PR}^{rec} \geq 0$ □

Theorem 3.2 *Given an algorithm* $\mathcal{A}$, *for all dataset* $D \subset \mathbb{R}$ *and privacy measures* $m \in \{range, BK, rec\}$, *and* $c > 1$ *(presence of collusion groups), if there is a collusion scenario decreasing the privacy level of dataset* D, *then* $\mathbf{PR}^{DCP}_{\mathcal{A}[1]}(D) \geq \mathbf{PR}^{DCP}_{\mathcal{A}[c]}(D)$.

Proof. Let $a = \mathbf{PR}^{DCP}_{\mathcal{A}[1]}(D)$ be the privacy level of dataset D with algorithm $\mathcal{A}$ with no collusion (i.e., $c = 1$), and $b = \mathbf{PR}^{DCP}_{\mathcal{A}[c]}(D)$ be the privacy level of dataset D with algorithm $\mathcal{A}$ in a collusion scenario with $c > 1$ malicious peers. By definition $\mathbf{PR}_{\mathcal{A}[c]}{}^{DCP}(D)$ is defined as the smallest privacy level considering all collusion scenarios. In that case, $\mathbf{PR}^{DCP}_{\mathcal{A}[c]}(D) = \min\{a, b\}$. Therefore, if the collusion group decreases the privacy level of the $c = 1$ scenario, then $a \geq \min\{a, b\}$. □

Lemma 3.4 $\forall x \in D : \mathbf{PR}^{range}_{\mathcal{A}[c]}(x) \geq \mathbf{PR}^{range}_{\mathcal{A}[c]}(D)$, *for all dataset* $D \subset \mathbb{R}$.

Proof. Consider a cluster map $\mathcal{C}$ from D, with k clusters $C_1, C_2, \ldots, C_k$. Let r_i denote $r_i = \max C_i - \min C_i$, the cluster range of C_i. For a given point $x \in C_i$, by definition, $\mathbf{PR}^{range}(x) = r_i$ and $\mathbf{PR}^{range}(D) = \min\{r_1, r_2, \ldots, r_k\}$. Therefore,

$$r_i \geq \min\{r_1, r_2, \ldots, r_k\} \ , \ i = 1, 2, \ldots, k$$

□

Lemma 3.5 $\forall x \in D : \mathbf{PR}^{rec}_{\mathcal{A}[c]}(x) \geq \mathbf{PR}^{rec}_{\mathcal{A}[c]}(D)$, *for all dataset* $D \subset \mathbb{R}$.

Proof. Consider a dataset $D \subset \mathbb{R}$ and a reconstructed set $R \subset \mathbb{R}$. Let x_a be any given point in D and r_a its reconstructed counterpart in R. By definition, $\mathbf{PR}^{rec}_{\mathcal{A}[c]}(x_a)$ is $|x_a - r_a|$ and $\mathbf{PR}^{rec}_{\mathcal{A}[c]}(D) = \min\{|\mathbf{x}_i - \mathbf{r}_i| : \mathbf{x}_i \in D, \mathbf{r}_i \in R, 1 \leq i \leq |D|\}$. Therefore,

$$|x_a - r_a| \geq \min\{|\mathbf{x}_i - \mathbf{r}_i| : \mathbf{x}_i \in D, \mathbf{r}_i \in R\}$$

□

Lemma 3.6 $\forall x \in D : \mathbf{PR}^{BK}_{\mathcal{A}[c]}(x) \geq \mathbf{PR}^{BK}_{\mathcal{A}[c]}(D)$, *for all dataset* $D \subset \mathbb{R}$.

Proof. Consider a cluster map $\mathcal{C}$ from D, with clusters $C_1, C_2, \ldots C_k$. Let X_a be a random variable modeling data points $x_a \in C_a$, for any $C_a \in \mathcal{C}$. By definition, $\mathbf{PR}^{BK}_{\mathcal{A}[c]}(x_a)$ is $2^{h(X_a)}$ and $\mathbf{PR}^{BK}_{\mathcal{A}[c]}(D) = \min\{2^{h(X_i)}\}$, $i = 1, 2, \ldots, k$. Therefore,

$$\mathbf{PR}^{BK}_{\mathcal{A}[c]}(x_a) = 2^{h(X_a)} \geq \min\{2^{h(X_i)}\}, \ i = 1, 2, \ldots, a, \ldots k$$

□

Theorem 3.3 $\forall x \in D : \mathbf{PR}^{DCP}_{\mathcal{A}[c]}(x) \geq \mathbf{PR}^{DCP}_{\mathcal{A}[c]}(D)$, *for all dataset* $D \subset \mathbb{R}$ *and any measure* $m \in \{rec, range, BK\}$.

Proof. By lemmas 3.4, 3.5 and 3.6, substituting a specific measure $m \in \{rec, range, BK\}$. □

Lemma 3.7 *Given a sensitive set $D \subset \mathbb{R}$, a cluster map $\mathcal{C}$, the privacy $\mathbf{PR}^{range}(D) = \rho$ means that with no further information ρ is the size of the smallest interval of where any $x \in D$ can be located.*

Proof. Assume that a cluster map defines the boundaries of each cluster. By definition 3.1, $\mathbf{PR}^{range}(x) = \max C_i - \min C_i$, for $x \in C_i$. Since $\mathbf{PR}^{range}(D) = \min\{\mathbf{PR}^{range}(x)\} = \rho$, then ρ is the size of the smallest interval $(\min C_i, \max C_i)$ considering each point x with its respective cluster $C_i \in \mathcal{C}$. □

Lemma 3.8 *Given a sensitive set $D \subset \mathbb{R}$, a reconstructed set $R \subseteq \mathbb{R}$, the privacy $\mathbf{PR}^{rec}(D) = \rho$ means that ρ is the lower bound of the reconstruction error, i.e.*

$$|x_i - r_i| \geq \rho$$

for all points $x_i \in D$ and $r_i \in R$.

Proof. By definition 3.2, $\mathbf{PR}^{rec}(D) = \min\{|\ x_i - r_i\ |\} = \rho$. Thus $|\ x_i - r_i\ | \geq \min\{|\ x_i - r_i\ |\} = \rho$, for all $x_i \in D$ and $r_i \in R$. □

Lemma 3.9 *Given a sensitive set $D \subset \mathbb{R}$, a cluster map $\mathcal{C}$, $\mathbf{PR}^{BK}(D) = \rho$ means that ρ is the size of the smallest interval of where $x \in D$ can be located.*

Proof. By definition (3.3) and equation (3.8), $\mathbf{PR}^{BK}(x) = 2^{h(X_i)}$ with random variable X_i modeling the points in cluster $C_i \in \mathcal{C}$ with probability density function $f_{X_i}(x)$. Assuming no knowledge about the distribution of points in C_i, f_{X_i} is the uniform distribution in the interval defined by the borders of cluster C_i. Thus, $X_i \sim U(\min C_i, \max C_i)$. Let $\max C_i - \min C_i = \rho_i$, the size of the interval of values in cluster C_i. The probability density function for any point in the cluster C_i with uniform distribution is $f_{X_i}(x) = 1/\rho_i$. The entropy of X_i is $h(X_i) = \log_2(\rho_i)$ and, therefore:

$$2^{h(X_i)} = 2^{\log_2 \rho_i} = \rho_i = \max C_i - \min C_i$$

with equation (3.9), $\mathbf{PR}^{BK}(D) = \min\{\mathbf{PR}^{BK}(x)\} = \min\{\rho_i\} = \rho$. Therefore,

$$\max C_i - \min C_i \ \geq \ \min\{\max C_i - \min C_i\} = \rho$$

for all $C_i \in \mathcal{C}$. □

Theorem 3.4 *Given a sensitive set D, a cluster map $\mathcal{C}$, $\mathbf{PR}^{DCP}(D) = \rho$ has a well-defined interpretation, with privacy measures $m \in \{range, BK, rec\}$, as the size of the smallest interval containing a given point of the dataset.*

Proof. This idea is demonstrated in lemmas 3.7, 3.8 and 3.9 □

Theorem 3.4 above shows that $\mathbf{PR}^{DCP}(D)$ represents intervals of values revealed by a cluster map. Larger values are better because it means more uncertainty on the exact values of a given point. Also, a domain expert could be alerted when a cluster map does not satisfy the local privacy constraints.

3.3.2 Privacy Measure for Time Series Mining

The time dimension in data describes how a process evolves through time [245]. *Amplitude* is the value of time series at a particular time point and can be compared to the data value in non-time series data. Amplitude in time series must be protected if it represents a measurement of a sensitive variable, such as sales volume or purchase history. Furthermore, *peaks* are extreme values assumed in the series and may indicate a sudden change of normal behavior, e.g., money flow problems. *Predictions* (or trends) are another aspect that may be considered sensitive since they allow the attacker to anticipate a given value in the future, with a given statistical confidence level. Predictions depend on the accuracy of the prediction model available to the attacker. Thus, predictions might represent a privacy breach if they are too accurate.

The essential information about all aspects of the time series is the amplitude, from which all other aspects can be derived (peaks, trends, and predictions). If a particular data point is not known or only known to lie in a given interval, all other aspects will have less accuracy than if the point was known with exact precision. Therefore, we focus on the amplitude, i.e., the raw value at a particular data point.

As discussed in Section 2.3.3, different measures have been proposed in the privacy-preserving data mining literature. The large majority of the measure is not defined for time series. Wagner et al. [216] surveyed more than 80 privacy metrics and found that only three measures are proposed

for time series. Moreover, the time series metrics found on Wagner's survey are domain-specific for smart meters and location-based services. In these domains the focus is on data point re-identification.

Information theoretical approaches for privacy focus on how confident the attacker can be given the entropy of a random variable. In this study, we have chosen to propose an information theoretical-based measure for time series that quantifies the uncertainty an attacker has concerning the exact timestamp and amplitude of an original data point.

The privacy measure for time series proposed in this study is an extension of the entropy-based measure called *bounded knowledge*, introduced by Agrawal and Aggarwal [4] and discussed in Section 3.3.1. The privacy level of a given point in the amplitude dimension of a time series is computed by modeling it from the attacker's point of view. Therefore, an arbitrary point x_t of the original time series T is modeled as a random variable X, which allows for the application of the above privacy definition. The probability density function $f_X(x)$ may be used to model the attacker's knowledge about the point x_t. If the attacker does not know how X is distributed, the uniform distribution is used, which gives us $\mathbf{PR}^{BK}(X) = 2^{log_2(a)} = a$, the size of the interval from where X is drawn. Nevertheless, if a better model for x_t is known, it can naturally be incorporated into the privacy level. For example, when the time series has a reasonable degree of predictability, the privacy level can be computed using the correct model.

We can apply the same idea to the time dimension perspective. For a given point x_t, its timestamp t can be modeled as a random variable V. The probability density function $f_V(t)$ may be used to model the knowledge the attacker has about the time point t when x_t occurred. Applying the idea of bounded knowledge, the size of the time interval a specific x_t occurred is $\mathbf{PR}^{BK}(V) = 2^{log_2(b)} = b$. Combined, $\mathbf{PR}^{BK}(X)$ and $\mathbf{PR}^{BK}(V)$ define a region in amplitude and time from where a given point x_t is drawn. The combined privacy level is defined as the area of this region.

An additional element is included in the extension of the model as this study accounts for the fact that parties may collude. The following definition gives the details.

Definition 3.5 (Time Bounded Knowledge) *The privacy level of a given point x in a time series T, with a random variable X modeling the amplitude*

of x, and a random variable V modeling the time stamp where x might occur, under algorithm $\mathcal{A}$ in the presence of c colluders, is given by:

$$\mathbf{PR}^{TBK}_{\mathcal{A}[c]}(x) = \mathbf{PR}^{BK}_{\mathcal{A}[c]}(X)\mathbf{PR}^{BK}_{\mathcal{A}[c]}(V) \tag{3.11}$$

$$= 2^{h(X)}2^{h(V)} \tag{3.12}$$

where $h(\cdot)$ is the differential entropy.

□

Now, we extended the previous definition to a complete time series.

Definition 3.6 *Given an algorithm $\mathcal{A}$, the privacy level of a time series T in presence of c colluders running $\mathcal{A}$ is given in terms of Def. 3.5:*

$$\mathbf{PR}^{TBK}_{\mathcal{A}[c]}(T) = \min\{\mathbf{PR}^{TBK}_{\mathcal{A}[i]}(x) \mid x \in T\} \tag{3.13}$$

with $c \geq 0$, and $i = 1, 2, \ldots, c$. □

Finally, using the previous measure, it is possible to measure the privacy level of a time series mining algorithm, assuming that a given point in the time series is modeled as a random variable.

Properties Analysis of Time Bounded Knowledge Measure

In the following, we state that $\mathbf{PR}^{TBK}_{\mathcal{A}[c]}(T)$ is non-negative (Theorem 3.5) and has the following properties:

P1 (inference and collusion): theorem 3.6;

P2 (point level privacy): theorem 3.7;

P3 (interpretation): theorem 3.8.

Theorem 3.5 $\mathbf{PR}^{TBK}_{\mathcal{A}[c]}(T) \geq 0$, *for all dataset T.*

Proof. By definition 3.6, $\mathbf{PR}^{TBK}(T)$ is defined as $\min\{2^{h(X)}2^{h(V)}\}$. Since $h(\cdot) \in \mathbb{R}$ and $2^a \geq 0$ for all $a \in \mathbb{R}$, it follows that $\min\{2^{h(X)}2^{h(V)}\} \geq 0$. □

Theorem 3.6 *Given an algorithm $\mathcal{A}$, a time series T, and $c > 1$ (presence of collusion groups), if there is a collusion scenario decreasing the privacy level of dataset T, then $\mathbf{PR}^{TBK}_{\mathcal{A}[1]}(T) \geq \mathbf{PR}^{TBK}_{\mathcal{A}[c]}(T)$.*

Proof. Let $a = \mathbf{PR}^{TBK}_{\mathcal{A}[1]}(T)$ be the privacy level of time series T with algorithm $\mathcal{A}$ with no collusion (i.e., $c = 1$), and $b = \mathbf{PR}^{TBK}_{\mathcal{A}[c]}(T)$ be the privacy level of time series T with algorithm $\mathcal{A}$ in a collusion scenario with $c > 1$ malicious peers. By definition ${\mathbf{PR}_{\mathcal{A}[c]}}^{TBK}(T)$ is defined as the smallest privacy level considering all collusion scenarios. Thus, $\mathbf{PR}^{TBK}_{\mathcal{A}[c]}(T) = \min\{a, b\}$. Therefore, if the collusion group decreases the privacy level of the $c = 1$ scenario, then $a \geq \min\{a, b\}$. □

Theorem 3.7 *Given a time series T, $\mathbf{PR}^{TBK}_{\mathcal{A}[c]}(x_t) \geq \mathbf{PR}^{TBK}_{\mathcal{A}[c]}(T),\ x_t \in T$.*

Proof. Let X_t be a random variable modeling amplitude datapoint $x_t \in T$. Similarly, consider V_t a random variable for the position of x_t in T. By definition, $\mathbf{PR}^{TBK}(T) = \min\{\mathbf{PR}^{TBK}(x_t) : x_t \in T\} = \min\{2^{h(X_t)}2^{h(V_t)} : x_t \in T\}$. For a given point $x_a \in T$:

$$\mathbf{PR}^{TBK}(x_a) = 2^{h(X_a)}2^{h(V_a)} \geq \min\{2^{h(X_t)}2^{h(V_t)}\}, t = 1, 2, \ldots, a, \ldots, |T|$$

□

Theorem 3.8 *Given a sensitive time series T, $\mathbf{PR}^{TBK}(T) = \rho$ has a well-defined interpretation, as the area, in time and amplitude, where x can be located.*

Proof. By definition 3.6 and equation (3.12), $\mathbf{PR}^{TBK}(x) = 2^{h(X)}2^{h(V)}$ with random variable X modeling the amplitude (the value of x) with probability density function $f_X(x)$ and random variable V modeling the time point t when x occurred. Let $S_i \sqsubseteq T$ be a subsequence of T with $x \in S_i$. Assuming no knowledge about the distribution of points in S_i, f_{X_i} and f_{V_i} are density functions of uniform distributions in the intervals of size a and b respectively. Thus, $X_i \sim U(\min\{S_i\}, \max\{S_i\})$ and $V_i \sim U(0, |S_i|)$. Let a be the interval of values of S_i, i.e. $a_i = \mid \max\{S_i\} - \min\{S_i\} \mid$, and b_i the interval of time stamps in S_i, i.e. the size of S_i. The probability density function for any value in S_i with uniform distribution is $f_{X_i}(x) = 1/a_i$. Similarly, $f_{V_i}(t) = 1/b_i$ for the time stamp t of x. The entropy of X_i is $h(X_i) = \log_2(a_i)$ and V_i is $h(V_i) = \log_2(b_i)$, therefore:

$$2^{h(X_i)}2^{h(V_i)} = 2^{\log_2 a_i}2^{\log_2 b_i} = a_i b_i$$

with equation (3.13), $\mathbf{PR}^{TBK}(T) = \min\{\mathbf{PR}^{TBK}(x)\} = \min\{a_i b_i\} = \rho$. Therefore, ρ is area of the smallest rectangle formed by amplitude and subsequence size in T where any point x can be located. □

Theorem 3.8 above shows that $\mathbf{PR}^{TBK}(T)$ represent intervals of values in time and amplitude revealed by the mining process. Larger intervals are better because it means more uncertainty on the exact values of a given point in time or amplitude. Also, a domain expert could be alerted when a time series mining algorithm does not satisfy the local privacy constraints.

3.4 Privacy Analysis with Inference and Collusion Attack Scenarios

In the next sections, selected algorithms for distributed data clustering are briefly reviewed, and their privacy properties are analyzed in light of our privacy definitions. The algorithm we analyze in this section were chosen because they are typical instances of each approach. VC-k-means [213] and EC-kmeans [161] are good representative of the SMC approach. Similarly, DDCGM [149] and ITDDC [184] are typical examples of distributed model aggregation. An interesting new approach is PP-AAC [108] which uses cluster description similar to model aggregation but uses a secure sum protocol to ensure data privacy.

To apply the privacy framework introduced in the previous section, we need to identify the parties involved in the mining section of a given algorithm, the information held by each party, and the information exchanged among them. From this point on, we can analyze how each party may use its local information, combined with information it receives during the protocol, to develop an inference attack against other parties. Next, we analyze how collusion groups may improve inference attacks. In general, we want to answer the following questions:

1. Which parties are involved? Which information does each party hold? Moreover, which information each party sends and receives?

2. What can be reconstructed from the information held by a single insider attacker?

3. Which information each collusion group holds?

Algorithm 3.1 Secure Multi-Party k-Means

Input: P parties, k clusters, n points in dataset X.
Output: vector of means $\{\mu_i\}_{i=1}^{k}$
Method:

```
1: initialize centroids μ_ij with random value;
2: initialize clusters C_i ← ∅, i = 1, ... k;
3: repeat
4:     for all x ∈ X do
5:         for all parties j = 1 to P do
6:             Securely compute the distance vector y⃗ from point x ∈ X to {μ_i}
                   with all parties;
7:             Securely decide to put x in the nearest cluster C_i with all parties
                   (with Alg. 3.2);
8:         end for
9:     end for
10:    Update μ_i as mean of points in cluster C_i, i = 1 to k;
11: until threshold is reached
```

4. What can be reconstructed from information held by a collusion group?

5. Which privacy measure is the most appropriate to quantify privacy loss in each case, depending on the information available to the attacker?

3.4.1 Secure Multi-Party k-Means Clustering

Vaidya and Clifton [213] proposed an extension of the classic k-means algorithm to the distributed setting, using cryptographic protocols to achieve privacy (VC-k-means, cf. Alg. 3.1). Data is assumed to be vertically partitioned in a multi-party scenario with three non-colluding parties. The solution is based on a secure protocol to find the closest cluster for any given point, and it also uses secure permutation and secure comparison. They offer proof that each peer only learns its part of each cluster centroid and the cluster assignment of all points at each iteration.

The fundamental problem faced in each iteration is securely assigning each point to its nearest cluster. This problem is non-trivial since each site owns a part of each tuple, which must remain private. It is solved by applying the following algorithm (cf. Alg. 3.2) for each point $\vec{x}$ (assuming $P \geq 3$ sites).

Let x_j and μ_{ij} be the portions of $\vec{x}$ and the i^{th} centroid at the j^{th} site, respectively. Let $\vec{y_j}$ be the length K vector where y_{ij} is the distance between x_j and μ_{ij}. The problem boils down to securely computing

Algorithm 3.2 SMC Closest Centroid

Input: P parties, each of which with a length k vector $\vec{y}$ of distances. Three parties are labeled L_1, L_2 and L_P
Output: the closest centroid
Method:

1: L_1 generates P random vectors v_i summing to 0;
2: L_1 generates a random permutation π over k elements;
3: **for all** $i = 2$ to P **do**
4: $\quad T_i = \pi(y_i + v_i)$; // *each party adds a distance and permutes*
5: **end for**
6: L_1 computes $T_1 = \pi(y_1 + v_1)$;
7: **for all** $i = 1, 3$ to $P - 1$ **do**
8: $\quad L_i$ send T_i to L_P;
9: **end for**
10: L_P computes $\vec{y} = T_1 + \sum_{i=3}^{p} T_i$;
11: **for all** $i = 1$ to k **do**
12: $\quad$ securely find the index l of the minimal distance in the vector $\vec{y}$;
13: **end for**
14: L_P sends l to L_1;
15: L_1 broadcasts the index l, the closest centroid, to all parties;

$argmin_{i=1}^{k}\{\sum_{j=1}^{P} y_{ij}\}$, i.e. the index of the smallest value in the distance vector $\vec{y}$. Site 1 computes random vectors (length k) $\vec{v}_1, \ldots, \vec{v}_P$ whose sum is zero and, π, a random permutation of $\{1, \ldots, k\}$. For each $2 \leq j \leq p$, site 1 then engages in a secure algorithm allowing site L_j to compute $\pi(\vec{v}_j + \vec{y}_j)$. At the end of this algorithm, site 1 does not know anything new and site 2 does not know π or $\vec{v}_j$. This algorithm uses homomorphic encryption to achieve security. Next, sites $1, 3, \ldots, P-1$ send $\pi(\vec{v}_j + \vec{y}_j)$ to site P. Site P sums these vectors with its own (note that site p does not know the vector at site 2). Then, site P and site 2 use SMC to securely determine the index ℓ of the minimum entry of vector $\sum_{j=1}^{P} \pi(\vec{v}_j + \vec{y}_j)$. After that, site 2 knows the minimum distance but not to which centroid it corresponds (due to the permutation known only to site 1). Finally, site 2 sends ℓ to site 1, which then broadcasts $\pi^{-1}(\ell)$ to all sites *i.e.* the closest centroid. They also presented a modified version of the protocol to handle collusion with an increased communication cost.

Inference and Collusion Attacks against VC-k-Means

Let L_1, L_2 and L_P, be three trusted parties in the VC-k-means protocol under study. Let L_i be any other non-trusted party in the mining group. The aim is to ascertain which information each party knows during the protocol

execution, examined in the following.

Single Insider Attacks. A given party L_i knows: (i) $\vec{\mu}_i$, a share of the centroid; (ii) y_{ij}, the distance from the cluster centroid μ_{ij} to the view of point x_i; (iii) and a random vector $\vec{v}_i$. The trusted parties know more than that, as follows.

L_1 is the party that starts the protocol. It does know: (i) a partial view of the cluster centroids, denoted $\vec{\mu}_1$; (ii) the cluster assignment for each data point x; (iii) a random vector $\vec{v}$; and (iv) a permutation π of 1 to k, used to preserve the privacy of information in the SMC protocol. L_2 knows $\vec{T}_2 = \pi(\vec{v}_2 + \vec{y}_2)$, the permuted sum of $\vec{v}_2$ with $\vec{y}_2$. This information is hidden from the other parties, and is used only in the clustering step in a protocol with L_P.

L_P knows its share of the centroid $\vec{\mu}_P$, and $\vec{T}_i = \pi(\vec{v}_i + \vec{y}_i)$, with $i = 1, 3, 4, \ldots, P$, the permuted sum of $\vec{v}_i$ with $\vec{y}_i$ of each party but L_2. Moreover, L_P knows the combined sum of $\vec{T}_i$ from all parties but L_2, i.e. $\vec{y} = \vec{T}_1 + \sum_{i=3}^{P} \vec{T}_i$

L_1 is the party holding the most valuable information, which can be used to reconstruct sensitive data: the random vector $\vec{v}$ and the permutation π. However, without the permuted sum of distances $\vec{y}_i$ from other parties ($i = 1, 3, 4, \ldots, P$), L_1 will not learn anything because it cannot reconstruct data points from other parties. Similarly, L_2 and L_P will not learn anything from the information they hold alone. In other words, each party will learn anything other than the results. With no further information beyond the cluster map (the output), we can only apply $\mathbf{PR}^{range}(D)$.

Lemma 3.10 *Let D be a dataset distributed over a network of peers. When there are only single insider attacks, algorithm VC-k-means produces a cluster map $\mathcal{C}$ of D with a privacy level given by:*

$$\mathbf{PR}^{DCP}_{VCkMeans[1]}(D) = \mathbf{PR}^{range}_{VCkMeans[1]}(D) = \min\{\max C_i - \min C_i\}$$

with $C_i \in \mathcal{C}$ and the privacy level, $\mathbf{PR}$, as defined in Section 3.3.1 (cf. Def. 3.4).

Proof. Any insider attacker working solo can only learn what is disclosed by the cluster map itself, namely, that each point $x \in C_i$ ranges in the interval $(\min C_i, \max C_i)$, with $C_i \in \mathcal{C}$. □

The result with no collusion is unsurprising, given that this protocol was designed to work with three trusted parties – a strong assumption. In the subsequent attacks, this assumption is dropped as malicious insiders are introduced into the scenario.

Collusion Attacks. Now, we analyze the privacy level of VC-k-means when $c \geq 2$ malicious insiders form a collusion group.

Attack with Collusion of Insiders L_1 and L_P. When L_1 and L_P collude, they may learn data from other parties with arbitrarily high precision. Together, L_1 and L_P hold information on the permuted sum of all parties except for L_2. Moreover, they hold information on the permutation π and the random vector $\vec{v}$. Therefore, this collusion group may compute the vector $\vec{y}$ using the inverse of permutation π and subtracting the random noise $\vec{v}$ from Y_i, as indicated below:

$$\vec{y_i} = \pi^{-1}(\vec{T_i}) - \vec{v_i} \tag{3.14}$$

with $i = 1, 3, 4, \ldots, P$

The vector $\vec{y_i}$ represents the distance between a given point x and the cluster centroid i with mean μ_i. Once the attacker has the true distance between all clusters centroids and a given point, every point can be located with an arbitrary error. Since L_1 and L_P hold information from all parties but L_2, it is enough to reconstruct data from all parties in the mining group but L_2. With a given reconstruction method, we can apply $\mathbf{PR}^{rec}(D)$.

Lemma 3.11 *Given a collusion group with L_1 and L_P, the privacy level of VC-k-means algorithm, is:*

$$\mathbf{PR}^{DCP}_{VCkMeans[2]}(D) = \mathbf{PR}^{rec}_{VCkMeans[2]}(D) = \min\{|x - r| : x \in D, r \in R\} \approx 0$$

where $D \subset \mathbb{R}$ is the original dataset and $R \subset \mathbb{R}$ is a reconstructed dataset built using Eq. 3.14.

Proof. When the collusion group includes L_1 and L_P, they know everything needed to use Eq. 3.14 and generate the dataset of reconstructed points R

with arbitrary precision. Therefore, the error in this attack is down to the minimum error between the original and the reconstructed data. □

For completeness purposes, other potential collusion groups are discussed in the following.

Attack with Collusion of Insiders L_1 and L_2. In this collusion group, there is not much new, useful information. L_2 adds its own permuted sum T_2 and nothing more. Consequently, the collusion will learn nothing other than the results.

Attack with Collusion of Insiders L_2 and L_P. In this attack scenario, the collusion holds the permuted sum from all parties but L_1. However, without information from L_1, the colluders cannot remove the random noise from the sum. Hence, nothing can be learned by the collusion group beyond the results.

Attack with Collusion of Insiders L_i, $3 \leq i \leq P$. Any other collusion group that does not include L_1 and L_P will not know the permuted sum from other parties and will not remove the random noise from the sum. Therefore, this collusion group will learn nothing.

Lemma 3.12 *The single and collusion scenarios in VC-k-means, with c colluders, are related as follows:*

$$\mathbf{PR}^{DCP}_{VCkMeans[1]} \geq \mathbf{PR}^{DCP}_{VCkMeans[c]} \approx 0$$

with $c \geq 2$.

Proof. By Lemma 3.11 the collusion group of size $c = 2$ with L_1 and L_P reach the lowest level of privacy since they have sufficient information to reconstruct the data with arbitrary precision. □

Together, these two lemmas reinforce the need for L_1 and L_P be trusted parties. If L_1 and L_P are trusted, the secure multi-party computation provides a high level of privacy. Otherwise, a collusion group may reconstruct sensitive data to arbitrary precision.

Outsider attack. An outsider needs to get all the information held by the parties, including the permutation π, the random vector $\vec{v}$, and the permuted sum from each party. Since messages exchanged in this protocol are encrypted, an outsider needs to secure assistance from L_1 and L_P to be successful. Although this is a theoretical possibility, an outsider is superfluous if L_1 and L_P are malicious. Without insider collaboration, an outsider can not reconstruct any information from the messages it manages to intercept. The outsider with zero insider collaboration does learn nothing about the sensitive data during the mining session. We expressed this result as:

$$\mathbf{PR}^{DCP}_{VCkMeans[0]} = \infty$$

3.4.2 Elliptic Curves-Based k-Means

Patel and colleagues [161] proposes a privacy-preserving distributed k-means algorithm based on elliptic curves (EC-kmeans). They assume no trusted party and use elliptic curves to achieve low overhead cryptography. The authors present no analysis of inference attack or collusion.

Single Insider Attack. Each peer knows its centroids, its cluster boundaries, the encrypted version of the global centroids, and the number of points on each global cluster. Without collusion, a malicious party does not know the boundaries of clusters from other parties, i.e., privacy is not compromised. Therefore, we assign the highest privacy degree this scenario:

$$\mathbf{PR}^{DCP}_{ECkmeans[1]}(D) = \mathbf{PR}^{range}_{ECkmeans[1]}(D) = \infty \tag{3.15}$$

Collusion Attacks. The initiator knows the information necessary to decrypt data in the mining session. Therefore, a collusion group with the initiator and any party L_i can learn about the centroids and number of points in each cluster on the party L_{i-1}. With the centroids, cluster boundaries of dataset D at L could be estimated and

$$\mathbf{PR}^{DCP}_{ECkmeans[2]}(D) = \mathbf{PR}^{range}_{ECkmeans[2]}(D) = \min\{\max C_i - \min C_i\} \tag{3.16}$$

Outsider Attacks. Any attacker that is not part of the mining session will not possess the information necessary to decrypt data. Recall that this

information is held only by the initiator. Therefore, outsiders have less information than single insiders. In this case, we indicate the impossibility to know even the range of any cluster as:

$$\mathbf{PR}^{DCP}_{ECkmeans[0]}(D) = \mathbf{PR}^{range}_{ECkmeans[1]}(D) = \infty \tag{3.17}$$

3.4.3 Generative Models for Privacy-Preserving Clustering

Merugu and Ghosh [149] present an algorithm for distributed clustering and classification (DDCGM). DDCGM still is a good representative of the distributed model aggregation approach, which remains a very active area of interest [12, 22, 91, 92, 136, 169, 176, 241].

DDCGM algorithm outputs an approximate model $\hat{\lambda}_c$ of a true global model λ_c from a predefined fixed family of models F, , *e.g.* multivariate 10-component Gaussian mixtures. This model approximates the underlying probability model that generated the global dataset D. Merugu's generative models first computes local models λ_i, from which the average global model $\bar{\lambda}$ is generated. The average model $\bar{\lambda}$ is given by

$$p_{\bar{\lambda}}(x) = \sum_{i=1}^{n} \nu_i p_{\lambda_i}(x) \tag{3.18}$$

where $p_\lambda(x)$ is the probability density function of a given model λ. Although the average model $\bar{\lambda}$ is a good approximation of the true model λ_c, it might not be very interpretable. Thus, the algorithm finds the model in a given family F, which is the closest model to the average model $\bar{\lambda}$ in terms of KL-divergence. The algorithm uses $\bar{\lambda}$ to generate a sample dataset $\bar{D}$ with a Markov Chain Monte Carlo method. This dataset $\bar{D}$ is used with an Expectation-Maximization (EM) approach[6] to find a good approximation $\hat{\lambda}_c$ of the true (and unknown) global model λ_c, i.e., $\hat{\lambda}_c$ maximizes the log-Likelihood of $\bar{D}$. The model $\hat{\lambda}_c$ is used as a cluster map, and every point x is assigned to the cluster that maximizes probability at x. The pseudocode for this idea is shown in Algorithm 3.3.

[6]Expectation-Maximization (EM) is a probabilistic approach to learning with missing values. In this case, the missing value is the cluster assignment of each data point.

Algorithm 3.3 Distributed Data Clustering with Generative Models

Input: Set of models $\{\lambda_i\}_{i=1}^{P}$ with weights $\{\nu_i\}_{i=1}^{P}$, summing to 1, and mixture model family F.
Output: Estimated global model $\hat{\lambda}_c$
Method:
1: Obtain mean model $\bar{\lambda}$ such that

$$p_{\bar{\lambda}}(x) = \sum_{i=1}^{P} \nu_i p_{\lambda_i}(x)$$

2: Generate $\bar{D} = \{x_j\}_{j=1}^{m}$ from mean model $\bar{\lambda}$ using Markov Chain Monte Carlo sampling.
3: Apply EM algorithm to obtain the estimated global model $\hat{\lambda}_c$, such that

$$\hat{\lambda}_c = \arg\max_{\lambda_c \in F} L(\bar{D}, \lambda_c) = \arg\max_{\lambda_c \in F} \frac{1}{m} \sum_{j=1}^{m} \log(p_{\lambda_c}(x_j))$$

where $L(\bar{D}, \lambda_c)$ is the average log-likelihood of $\bar{D}$ with respect to λ_c.

Inference and Collusion Attacks against Generative Models

DDCGM does not exchange any points, only models [149]. It is important to remember that generative models, as proposed by Merugu and Ghosh [149], do give users a mechanism to enforce any particular level of privacy for local datasets – namely, the number of models in the mixture. Therefore, a local site chooses the number of models in the local model and ensures that the local dataset has the desired level of likelihood of being generated by the global model (cf. Example 2.9). According to the proposed approach, a low likelihood means high privacy and vice-versa. Observe that the number of models is the maximum number of clusters in the final model, typically many orders of magnitude smaller than the dataset size.

Single Insider Attacks. In DDCGM, a central entity receives local generative models and combines them into an average generative model. Let us assume that this central entity is a malicious peer. This entity has every individual generative model from each party in the mining group. With no further information beyond the output, we can apply $\mathbf{PR}^{range}(D)$.

Lemma 3.13 *Let $p_\lambda(x)$ be a mixture model with k elements. The privacy*

level provided by generative models using $p_\lambda(x)$ and with no collusion is

$$\mathbf{PR}^{DCP}_{DDCGM[1]}(D) = \mathbf{PR}^{range}_{DDCGM[1]}(D) = \min\{\max C_i - \min C_i\}$$

where $C_i \in \mathcal{C}$, and $\mathcal{C}$ is a cluster map, according to the model $p_\lambda(x)$.

Proof. The central entity has information concerning every local model from each party. With this information, the central attacker can compute the min and max element in each dimension of every cluster from each party. Assume that clusters are modeled by Gaussians. Thus, a given cluster C_i is modeled with mean μ_i and variance σ_i^2. Therefore, the attacker can infer that the points in this cluster lies in the interval $(\mu_i - 3\sigma_i, \mu_i + 3\sigma_i)$,i.e. the interval has size $6\sigma_i$, with 99.7% confidence[7]. □

Each model, in the generative models approach, represents a probability density function of data points in a given cluster. Thus, $\mathbf{PR}^{BK}$ could also be used to measure the privacy provided by this approach.

For example, assuming each model is given by a Gaussian in a n dimensional data space with covariance matrix Σ_i for a given model, the entropy of each model is

$$h_i(x) = \ln\sqrt{(2\pi e)^n |\Sigma_i|}$$

where $|\Sigma_i|$ is the determinant of the covariance matrix of i^{th} model. Therefore,

$$\mathbf{PR}^{DCP}_{DDCGM[1]}(D) = \mathbf{PR}^{BK}_{DDCGM[1]}(D) = \min_i\{2^{h_i(x)}\} = \min_i\{2^{\ln\sqrt{(2\pi e)^n|\Sigma_i|}}\} \tag{3.19}$$

Parties in this scheme can control the precision of attack and, consequently, privacy by choosing the number of generative models used locally. More local models lead to a more accurate mixture model. Therefore, to maintain a desired amount of privacy, parties may want to set locally the max number of models to be used. The amount of privacy preservation can be controlled by the number of models used and the range of the cluster described by any given model. Therefore, any given site can refuse to partake in any given mining group when the number of generative models agreed by the mining group is not in agreement with a local privacy policy or if the

[7] In statistics, the 3σ *rule* for normal distributions says that 99.7% of the values drawn from a normal distribution are three standard deviations away from the mean.

local models are too narrow (a small range implies a low privacy level).

Collusion Attack. When using the generative models approach, a collusion group may learn which models are being utilized by an attacked site. The attack could be carried out by discarding the models owned by the colluders, thereby isolating the models produced by the attacked parties. In that case, if the central party, which aggregates local models, takes part in the collusion, the attack is reduced to a single aggregator attack as the aggregator site holds the local models from all parties.

Lemma 3.14 *Central insider attack and collusion attack against generative models produce the same privacy level.*

$$\mathbf{PR}^{DCP}_{DDCGM[1]}(D) = \mathbf{PR}^{BK}_{DDCGM[1]}(D) = \mathbf{PR}^{BK}_{DDCGM[c]}(D) \tag{3.20}$$

with $c \geq 2$.

Proof. If the collusion group manages to get the local model from an attacked site, the collusion attack reduces to the case where the central party is malicious. In that case, the attacker's problem is to find the boundaries of each cluster. Since the central site has more information than any site, a collusion attack will not produce a more precise attack than an insider attack perpetrated by the central site. Therefore, collusion and insider scenarios produce the same privacy level. □

If the central entity is a malicious party, the privacy of sensitive data may be at stake. On the other hand, if the central entity is trusted, the collusion group must garner the local models from the attacked parties. In an extreme case, a collusion group would involve all parties, excluding the aggregator and the attacked party.

Outsider Attack. Outsiders need to know the number of mixture models and the parameters describing each local model. Assuming an outsider does not have any parameter value, it cannot use the data it gets since generative models exchange models rather than data. For that reason, intercepting messages is useless unless the attacker gets the model parameters.

$$\mathbf{PR}^{DCP}_{DDCGM[0]}(D) = \infty \tag{3.21}$$

Now, let us assume that an outsider attacker gets the parameter values about the local model from a group of mining parties. In this case, the attack has the same initial information as an insider attacker has, with no improvement on the privacy bounds already discussed in the insider case.

3.4.4 Information Theoretical Approach to DDC

Shen and Li [184] proposed an information-theoretical approach to distributed clustering (ITDDC). They assume a peer-to-peer network where each node solves a local clustering problem and updates its neighbors. The clustering problem is to fit a discriminative model to cluster boundaries that maximize the mutual information between cluster labels and data points. With low communication, local clusters are formed based on global information spread through the network. The algorithm needs several rounds of iterations to converge.

Inference and Collusion Attacks against ITDDC

When it comes to privacy, the authors do not investigate how the algorithm would behave under inference attacks and do not investigate how much privacy this approach does provide. Therefore, we discuss the ITDDC privacy properties indirectly from the features of the algorithm as presented in its original paper.

Single Insider Attack. Each party in ITDDC knows a set of discriminative models defining the clusters boundaries of points on data sets and from all its direct neighbors. We can apply $\mathbf{PR}^{range}(D_j)$ to compute how much privacy is preserved at local dataset D_j for a given model. Each party estimates $\hat{p}_j(k|x)$, a class label distribution defined by a local discriminative model (e.g. logistic regression). The distribution of x in a given cluster is not disclosed. Thus, each point can only be located in the interval corresponding to its cluster boundaries. The privacy provided by ITDDC using $\hat{p}_j(k|x)$ and with no collusion is:

$$\mathbf{PR}^{DCP}_{ITDDC[1]}(D) = \mathbf{PR}^{range}_{ITDDC[1]}(D) = \min\{\max C_i - \min C_i\} \qquad (3.22)$$

where $\max C_i$ and $\min C_i$ are inferior and superior elements at the each cluster, according to the boundaries defined by model $\hat{p}_j(k|x)$.

Collusion attack. Local models are the only information being exchanged among the parties. Moreover, there is no special central entity holding extra information on data distribution at local datasets. Therefore, even if malicious parties collude against another party, they cannot improve on the single insider attack. Therefore,

$$\mathbf{PR}^{DCP}_{ITDDC[c]}(D) = \mathbf{PR}^{range}_{ITDDC[c]}(D) = \mathbf{PR}^{range}_{ITDDC[1]}(D) = \mathbf{PR}^{DCP}_{ITDDC[1]}(D) \tag{3.23}$$

with $c \geq 1$ colluding parties. More directly,

$$\mathbf{PR}^{DCP}_{ITDDC[c]}(D) = \mathbf{PR}^{DCP}_{ITDDC[1]}(D) \tag{3.24}$$

Outsider attack. Any outsider that listens in on the messages may acquire information about the discriminative models describing clusters' boundaries. However, if the attackers do not know which family of models were chosen to describe cluster boundaries, they will not locate said boundaries. In this case

$$\mathbf{PR}^{DCP}_{ITDDC[0]}(D) = \infty \tag{3.25}$$

On the other hand, if outsiders know which family of models to model cluster boundaries, the scenario becomes equivalent to a single insider attack.

3.4.5 Average Consensus-Based Clustering

Jia et al. [108] proposes a general framework to achieve privacy-preserving distributed clustering. The authors propose a distributed algorithm called privacy-preserving accelerated average consensus algorithm (PP-AAC). This algorithm computes summations of terms from different parties without a trusted third party. The protocol is iterative and is guaranteed to converge. PP-AAC is based on noise addition to the sum terms to avoid unintentional data leakage. The noise follows an exponential decay and tends to zero with the increasing number of iterations.

PP-AAC is used as a building block to implement clustering algorithms. The proposed framework consists of running a clustering algorithm locally to compute a local model and using PP-AAC to compute a global model by the sum of local parameters. The authors implemented k-means, fuzzy clustering, and Gaussian mixture algorithms following this approach.

PP-AAC assumes there are M parties organized in a graph known to all participants. Each peer holds a dataset with N_i data points and local sensitive information $S_{k,i}$ about each local cluster, with $k = 1, \ldots, K$ and $i = 1, \ldots, M$. $S_{k,i}$ stores the sum of attribute values, number of points, and covariance matrix. During the protocol execution, parties send a perturbed version of $S_{k,i}$ to other peers until convergence of the global values G_k, with $k = 1, \ldots, K$. The summation protocol is repeated at each step of the clustering algorithm. Moreover, each peer communicates only with its direct neighbors. When the algorithm ends, each local party knows the global cluster centroids μ_k, with $k = 1, \ldots, K$.

Inference and Collusion Attacks against PP-AAC

PP-AAC's approach is similar to model aggregation as it uses cluster description (centroids) and does not exchange any local data points. It is also similar to SMC, as it is an iterative algorithm that protects original data in a multi-party computation. In the following, we discuss attack scenarios against PP-AAC.

Single insider attack. A single party knows the local dataset, the number of data points N_i and local sensitive information S_k, i describing sum and number of points at local clusters, and the global cluster centroids μ_k. Additionally, the graph describing the connections among the parties is public. After the protocol, all parties also know the global information G describing the centroids of each cluster. No other information about the data is available to single peers, such as distance to the centroid or standard deviation of points in a given cluster. Therefore, single peers cannot reconstruct points from other peers due to a lack of information. We represent this as:

$$\mathbf{PR}^{DCP}_{PPAAC[1]}(D) = \infty \tag{3.26}$$

Collusion attack. According to the original paper [108], a group of malicious peers may be able to compute the centroids of a victim peer P_j. The attack is possible if the collusion group shares the information they exchanged with the victim P_j. In this case, the attackers can cancel out the noise and subtract each contribution from the global values. However, there is no global cluster map available to the malicious group. Furthermore, the

distances from each point to cluster k at P_j are never exchanged during the protocol. Therefore, local data from victim P_j cannot be reconstructed under collusion in $PP-AAC$. Therefore, we have:

$$\mathbf{PR}^{DCP}_{PPAAC[c]}(D) = \infty \tag{3.27}$$

Outsider attack. PP-AAC iteratively compute aggregation of local sensitive information using a noise-based distributed algorithm. An outsider working with no help from insiders, no collusion group, has access only to the perturbed information exchanged during the protocol's execution. Therefore,

$$\mathbf{PR}^{DCP}_{PPAAC[0]}(D) = \infty \tag{3.28}$$

3.5 Summary

This chapter introduced new privacy measures for specific data mining tasks (cf. Sec. 3.3), which are applied in the subsequent chapters. These new measures were proposed to overcome the limitations identified in the studied measures. Starting from a set of formal properties, it was shown that the new measures satisfy all required properties and, therefore, improve pragmatically over the previous ones. A summary of the properties of the new measures is presented in Table 3.2.

	$\mathbf{PR}^{DCP}(D)$	$\mathbf{PR}^{TBK}(T)$
Application	Clustering	Time Series
Non-negative	Theorem 3.1	Theorem 3.5
P1 (collusion)	Theorem 3.2	Theorem 3.6
P2 (point-level)	Theorem 3.3	Theorem 3.7
P3 (interpretation)	Theorem 3.4	Theorem 3.8

Table 3.2: *Summary of New Privacy measures for Distributed Data Clustering and Time Series Mining*

The new measures were applied to some representative privacy-preserving algorithms, and Table 3.3 presents an overview of the main findings from studied algorithms. The analysis above shows that collusion is indeed a

Approach	# of TTP	Outside attacks $\mathbf{PR}_{[0]}^{DCP}$	Single attacks $\mathbf{PR}_{[1]}^{DCP}$	Collusion attacks $\mathbf{PR}_{[c]}^{DCP}$
VC-kmeans [213]	3	∞	$\min\{\max C_i - \min C_i\}$	≈ 0, $c \geq 2$
EC-kmeans [161]	0	∞	∞	$\min\{\max C_i - \min C_i\}$, $c \geq 2$
DDCGM [149]	0	∞	$\min\{2^{\ln\sqrt{(2\pi e)^n \lvert\Sigma_i\rvert}}\}$	$\min\{2^{\ln\sqrt{(2\pi e)^n \lvert\Sigma_i\rvert}}\}$, $c \geq 2$
ITDDC [184]	0	∞	$\min\{\max C_i - \min C_i\}$	$\min\{\max C_i - \min C_i\}$, $c \geq 2$
PP-AAC [108]	0	∞	∞	∞, $c \geq 2$

Table 3.3: *Summary of privacy-preserving distributed data clustering algorithms. (TTP = Trusted Third Party)*

primary source of privacy breaches and that algorithms can be separated according to their vulnerability to collusion groups and the malicious behavior of a site with a unique role in the protocol, e.g., a central site or an aggregator, or a protocol initiator. VC-kmeans algorithm leaks information if the central site colludes, whereas EC-kmeans is secure and only discloses range information under a collusion attack that involves the initiator. DDCGM has a limited vulnerability to the central site but not to collusion; ITDDC does not use a central site and only discloses cluster ranges, irrespective of collusion. PP-AAC is an interesting case where the privacy level is high, even with collusion. However, PP-AAC trades off privacy for time since it needs several rounds to compute each step.

Some identified benefits from the new measures are the ability to indicate the vulnerabilities of selected algorithms to collusion in different attack scenarios and detect point-level privacy breaches. Moreover, the proposed measures provide an intuitive notion of privacy as the size of the interval from where a given random variable can be drawn. On the downside, it is necessary to assume a particular malicious view on the attacked dataset, which is quite hard to preview in some scenarios. Nonetheless, such analysis may result in an upper or lower bound on the privacy level, in a process that can be refined in subsequent analysis.

The subsequent chapters endeavor to investigate inference attacks and appropriate privacy measures for two specific data mining tasks: *clustering* and *pattern discovery*. In Chapter 4, the case of distributed data clustering is investigated. Then, in Chapter 5, we focus on the distributed pattern discovery in time series.

Chapter 4

Privacy-Preserving Distributed Data Clustering

> "Civilization is the progress toward a society of privacy."
>
> (Ayn Rand)

> "The Milky Way is nothing else but a mass of innumerable stars planted together in clusters."
>
> (Galileo Galilei)

The previous chapter discussed the limitations of current privacy-preserving measures when malicious peers are part of the mining group. This chapter further develops these initial ideas focusing on distributed data clustering (DDC). We start this chapter by analyzing the KDEC algorithm for DDC [118].

KDEC is a density-based clustering algorithm that can find arbitrary shape clusters with a few rounds of communication between the mining agents (cf. Sec. 4.2). It uses a kernel estimate density and does not assume any underlying probability model. Therefore, it can be used even when there is no information about the data distribution. Furthermore, the communication costs are reduced since only models describing datasets are exchanged among agents and not raw data.

The kernel density estimation-based approach is interesting because it promises privacy preservation as no original data point is ever sent over the network – only a dataset model is communicated. However, as we discuss later in this chapter (cf. Sec. 4.2.3), under certain circumstances, the density estimates may be used to reconstruct the original data with high accuracy.

It follows that sensitive data may have its privacy compromised even if no data point is transmitted off the original site.

This chapter investigates how to improve density-based DDC so that users can have control over the potential privacy leakage during a mining session. We first briefly review kernel-based density estimation and analyze the privacy level of KDEC employing privacy measures developed in the previous chapter. As a result, we propose KDEC-S, an algorithm for privacy-preserving distributed data clustering. KDEC-S uses a kernel density estimation based on a new kernel approximation function, which allows for a controlled level of ambiguity during the process of density estimation. As a direct result, KDEC-S enhances the protection provided by the density estimation-based approach. We also present a complete privacy analysis of the said algorithm, sourced from the perspective of inference attack scenarios, as defined in our privacy framework (cf. Ch. 3).

4.1 Clustering Distributed Sensitive Data

In this chapter, the distributed scenario defined in Section 2.3.3 is assumed, i.e. there is a set of sites $\{S_i\}_{i=1}^{P}$, which are called *local sites*. Each local site S_i has a *local dataset* D_i and a *local mining agent* L_i. Mining agents are interconnected to other mining agents forming a pure peer-to-peer architecture. Only the local mining agent L_i, residing at site S_i, has access to local dataset D_i.

Recall the definition of sensitive data cited in Chapter 3: *sensitive data* is any piece of data that an agent decides to keep hidden from other agents. Here, two privacy constraints are considered: (i) data about individuals should not be disclosed due to law imposition (e.g., medical data), strategic decision (e.g., business data), or personal decision (e.g., annual income); (ii) data about the data collector should not be disclosed as well. To illustrate the second constraint, consider a group of hospitals planning to share information about death during heart surgery. Likely, a particular hospital does not want to be indicated as the one with the highest death rate, even though the cooperation may represent an opportunity to improve medical research.

In the following, the problem of privacy-preserving distributed data clustering is presented formally.

Problem 4.1 (PP-DDC) *Let $\mathcal{A}(D)$ be a clustering algorithm that maps any dataset D to a cluster map $\mathcal{C} = \{C_k\} \subseteq 2^D$, whose elements are pairwise disjoint. $\mathcal{C}$ is called a* clustering *of D. Let $D = \{\mathbf{x}_i \mid i = 1, \ldots, N\} \subseteq \mathbb{R}^n$ be a set of objects. Let $\mathcal{L} = \{L_j|, j = 1, \ldots, P\}$ be a finite set of* mining agents. *Each agent L_j, residing on site S_j, has access to one dataset D_j of size N_j. We assume that $D = \bigcup_{j=1}^{P} D_j$. The PP-DCC problem is to find for $j = 1, \ldots, P$, a local cluster map $\mathcal{C}_j$ residing in the data space of L_j, such that:*

(i) $\mathcal{C}_j = \{C_k \cap D_j : C_k \in \mathcal{C}\}$ *(*correctness requirement*);*

*(ii) Time and communications costs are minimized (*efficiency requirement*);*

*(iii) At the end of the computation, $\mathbf{PR}_{\mathcal{A}}$ (D_j) is not lower than a user given local privacy threshold. (*privacy requirement*).*

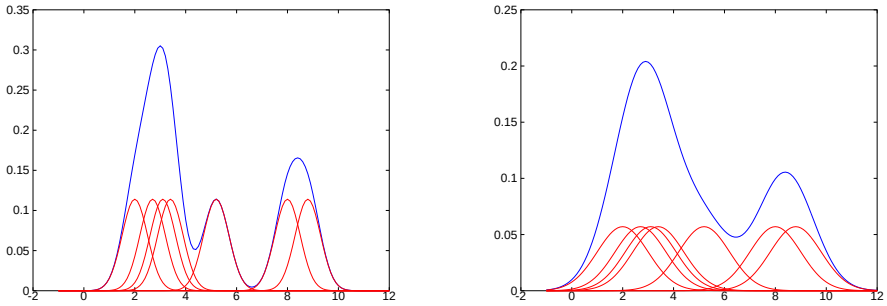
Notice that the data distribution across sites is assumed to be homogeneous, as discussed in Chapter 2. Homogeneity impacts the design of a solution because it assumes that all sites know the attributes describing the data objects. The centralized solution to the DDC problem is to collect all the distributed datasets D_j into one centralized repository where the clustering of their union is computed and transmitted back to the sites. This approach, however, does meet neither communication requirements nor privacy preservation requirements.

4.2 KDEC Algorithm

This section discusses KDEC, a density-based approach to distributed clustering that is the starting point to our solution. The privacy properties of KDEC in different attack scenarios are then extensively analyzed. After this privacy analysis, a new privacy-preserving scheme, called KDEC-S, is proposed.

4.2.1 Algorithm Overview

Density-based clustering reduces the search for clusters to the search for dense regions in an n-dimensional data space. The first step is to estimate a probability density function from which the given dataset is assumed to have



(a) $h = 0.5$ (b) $h = 1$

Figure 4.1: *Density estimates using Gaussian kernels. Each kernel is centered at a different point. The window width h is set to 0.5 in (a) and 1.0 in (b).*

arisen. An important family of methods for density estimation is known as *kernel estimator* [94], which does not make any assumptions on underlying generative models, computing the estimates directly from the data instances.

Clusters, in density-based clustering, are defined by local maxima in the density surface. Through hill-climbing, each point is assigned to one local maximum, and all points connected to the same local maxima receive the same cluster label [102].

Let $D = \{\mathbf{x}_i \mid i = 1, \ldots, N\} \subset \mathbb{R}^n$ represent a set of data objects. Let $K : \mathbb{R} \to \mathbb{R}$ be a non-negative, non-increasing function on $\mathbb{R}$ with finite integral over $\mathbb{R}$. Let $d : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}_+ \cup \{0\}$ be a distance function. Let $h \in \mathbb{R}$ be a scaling parameter called *window width*. A *kernel-based density estimate* $\hat{\varphi}_{K,h}[D] : \mathbb{R}^n \to \mathbb{R}_+ \cup \{0\}$ is defined as follows:

$$\hat{\varphi}_{K,h}[D](\mathbf{x}) = \frac{1}{N} \sum_{i=1}^{N} K\left(\frac{d(\mathbf{x}, \mathbf{x}_i)}{h}\right) \tag{4.1}$$

Equation (4.1) defines the estimate at $\mathbf{x} \in \mathbb{R}^n$ as a weighted sum of scaled distances of all data points from the neighborhood of $\mathbf{x}$. An example of kernel function used in density estimation is the restriction of the Gaussian function to $\mathbb{R}_+ \cup \{0\}$. Figure 4.1 shows two examples of density estimates computed with Gaussian kernels. With $h = 0.5$ the density estimate is more bumpy and with $h = 1$ the density is smoother.

KDEC [118] is a distributed mining scheme conceived to perform data

clustering based on non-parametric kernel density estimate of data [102, 159, 188]. The central idea is built on the fact that kernel density estimates are: (i) additive for homogeneously distributed datasets; and (ii) can be transmitted in the sampled form to hide the original data points.

Under the KDEC scheme, it is assumed that there is a set $\mathcal{L}$ of agents willing to start a clustering session, each of which is located at a different site S_j, with access to a local dataset D_j. Agents are organized in a pure peer-to-peer network. All agents in $\mathcal{L}$ agree on using the same distance d, kernel K, and window width h. Moreover, a distinguished agent is chosen among the peers to act as a helper, denoted L_{helper} (cf. Alg. 4.4).

The **negotiation protocol** is not explored in further detail in this thesis. However, we assume it to be a multiplayer negotiation protocol that seeks to find a consensus agreement about the parameters' values, e.g., the kernel bandwidth. To ensure outsiders will not eavesdrop on the negotiation, we assume an asymmetric key system, like RSA or ElGamal [148, Ch. 8], is in place. The negotiation could consist of two steps in the simplest form: (a) the initiator broadcasts a call for a mining session in the peer-to-peer network with proposed parameter values; (b) interested peers answer to the call accepting the proposed values. A more flexible approach would allow several rounds of counter-proposals until an agreement or a deadline is reached. There are several sophisticated negotiation protocols for multiparty negotiation, e.g. [10, 16, 137, 146] to name a few. After the agreement, the peers in the mining session are coordinated as indicated in the algorithm using direct messages between peers.

The KDEC scheme has four main steps, detailed below.

Local Density Estimation. Each agent $L_j \in \mathcal{L}$ computes its local kernel density estimate (LDE), which is denoted by $\hat{\varphi}_{K,h}[D_j](\cdot)$, as follows:

$$\hat{\varphi}_{K,h}[D_j](\mathbf{x}) = \frac{1}{N} \sum_{\mathbf{x}_i \in Neigh(\mathbf{x})} K\left(\frac{d(\mathbf{x}, \mathbf{x}_i)}{h}\right) \tag{4.2}$$

where $Neigh(\mathbf{x})$ represents the set of neighbors of point $\mathbf{x}$. LDE represents a local model of the data distribution at site $\mathcal{L}_j$ (cf. Fig. 4.2).

The sum of the LDEs equaling the global density estimate (GDE) is denoted as $\hat{\varphi}[D](\cdot)$. For brevity, this study will omit K, h from the notation in the following.

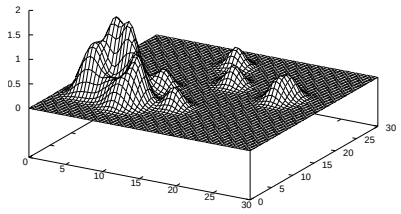


Figure 4.2: *LDE at site j. Density describes only local data*

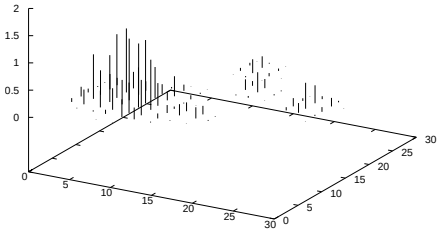


Figure 4.3: *Sampled LDE at site j, which will be sent to the Helper Agent.*

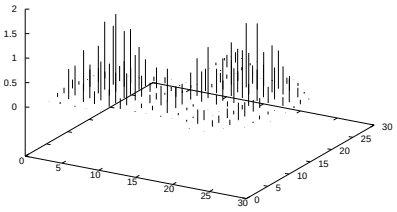


Figure 4.4: *Sampled GDE at the Helper site. This sampling represents the addition of all LDEs.*

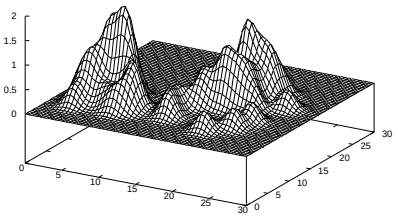


Figure 4.5: *Reconstructed GDE at Site j. GDE describes the whole distributed dataset.*

Sampling. The approach exploits multi-dimensional information sampling to minimize communications among sites. Before sending its LDE to the helper, each site transforms it into sampled form. For any $\mathbf{x} \in \mathbb{R}^n$, let $x_1, \ldots, x_n$ be its components. Let $\tau = [\tau_1, \ldots, \tau_n] \in \mathbb{R}^n$ be a vector of sampling periods and let $\mathbf{z} \bullet \tau$ denote $[\tau_1 z_1, \ldots, \tau_n z_n]$, where $\mathbf{z} \in \mathbb{Z}^n$. Let $R(\mathbf{z}_1, \mathbf{z}_2) \subseteq \mathbb{Z}^n$ be the n-dimensional rectangle having diagonal $(\mathbf{z}_1, \mathbf{z}_2)$. The sampled form of $\hat{\varphi}[D_j](\cdot)$ is the finite real sequence $\{\hat{\varphi}_{\mathbf{z}}[D_j]\}$ defined by:

$$\hat{\varphi}_{\mathbf{z}}[D_j] = \{\hat{\varphi}[D_j](\mathbf{z} \bullet \tau) \ : \ \mathbf{z} \in R(\mathbf{z}_1, \mathbf{z}_2)\} \tag{4.3}$$

where the sampling parameters $\mathbf{z}_1, \mathbf{z}_2 \in \mathbb{Z}^n$ and $\tau \in \mathbb{R}^n$ are previously agreed among the mining agents (cf. Fig. 4.3).

Global Density Estimates. The helper site, L_{helper}, receives all samples of local density estimates and computes the sampled global density estimates (GDE) using Equation (4.4) for all $\mathbf{z} \in R(\mathbf{z}_1, \mathbf{z}_2)$ as the sum of sampled LDEs.

$$\hat{\varphi}_{\mathbf{z}}[D] = \sum_{j=1}^{P} \hat{\varphi}_{\mathbf{z}}[D_j] \tag{4.4}$$

Algorithm 4.1 DECluster

Input: local dataset D_j, global sample density estimate *globalSam*,kernel K, window width h, initial local cluster map $\mathcal{C}$
Output: final local cluster map $\mathcal{C}$
Method:

```
1: for i = 1 to D_j.count do
2:     if not(C.clustered(i)) then fixedPoint(i, globalSam, K, h, C);
3:     end if
4: end for
5: return C
```

This sum is possible because density estimates in the sampled form are additive. Then, the helper sends the sampled GDE back to the peers (cf. Figs. 4.4 and 4.5).

Local Clustering. From the sampled global density estimate $\hat{\varphi}_{\mathbf{z}}[D]$ the local agents can approximate the true GDE using the interpolation formula

$$\hat{\varphi}[D](\mathbf{x}) = \sum_{\mathbf{z} \in R(\mathbf{z}_1, \mathbf{z}_2)} \hat{\varphi}_{\mathbf{z}}[D] \; sinc\left(\frac{x_1}{\tau_1} - z_1\right) \cdots sinc\left(\frac{x_n}{\tau_n} - z_n\right) \tag{4.5}$$

where

$$sinc(x) = \begin{cases} 1 & \text{if } x = 0, \\ \frac{\sin \pi x}{\pi x} & \text{otherwise.} \end{cases}$$

Expression (4.5) is an application of the well-known Whittaker-Shannon interpolation formula (see, e.g., [101]) to the domain of density estimates. Note that the function represented by Equation (4.5) is not extensionally equal to the kernel global estimate $\hat{\varphi}[D](\cdot)$, both because kernel estimates are not band-limited to any frequency region, and because of the truncation in the series. However, it was shown in [118] that the approximation only introduces a small error and, consequently, we can choose τ so that the Fourier transform of the estimate $\hat{\varphi}[D](\cdot)$ is negligible outside the region $[-\pi/\tau_1, \pi/\tau_1) \times \cdots \times [-\pi/\tau_n, \pi/\tau_n)$, and $\mathbf{z}_1, \mathbf{z}_2$ such that the estimate is negligible outside the region defined by the corners $\mathbf{z}_1$ and $\mathbf{z}_2$.

Finally, each local mining agent L_j, at site S_j, can use the reconstructed global density estimate to cluster its local data (cf. Alg. 4.1). To this end, it uses a gradient-driven hill-climbing procedure to find local maxima points in the global density estimate. All points connected to one given local maximum are labeled to the same cluster. Function uphill() advances a fraction δ of

Algorithm 4.2 DECluster's Auxiliary Functions

```
 1: function fixedPoint(i, D_j, globalSam, τ, C)
 2:     C.setVisited (i);
 3:     u ← uphill(i, D_j, globalSam, τ, epsilon);
 4:     if C.clustered (u) then
 5:         C.setLabel(i,C.getLabel (u));
 6:     else
 7:         if C.visited (u) then C.setLabel(i, u);
 8:         else
 9:             fixedPoint(u, D_j, globalSam, τ, C);
10:             C.setLabel(i, C.getLabel(j));
11:         end if
12:     end if
13:     C.setClustered(i);
14: end function

 1: function uphill(i,S, globalSam, τ, ϵ)
 2:     x ← S.get(i);
 3:     v ← SeriesGradient(x, globalSam, τ);          // using Eq. 4.5
 4:     if ||v|| > ϵ then
 5:         return x + v * δ
 6:     else
 7:         return x
 8:     end if
 9: end function
```

the gradient in its direction, if the gradient's norm exceeds a threshold ϵ. If a δ-neighborhood of the object returned by uphill() contains an already clustered data object, the current cluster label Id is set from that object's label. Otherwise, the act of checking whether uphill() returned the same space object signals to fixedPoint() that the proximity of the local maximum has been reached. The maximum is marked by the method setVisited() applied to the current space object x; this ensures that subsequent paths converging to the same local maximum will use the same cluster label as the current path. Algorithm 4.1 shows the pseudo-code for the local clustering procedure.

The pseudo-code for an arbitrary agent L_j is shown in Algorithm 4.3, while Algorithm 4.4 shows the pseudo code for Helper.

4.2.2 Complexity Analysis

Time. KDEC is superlinear in the number of points in the dataset.

Theorem 4.1 *KDEC requires $O(|G|M_j + Nq(N))$ steps at a local site and $O(|G|)$ at helper site, where $|G|$ denotes the number of sampling points and*

Algorithm 4.3 KDEC: Arbitrary Site

Input: a group of mining agents $\mathcal{L}$, local dataset D_j, kernel function K, widow width h, sample rate τ, corners z_1 and z_2
Output: Cluster map $\mathcal{C}_j$
Method:
At an arbitrary party j do:
1: negotiate(L_{Helper}, K,h,τ, z_1,z_2);
2: $sam_j \leftarrow$ sampleDensityEstimate(D_j, K, h,τ, z_1, z_2);
3: **send** sam_j **to** L_{Helper};
4: **receive** $globalSam$ **from** L_{Helper};
5: **return** $\mathcal{C}_j \leftarrow$ DECluster(D_j, $globalSam$); // *cf. Alg. 4.1*

Algorithm 4.4 KDEC: Helper Site

Input: a group of mining agents $\mathcal{L}$.
Output: global sampled density estimate $globalSam$
Method:
At the *Helper* site do:
1: **receive** sam_j **from** all agents $L_j \in \mathcal{L}$;
2: **send** $globalSam = \sum_{j=1}^{|\mathcal{L}|} sam_j$ **to** all agent $L_j \in \mathcal{L}$;

M_j average size of the neighbor set, $q(N)$ is the cost of the nearest neighbor query and N is the size of the data set.

Proof. Density estimates are computed only at $|G|$ sample points (grid points), spaced by the sampling rate τ. For each sample point, only the nearest neighbors are considered. Therefore, local estimation and sampling take $O(|G|M_j)$ steps, where $|G|$ denotes the number of sampling points and M_j is the average size of the neighbor set. In most cases, to obtain reasonable density estimates, h must not be less than a small multiple of the smallest object distance. As $\tau \approx h/2$, the number of samples should rarely exceed the number of objects in D_j if only space regions where the density estimate is not negligible are sampled.

The time complexity of DECluster is $O(Nq(N))$, where $q(N)$ is the cost of a nearest neighbor query. Function DECluster calls $N = D_j.count$ times fixedPoint(). At the beginning of every iteration in DECluster, the sets of clustered and visited objects are equal. fixedPoint() is never called with a clustered object as an argument and visits unclustered objects at most once. Therefore, even if the number of visited data objects in one call of fixedPoint() is bounded only by N, the number of visited data objects in all calls is only N. For each visited point, a single k-nearest neighbor query suffices to compute the gradient and the next uphill object. The methods

for nearest neighbor queries can be efficiently implemented by spatial access structures like the KD-, or MVP-, or M-tree.

Upon receiving all samples from the local sites, the helper adds all samples up with Eq. 4.4, summing all estimates from different peers for a given sample point z. Note that $|G|$ is given by the size of the sampling rectangle defined by the corners $\mathbf{z}_1$ and $\mathbf{z}_2$ and sample rate τ.

Therefore, the overall complexity of KDEC at a local site is then $O(|G|M_j + Nq(N))$. At the helper, it is necessary $O(|\mathcal{L}||G|)$ steps to compute global density, where $|\mathcal{L}|$ is the number of peers. Notice that the cost of clustering $Nq(N)$ dominates over the cost of density estimation $|G|M_j$.

□

Communication. The size of messages in KDEC is given by sampling parameters and is independent of the size of the dataset.

Theorem 4.2 *KDEC generates message of size $O(|G|)$, where G sampling grid.*

Proof. The sampling points are inside a rectangle defined by the corners $\mathbf{z}_1$ and $\mathbf{z}_2$ and sample rate τ. The size $|G|$ is independent of the size of the dataset. □

Let us remark that $|G|$ is usually much smaller than the size of a dataset D_j when only space regions with non-negligible density estimates are sampled.

4.2.3 Inference and Collusion Attacks Analysis

In this section, we analyze the extent to which KDEC preserves privacy under the different attack scenarios presented in Chapter 3.

Insider attacks

In the following, we first analyze single insider attacks and then investigate attacks under collusion with different formations of malicious groups.

Single Insider Attack. The most simple attack scenario is the attack performed by a single malicious participant of the mining group. The attacker is assumed to be a mining peer who executes all steps of the KDEC protocol as expected and tries to reconstruct the original data points after receiving the global density. Since this attempt occurs after the protocol termination, this attack is always likely to occur, and there is no way to detect it. The question is: *what can an insider learn from other parties and to what extent?*

Recall that an insider attacker knows the parameter values agreed before the KDEC protocol starts, i.e., the kernel function K, the window width h, and the distance function d. The attacker is also assumed to have a local dataset D_j, used during the protocol. Moreover, the sampled global density estimation is distributed to all agents who cooperate in the data mining process, including the attacker.

From the above, we can state the basic privacy levels of KDEC, under the different privacy metrics from the previous chapter. Note that $\mathbf{PR}^{range}_{KDEC}$ is not applicable in this context since a global cluster map is not disclosed. Furthermore, a global density estimate is available. In this case, a more precise metric is $\mathbf{PR}^{BK}_{KDEC}$. The resulting privacy level will be expressed as $\mathbf{PR}^{DCP}_{[c]}(D)$, as discussed in the previous chapter.

Lemma 4.1 *Given dataset D, with global density estimates $\hat{\varphi}[D](\mathbf{x})$, the privacy level of KDEC under a single insider attack can be computed as:*

$$\mathbf{PR}^{DCP}_{KDEC[1]} = \mathbf{PR}^{BK}_{KDEC[1]}(D) = 2^{h(D)} \tag{4.6}$$

with the entropy $h(D)$ computed as

$$h(D) = -\int \hat{\varphi}(x) \log_2 \hat{\varphi}(x) dx \tag{4.7}$$

where $\hat{\varphi}[D](\mathbf{x})$ is the global density estimates.

Proof. This is a direct application of Eq. 3.9, using the global density estimates $\hat{\varphi}[D](\mathbf{x})$ as background knowledge the attacker has on the global data distribution. The attacker has only samples of the global density. However, the global estimates $\hat{\varphi}[D](\mathbf{x})$ can be reconstructed using the Eq. 4.5, which is possible for the single inside attacker because it knows all parameter values necessary to use the said equation. □

Algorithm 4.5 Pointhunt

Input: x_0, D_0, ε;
Output: D';
Method:
1: $x \leftarrow \mathsf{search}(x_0, D_0, \varepsilon)$;
2: $\delta \leftarrow |\ \mathsf{density}(x) - \hat{\varphi}[D_0](x)\ |$;
3: $D' \leftarrow D_0$;
4: **if** $x \neq x_0$ **then**
5: $\quad s_{new} \leftarrow \mathsf{reconstruct}(x,\delta)$;
6: $\quad D' \leftarrow D_0 \cup \{s_{new}\}$;
7: **end if**
8: **return** D'

Algorithm 4.6 Pointhunt's Auxiliary Functions

1: **function** $\mathsf{reconstruct}(x, \delta)$;
2: $\quad$ **return** $x + hK^{-1}(\delta)$;
3: **end function**
1: **function** $\mathsf{search}(x, D_0, \varepsilon)$;
2: $\quad Y \leftarrow \{y \in [x, max]\ :\ |\mathsf{density}(y) - \hat{\varphi}[D_0](y)| \leq \varepsilon\}$
3: $\quad$ **if** $Y \neq \emptyset$ **then**
4: $\quad\quad$ **return** $\min Y$;
5: $\quad$ **end if**
6: $\quad$ **return** x;
7: **end function**

An important question is: can a malicious peer use a reconstruction algorithm to infer non-local sensitive data in KDEC? To illustrate the intricacies of an insider attack in KDEC, the pointhunt() algorithm (cf. Alg. 4.5) was developed – a simple data reconstruction method for datasets of reals. pointhunt() can be used to perform an inference attack given a density estimate of the victim's dataset [52].

Lemma 4.2 *Given a data set D and its global density estimates $\hat{\varphi}[D](\mathbf{x})$, if the estimates were computed with a kernel K, an inside attacker is able to produce D', which is a reconstruction of the original dataset D containing at least one point, with arbitrary precision using the pointHunt algorithm (cf. Alg. 4.5).*

Proof. As input, pointhunt needs a starting point x_0, a (possibly empty) set of reconstructed data points D_0, and a threshold ε representing the deviation of the current from density. The density function implements the computation of the global density estimates of any given point with the current kernel function and dataset D (using Eq. (3.9)). Furthermore, it assumes that the

parameter values of the mining section being attacked are known, i.e. a function K and its inverse K^{-1}, window width h, the interval $[min, max] \supseteq D$ such that $\{\mathsf{density}(min), \mathsf{density}(max)\} \subset [0, \varepsilon]$. Note that parameter values, which are defined in the negotiation step, are available to any insider peer.

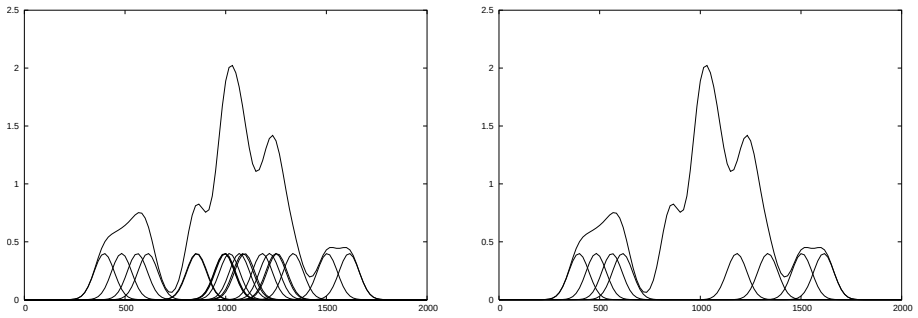
The algorithm works by reconstructing one data point at a time, from left to right, as follows. Initially, $x_0 = min$ and D_0 must equal a (possibly empty) set of points that are already known to be in D, e.g. the attackers' local dataset. Function search locates the leftmost point x to the right of x_0 where the difference between the actual and reconstructed density is not negligible, i.e. exceeds ε. Given x, a point s_{new}, which is likely to be in D, is calculated by function reconstruct using hK^{-1}. This heuristic can be informally justified by noting that x is the leftmost location that is significantly influenced by $D \setminus D_0$ and, therefore, x is likely to be significantly influenced by only one point in $D \setminus D_0$.

The ideal case for an attacker occurs when K, and, consequently, the estimate, has bounded support. Examples of kernels with bounded support include the triangular pulse kernel[1] and the Epanechnikov's kernel[2]. Then ε can be set to zero and search returns a point of the border of the support of the function $\mathsf{density}(x) - \hat{\varphi}[D_0](x)$. Assuming, without loss of generality, that $K^{-1}\colon [0, w_{max}] \to [0, 1]$, with $w_{max} = K(0)$, then there must be one data point that is located at $x + hK^{-1}$, which is returned by the function reconstruct. D' is built by calling pointhunting iteratively using the D' as initial guess for the next iteration. □

Whereas the experiments with bounded kernels led to full disclosure of the dataset, the attacker is less likely to succeed if the kernel has unbounded support, e.g., the Gaussian kernel. In general, it is fair to affirm that the best results are obtained when points are not too close to one another given a value of h. Figure 4.6(a) shows the density estimate and the contribution of each kernel corresponding to the original data points. Figure 4.6(b) shows the data points which were correctly reconstructed by pointhunt. Notice that this simple algorithm could not locate the points in the region where said points are too close to one another. On the other hand, one cannot ignore

[1] The triangular pulse kernel is defined as $K(u) = (1 - |u|)\,\mathbf{I}_{\{|u| \leq 1\}}$, where $\mathbf{I}$ is the indicator function.

[2] The Epanechnikov kernel is defined as $K(u) = \frac{3}{4}(1 - u^2)\,\mathbf{I}_{\{|u| \leq 1\}}$, where $\mathbf{I}$ is the indicator function.



(a) Original data (b) Reconstructed data

Figure 4.6: *Original data (a): Gaussians centered at each point are indicated under the density curve. Reconstructed data (b): some points were not reconstructed where the density is too high.*

the effect of the attacker's local dataset on the global density estimate. The attacker may use its local dataset as an initial set in the pointhunt, thus increasing the accuracy of the attack. In this study's experiments, the better the initial dataset, the better the reconstruction results were.

This attack leads to a reconstruction of data points with arbitrarily high precision, which is controlled by the parameter ε in the algorithm pointhunt. Therefore, assuming that the sensitive information is an attribute value, the single insider attacker may reconstruct it with an arbitrary level of confidence, which is summarized as follows:

- If the kernel is bounded, the single insider attacker can reconstruct all sensitive points to an arbitrary degree of precision chosen by the attacker.
- If the kernel is unbounded, the single insider attacker can reconstruct some sensitive points to arbitrary precision chosen by the attacker. Points in crowded regions are less likely to be reconstructed by pointhunt.

If the kernel is bounded, the attacker can transverse the density estimates detecting the kernel borders, one border for each point in the dataset. This attack is possible since the malicious peer knows which kernel K was used to produce $\hat{\varphi}[D](\mathbf{x})$, with all relevant parameters to compute its inverse.

In an attack against an unbounded kernel, the attacker needs to apply pointhunt iteratively. The total computational effort is $O(nN)$ since every invocation of pointhunt requires the computation of the density around a

given point x, where n is the size of the dataset being attacked and N is the number of neighbors of an arbitrary point x considered by the density function.

Nonetheless, if there are more than two parties, the attacker cannot assign the reconstructed data points to the data owner. In the two-party case, the attacker can subtract its density estimate from the global estimate, and, therefore, the reconstructed points can be assigned to the other party.

Lemma 4.3 *A single insider attack against KDEC with pointhunting results in privacy level*

$$\mathbf{PR}^{DCP}_{KDEC[1]} = \mathbf{PR}^{rec}_{KDEC[1]}(D) \approx \varepsilon \tag{4.8}$$

with arbitrarily small ε, i.e. $\varepsilon \approx 0$.

Proof. From lemma 4.2 we have that pointhunting produces a reconstructed dataset D'. The error in the estimation of the true data points is controlled by ε which is given as parameter and is used in the search function (cf. auxiliary functions in 4.6). Points in the extremity of the data space will receive influences of fewer neighbors in the density space and will be more easily detected by pointhunt (cf. Alg. 4.5). Therefore, at least the points in the less dense regions of the original dataset will be reconstructed into the dataset D' within the range of precision with $\varepsilon \approx 0$. □

Collusion Attack without Helper. Let $\mathcal{L}$ be a mining group and $\mathcal{M} \subset \mathcal{L}$ a collusion group. Further, $\mathcal{M}$ does not include the helper. In this attack, the members of $\mathcal{M}$ coordinate their actions to learn sensitive information owned by the remaining peers in the group $\mathcal{L} \setminus \mathcal{M}$. The reconstruction method used by the attackers can be the same as in the insider attack. It is assumed that colluders know information as in the previous attack scenario, i.e., all parameter values and a local dataset. Additionally, the assumption is made that attackers can exchange local density estimates.

As in the single insider attack, a collusion attack may disclose points located in not overly crowded regions. Moreover, attacker agents exchange their local density estimates so that the partial density produced by the summation of attackers can be subtracted from the global density estimate. The resulting density estimate represents datasets owned by $\mathcal{L} \setminus \mathcal{M}$. Using pointhunt, the attackers may use their local datasets as an initial set, thereby

increasing the precision of the reconstructed sensitive data. Therefore, every peer needs to be sure its local dataset does contain less populated regions to safeguard against the risk of disclosure.

There is another critical point to mention regarding collusion attacks. If the number of attackers is $|\mathcal{L}| - 1$, the data ownership is also revealed, i.e., attackers can assign the data points to a specific peer or data holder, which is sensitive information in some domains, like health care (knowing the death rate of a specific hospital is a piece of sensitive information about that hospital).

In essence, colluders may learn:

- everything a single insider attacker may learn, i.e., sensitive values of points in regions with low density
- additionally, a collusion group may be able to disclose the data holder's identity.

A collusion attack involves an extra message round to exchange extra knowledge after the protocol is over. The reconstruction procedure, as in the single insider attack, takes a computational effort which is $O(nN)$, where n is the number of points to be disclosed, and N denotes the number of neighbors of an arbitrary point $\mathbf{x}$.

Collusion Attack with Helper. The helper is a distinguished peer chosen in the negotiation phase, used to collect local densities sent by different peers and sum them up. The helper in KDEC knows all of the sampled local densities. However, it does not know the parameters' values since they are negotiated directly between the mining peers in the first phase of the protocol. Consequently, a malicious helper must cooperate with other peers inside the mining group to get the parameter values.

Assuming that a malicious helper forms a collusion group with a mining peer, it can use pointhunt to reconstruct sensitive data based on the individual local density estimate, as received from the peers. Additionally, this attack reveals data ownership since the helper can assign each local density estimate to a specific mining peer. Further, colluders may also send a subset of local datasets to improve the reconstruction results obtained by the malicious helper.

The colluding helper attack has the same complexity concerning time and communication as the collusion attack discussed previously.

Summary of Insider Attacks.

Theorem 4.3 *When colluding groups are formed by $c > 1$ colluders, a collusion attack, with or without the helper, against KDEC with* pointhunting *yields privacy level of* $\mathbf{PR}^{DCP}_{KDEC[1]}(D) \geq \mathbf{PR}^{DCP}_{KDEC[c]}(D) \approx 0$.

Proof. Colluders already have access to the global density. Therefore, using pointhunting the attackers may reconstruct sensitive data as shown in lemma 4.2. With more information, e.g. local density estimates, the precision of the attack can be improved, because the initial dataset passed to pointhunting allows the process to focus only on the data points in $D \setminus D_c$, where D_c is the union of datasets owned by the malicious peers in the collusion. □

Outsider attacks

This study's scrutiny must now turn to the situation where the attacker is not a member of the mining group. In this scenario, it is unknown which information the attacker has. For that reason, the analysis of the different degrees of reconstruction is based on assumptions made regarding the information the attacker managed to acquire. It is assumed that the outside attacker does not collude with any group member; otherwise, the scenario would be equivalent to an insider attack. Here, the goal is to answer what can be learned by the outside attacker, assuming it is working alone, without any insider help.

Extreme Case. An extreme case occurs when the attacker manages to eavesdrop on all of the parameters' values. It is assumed that the attacker can use a reconstruction function like the pointhunt algorithm. Note, however, that the attacker has no access to any local dataset used to compute the global density estimate. A local dataset would allow the attacker to reconstruct more data points and improve the reconstructed data's precision.

Considering every factor combined, the attacker can learn sensitive data represented by points in less dense regions. Compared to the single insider

attack, the main difference is that the outsider's initial set is empty, whereas the insider uses its local dataset in data reconstruction.

Attack without the window width h. The parameter h defines how the kernel function will be stretched in the x-axis, i.e., h determines the range of influence of one point over its neighbors. If h is unknown, it is impossible to compute $\varphi[D]$ and, consequently, pointhunt cannot be used. However, if there is at least one outlier point we can compute h in the following way. Let the set $X^* = \{x^* : \varphi[D](x^*) = K(0)\}$ is a set of outliers. Let us choose a point x_c close to x^* such that $\varphi[D](x_c) = w < K(0)$. Using the kernel inverse K^{-1}: $[0, w_{max}] \rightarrow \mathbb{R}_+ \cup \{0\}$, where $w_{max} = K(0)$, we can compute $K^{-1}(\varphi[D](x_c)) = K^{-1}(w) = d_w$ representing the distance from x^* where one point x_c must be placed to receive the influence w from x^*. Nevertheless x_c lies at $d(x^*, x_c)$ from x^* because it was scaled by h in the computation of $\varphi[D]$. We have that $K(\frac{d(x^*,x_c)}{h}) = w = K(d_w)$ what give us $\frac{d_c}{h} = d_w$. After substitutions we get $\frac{d(x^*,x_c)}{h} = K^{-1}(\varphi[D](x_c))$. Finally, h can be computed with:

$$h = \frac{d(x^*, x_c)}{K^{-1}(\varphi[D](x_c))} \tag{4.9}$$

In this scenario, the extra effort to the attacker comes from computing the parameter h. Building X^* can be done in a single pass through the points in $\varphi[D]$ and involves only looking for the smallest local maxima, which takes $O(n)$, where n is the number of outliers in the density estimate $\varphi[D]$.

Attack without the distance function. The distance function plays a crucial role in the computation of the GDE. Its inverse is also critical since the attack algorithm uses it to reconstruct the points. Assuming that the distance is unknown to the attacker, he/she may try to use well-known distance functions, e.g., Euclidean distance. To test the efficiency of the distance function chosen, the attacker can use $hK^{-1}(\varphi[D](x_i)) = d(x^*, x_i)$, where x^* is an outlier point, and $x_i, i = 1, ..., n$ are points close to x^*. The success of this approach will depend on how many different candidate functions are chosen and if there is at least one outlier in the GDE. Once the attacker has found a suitable distance function, the attack can proceed normally.

Attack without the kernel function. Without the kernel function, the attacker cannot use the trial-and-error approach used in the inside attack

to reconstruct the points. There are at least two options; the first involves attempting to guess the kernel function, while the second option consists of trying to find the points with methods independent of knowing the kernel function.

Guessing the kernel can be accomplished if the dataset contains outliers. In this case, the attacker can build a table with the density of points located around an outlier point x^*. This table can be interpolated to get a candidate kernel function $\hat{K}$. This $\hat{K}$ is simple to build though it will undoubtedly lead to many approximation errors, thus compromising its usability in pointhunt.

Finding points without kernel can be accomplished if the kernel function, or its derivatives, has discontinuities. It is possible to find the points using a simple observation: the distances between discontinuities on one axis are equal to the distances between data points on the same axis.

Attack without the sampling parameter. KDEC uses a multidimensional sampling technique to transform the density estimates into a sequence of indexed values. These indexes allow the peer agents to transmit information without explicit reference to the original data points. The sampling parameter is $\tau \in \mathbb{R}^n$, which is chosen in the initial phase of the protocol.

Without τ, it is impossible to reconstruct the data points. However, if h and K are known, two successive values can be chosen with respect to one axis, $w_{z1}, w_{z2} < K(0)$ such that $hK^{-1}(w_{z1}) = d_1$ and $hK^{-1}(w_{z2}) = d_2$ and attempt to find $\tau = \frac{|d_1 - d_2|}{|z_1 - z_2|}$.

It is difficult to pinpoint the accuracy of any given outside attack since it depends on the attacker's ability to reconstruct the missing information. The practical approach is to assume the worst-case scenario (the eavesdropping of all parameter values) since the consequences of assuming a less dangerous scenario may have drastic consequences.

The bottom line is that outside attackers will only enjoy an effective reconstruction if they manage to eavesdrop on all of the relevant information used by members of the mining group. The main limitation for the attackers is that, without a good initial set, the pointhunt() algorithm may fail to disclose points in dense regions, as discussed in previous sections.

Therefore, KDEC provides more privacy level under outsider attack, or

at least similar, to the privacy level under insider attacks.

$$\mathbf{PR}^{DCP}_{KDEC[0]}(D) \geq \mathbf{PR}^{DCP}_{KDEC[c]}(D) \approx 0 \tag{4.10}$$

with $c \geq 1$.

Summary of privacy analysis

The different attack scenarios discussed in the previous section showed that insider attacks with collusion have the best chance of disclosing sensitive information. When the attacker is part of the mining group, the accuracy of reconstructed data may be very high, leading to possible privacy breaches. Unfortunately, the level of privacy cannot be controlled by the data holder, and the success of an attack depends only on the attacker's local dataset and the particularities of data distribution. For example, data points in dense regions are more likely to be poorly reconstructed. The following lemma captures the idea.

Lemma 4.4 *Let $\mathcal{L}$ be a mining group which is performing KDEC protocol with $c < |\mathcal{L}|$ malicious agents forming a collusion group. Let $\tau \in \mathbb{R}$ be the sampling rate chosen by the mining group. Let K be a kernel function and a density estimate point $y = \varphi[D](\mathbf{x})$. If $y < K(0)$ there is a reconstruction procedure such that $\mathbf{PR}^{DCP}_{KDEC[c]}(D) = \mathbf{PR}^{rec}_{KDEC[c]}(D) \ll \tau$, for all $c > 1$.*

Proof. We assume that the attacker uses pointhunt algorithm (cf. Alg. 4.5). Lemma 4.3 shows that $\mathbf{PR}^{rec}_{KDEC[1]} \approx \varepsilon$. Recall that ε is chosen to be arbitrarily small, i.e. $\varepsilon \ll \tau$. Thus, $\mathbf{PR}_{KDEC[1]} \ll \tau$. With collusion group (lemma 4.3), this reconstruction may be more accurate. Therefore, $\mathbf{PR}^{rec}_{KDEC[c]} \leq \mathbf{PR}^{rec}_{KDEC[1]} \ll \tau$, for all $c > 0$. □

In the next section, we address how to improve the privacy level of KDEC and propose a novel algorithm, the KDEC-S.

4.3 KDEC-S Algorithm

KDEC-S is a distributed clustering scheme based on the KDEC scheme (cf. 4.2), which aims to provide better privacy-preserving properties than KDEC.

Recall that, in the KDEC scheme, each site transmits the local density estimate to a helper site, which is responsible for building a global density estimate and sending it back to the mining peers. Using the global density estimate, the mining peers can locally execute a density-based clustering algorithm. KDEC-S works similarly but replaces the original estimation with an approximated value. The aim is to preserve data privacy while maintaining enough information to guide the clustering process.

4.3.1 Algorithm Overview

Following the density-based approach for DDC, each peer contributes to the mining task with a local density estimate of the local dataset. Consequently, no data point, original or randomized, needs to be exchanged among the peers. Using estimates instead of raw data is a first step towards making the distributed mining operation safer concerning data disclosure. Still, as shown in previous sections, knowing the inverse of kernel function affects the reconstruction of original sensitive data in some cases.

Once inference and collusion attacks against KDEC have been investigated, the question is how to address said attacks. This study aims to improve the density estimate's privacy level, proposing to substitute the density estimate with an approximated (non-invertible) function. By doing this, one of the attack's assumptions is removed – namely, the assumption that the kernel inverse exists. Additionally, this study wants to give the users the freedom to choose whichever kernel they want to work with, whether it is bounded or not. In the following, the details of this approach are outlined.

First, it is necessary to present some basic definitions.

Definition 4.1 (Iso-levels) *Let $f : \mathbb{R}_+ \cup \{0\} \to \mathbb{R}_+$ be a decreasing function. Let $\tau \in \mathbb{R}$ be a sampling rate and let $z \in \mathbb{N}$ be an index. Denote by $\mathbf{v} \in \mathbb{R}^n$ a vector of iso-levels[3] of function f, whose each component $v^{(z)}$, $z = 1, 2, \ldots, n$, is built as follow:*

$$v^{(z)} = f(z\tau) \tag{4.11}$$

Moreover $v^{(1)} > \ldots > v^{(n)} > 0$. □

[3] One can understand $\mathbf{v}$ as iso-lines used to contour plots

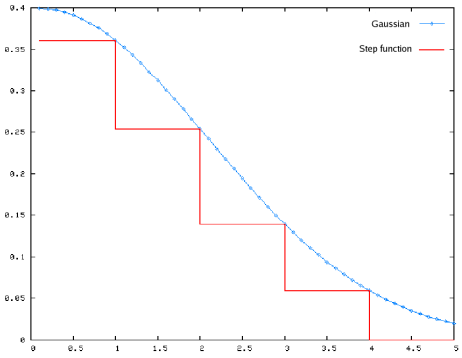


Figure 4.7: $\psi_{f,\mathbf{v}}$ *of the Gaussian function.*

Definition 4.2 (Kernel Transformation) *Let* $f : \mathbb{R}_+ \cup \{0\} \rightarrow \mathbb{R}$ *be a decreasing function. Let* $\mathbf{v}$ *be a vector of iso-levels of* f*. Then we define the function* $\psi_{f,\mathbf{v}}$ *as:*

$$\psi_{f,\mathbf{v}}(x) = \begin{cases} v^{(1)}, \textit{if } v^{(1)} \leq f(x) \\ v^{(z)}, \textit{if } v^{(z)} \leq f(x) < v^{(z-1)} \\ 0, \textit{if } f(x) \leq v^{(n)} \end{cases} \tag{4.12}$$

□

Together, definitions 4.1 and 4.2 define a step function based on the shape of a given function f. Figure 4.7 shows an example of $\psi_{f,\mathbf{v}}$ applied to a Gaussian[4] function with $\mu = 0$ and $\sigma = 2$, using four iso-levels. Note that the resulting stepwise function has no inverse, although it resembles the original shape of the desired function. The number of iso-levels and the sample rate used controls how close the approximation is to the original shape. As a direct consequence, it is evident that many different functions will be transformed to the same $\psi_{f,\mathbf{v}}$. This ambiguity associated with the function $\psi_{f,\mathbf{v}}$ is the key in avoiding the reconstruction attack based on kernel inverse (cf. Fig. 4.8).

The following lemma formalizes the amount of ambiguity provided by the transformation $\psi_{f,\mathbf{v}}$.

Lemma 4.5 *Let* $\tau \in \mathbb{R}$ *denote a sampling rate, and* $z \in \mathbb{N}$ *be an index.*

[4]Gaussian function is defined by $f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-(x-\mu)^2/2\sigma^2}$

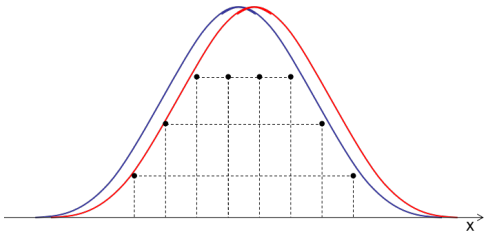


Figure 4.8: *Ambiguity of transformation: given two similar functions f_1 and f_2 and the same iso-levels $\mathbf{v}$ the resulting $\psi_{f,\mathbf{v}}$ is the same.*

Define $f_1 : \mathbb{R}_+ \to \mathbb{R}_+$, a decreasing function and $\mathbf{v}$, a vector of iso-levels. If we define a function $f_2(x) = f_1(x-k)$, then $\forall k \in [0,\tau), \forall z \in \mathbb{N}$ we will have $\psi_{f_2,\mathbf{v}}(z\tau) = \psi_{f_1,\mathbf{v}}(z\tau)$.

Proof. For $k = 0$, we get $f_2(x) = f_1(x-0)$ and it is trivial to see that the assertion holds. For $0 < k < \tau$, we have $f_2(x) = f_1(x-k)$. Without loss of generality, let $z > 0$ be some integer and let $x = z\tau$. Consequently, $(z-1)\tau < x = z\tau$. So, $f_2(x) = f_2(z\tau) = f_1(z\tau - k)$. With decreasing f_1 we have that $f_1(z\tau) < f_1([z-1]\tau)$ and $\psi_{f_1,\mathbf{v}}(z\tau) = f_1(z\tau)$. Now, if we rewrite $(z-1)\tau$ as $z\tau - \tau$, and since f_1 is decreasing, $f_1(z\tau) \leq f_1(z\tau - k) < f_1(z\tau - \tau) = f_1((z-1)\tau)$. With $0 \leq k < \tau$ we have that $f_1(z\tau - k) < f_1((z-1)\tau)$. By Definition 4.2, we can write $\psi_{f_1,\mathbf{v}}(z\tau - k) = \psi_{f_1,\mathbf{v}}(z\tau)$. Additionally, $f_2(z\tau) = f_1(z\tau - k) < f_1((z-1)\tau)$ and $\psi_{f_2,\mathbf{v}}(z\tau) = \psi_{f_1,\mathbf{v}}(z\tau - k)$. Therefore, $\psi_{f_2,\mathbf{v}}(z\tau) = \psi_{f_1,\mathbf{v}}(z\tau)$ □

Lemma 4.5 tell us that the inverse of $\psi_{f_2,\mathbf{v}}(x)$ is undetermined because several different functions generates the same approximation. All translations $f_k(x)$ of a given function $f(x)$ given by $f_k(x) = f(x-k)$ generates the same transform with $k < tau$. As a consequence, the attacker cannot determine the exact location of a given point from its density, because several points generates the same approximated density estimate. We exploit this feature as a countermeasure to point hunting attacks discussed earlier in this chapter (cf. Section 4.2.3).

Now we substitute the kernel K by $\psi_{K,\mathbf{v}}$ for a given sample rate τ, in the process of estimating the density of the local datasets. Since $\psi_{K,\mathbf{v}}$ is a non-increasing function, we can use it as a kernel function. We compute a rough approximation of the local density estimate using the function $\psi_{K,\mathbf{v}}$

as follows:

$$\tilde{\varphi}[D_j](x) = \begin{cases} \sum_{x_i \in N_x} \psi_{K,\mathbf{v}}(\frac{d(x,x_i)}{h}) & \text{, if } (x \mod \tau) = 0 \\ 0 & \text{, otherwise.} \end{cases} \tag{4.13}$$

where N_x denotes the neighborhood of x. The global approximation can be computed by:

$$\tilde{\varphi}[D](x) = \sum_{j=1}^{P} \tilde{\varphi}[D_j](x) \tag{4.14}$$

According to Lemma 4.5, it is not possible to decide which one of the original functions produced the approximate kernel $\psi_{K,\mathbf{v}}$.

Auxiliary Definitions

Definition 4.3 (Grid) *Given two vectors $\boldsymbol{z}_{low}, \boldsymbol{z}_{high} \in \mathbb{Z}^n$, which differ in all coordinates (called the sampling corners), we define a grid G as the filled-in cube in $\mathbb{Z}^n$ defined by $\boldsymbol{z}_{low}, \boldsymbol{z}_{high}$. Moreover, for all $z \in G$, define $n_z \in \mathbb{N}$ as a unique index for z (the index code of z). Assume that z_{low} has index code zero.*

□

Definition 4.4 (Sampling Set) *Let G be a grid and $\tau \in \mathbb{R}^n$ be a sampling rate. We define a sampling $\mathcal{S}_j$ of $\tilde{\varphi}[D_j]$ given a grid G, as:*

$$\mathcal{S}_j = \left\{ \tilde{\varphi}_z^j \mid \forall z \in G,\ \tilde{\varphi}_z^j > 0 \right\} \tag{4.15}$$

where $\tilde{\varphi}_z^j = \tilde{\varphi}[D_j](z \cdot \tau)$. Similarly, the global sampling set will be defined as: $\mathcal{S} = \{\tilde{\varphi}_z \mid \forall z \in G,\ \tilde{\varphi}_z > 0\}$

□

Definition 4.5 (Cluster guide) *A cluster guide $CG_{i,\theta}$ is a set of index codes representing the grid points forming a region with density above some threshold θ:*

$$CG_{i,\theta} = \{n_z \mid \tilde{\varphi}_z \geq \theta\} \tag{4.16}$$

such that $\forall n_{z_1}, n_{z_2} \in CG_{i,\theta} : z_1$ and z_2 are grid neighbors and $\bigcap_{i=1}^{k} CG_{i,\theta} = \emptyset$.

Algorithm 4.7 KDEC-S: Local Peer

Input: D_j (local dataset), $\mathcal{L}$ (list of peers agents), L_{Helper};
Output: $clusterMap$;
Method:

```
1: negotiate(L, K, h, G, θ);
2: φ̃[D_j] ← estimateApprox(K, h, D_j, G, δ);
3: S_j ← buildSamplingSets(φ̃[D_j], G, θ, v);
4: send S_j to L_Helper;
5: receive CG_θ from L_Helper;
6: clusterMap ← cluster(CG_θ, D_j, G);
7: return clusterMap

8: function cluster(CG_θ, D_j, G)
9:     for each x ∈ D_j do
10:        z ← nearestGridPoint(x, G);
11:        if n_z ∈ CG_{i,θ} then
12:            clusterMap(x) ← i;
13:        end if
14:    end for
15:    return clusterMap;
16: end function
```

A complete cluster guide is defined by: $CG_\theta = \{CG_{i,\theta} |\ i = 1, \ldots, k\}$ *where* k *is the number of clusters found using a given* θ. □

A cluster guide $CG_{i,\theta}$ can be viewed as a contour defining the cluster shape at level θ (an isoline), but, in fact, it only shows the internal grid points and not the actual border of the cluster, which should be determined using the local dataset.

The KDEC-S algorithm is structured in two parts, as discussed in the following.

Local Peer. (cf. Alg. 4.7) The first step is the function negotiate(), which only succeeds if an agreement on the parameters is reached. A distinguished site is chosen as a helper, denoted L_{helper}. Note that the helper does not take part in this phase. In the second step, each local peer L_j computes its local density estimate $\tilde{\varphi}[D_j](z \cdot \tau)$ for each $z \cdot \tau$, with $z \in G$. Using the Definition 4.4, each local peer builds its local sampling set S_j and sends it to the helper. The clustering step (line 6 in Alg. 4.7) is performed as a lookup in the cluster guide CG_θ received from the helper. The function cluster() shows the details of the clustering step. The data object $\mathbf{x} \in D_j$ will be assigned to the cluster i, the cluster label of the nearest grid point z, if the index code of z is in cluster guide i, i.e. $n_z \in CG_{i,\theta}$.

Algorithm 4.8 KDEC-S: Helper

Input: $\mathcal{L}$ (list of peers); θ (density threshold);
Output: CG_θ, a cluster guide a given level θ;
Method:

```
 1: receive S_j from L;
 2: φ̃[D_j] ← recover(S_j);
 3: φ̃[D] ← Σ φ̃[D_j];
 4: CG_θ ← buildClusterGuides(φ̃[D], θ);
 5: send CG_θ to L;

 6: function buildClusterGuides(φ̃[D], θ)
 7:     cg ← {n_z | φ̂_z > θ};
 8:     n ∈ cg;
 9:     CG_{i,θ} ← {n};
10:     i ← 0;
11:     for each n ∈ cg do
12:         if ∃a((a ∈ neighbors(n)) ∧ (a ∈ cg)) then
13:             CG_{i,θ} ← {n, a} ∪ CG_{i,θ};
14:         else
15:             i++;
16:             CG_{i,θ} ← {n};
17:         end if
18:         cg ← cg \ CG_{i,θ};
19:     end for
20:     CG_θ ← {CG_{i,θ} | i = 1, ..., C};
21:     return CG_θ
22: end function
```

Helper. (cf. Alg. 4.8) For a given value of θ, the helper sums up all sample sets and, using Definition 4.5, computes the cluster guides CG_θ. Function buildClusterGuides() in Algorithm 4.8 shows the details of this step.

4.3.2 Complexity Analysis

Time. The dominant part of the processing in KDEC-S is superlinear on the size of the dataset.

Theorem 4.4 *KDEC-S needs $O(|G|M_j + log(k)|D_j|)$ steps, where $|G|$ is the grid size, M_j is the average size of the neighborhood, k is the number of cluster guides and D_j is the set of points owned by peer L_j.*

Proof. The first steps in local peer algorithm (cf. Alg. 4.7) have complexity $O(|G|M_j)$, since the algorithm computes the density for each point z in the grid G. This step uses the subset of points in D_j which are neighbors from z, with average size M_j. Line 3 has its complexity determined by the size of

sampling set S_j, which is a subset of G, i.e., its complexity is $O(|G|)$. Line 5 has complexity $O(k)$. The last step (line 6) has to visit each point in D_j and, for each point, it has to decide its label by searching the corresponding index code in one of the cluster guides. There are k cluster guides. Assuming the look-up time for a given cluster to be $log(k)$ it can be said that $O(log(k)|D_j|)$ is the complexity of the last step.

The helper (cf. Alg. 4.8) will receive at most $|G|$ sampling points from P peers and it needs to reconstruct and sum them up (lines 2 and 3), which takes in the worst-case $O(P|G|)$ steps. Thus, the process of building the cluster guides (line 4) will take $O(|G|)$ steps in the worst case.

Therefore, local Peer requires $O(|G|M_j + log(k)|D_j|)$ to complete, and the helper peer has its time complexity mainly determined by the size of the entire sampling set which requires $O(|G|)$ steps in the worst case. □

Communication. KDEC-S's message size is given by sampling parameters and is independent of the size of the dataset.

Theorem 4.5 *KDEC-S message size is $O(|G|)$, where $|G|$ is the size of the sampling grid.*

Proof. KDEC-S algorithm only uses a few rounds of messages. Each site will have, at most, $|S_j| < |G|$ sampling points (index-codes) to send to the helper site. The helper site has, at most, $|G|$ index-codes to inform back to local agents. In the first round of communication, each site sends one message informing the local sampling S_j set to the helper and one (or more, subsequent) message(s) requesting a cluster guide with some desired θ. Then, the helper answers back, informing the cluster guides, as requested by local peers. Therefore, KDEC-S produces messages of size $O(|G|)$. □

4.3.3 Inference and Collusion Attacks Analysis

It is now necessary to return to the core question: does KDEC-S present any improvement in the privacy level of KDEC scheme? Below, inference attacks scenarios for KDEC-S are analyzed, and, in the subsequent section, the KDEC-S' privacy level is scrutinized.

Single Insider Attack

It is assumed, in this thesis, that the single attacker has a local dataset and knows the parameter values used to compute the mining results since the attacker is part of the mining group. Further, it is assumed that the single attacker does not exchange attacking information with other (possibly malicious) mining peers in the group. When the protocol is complete, the attacker receives, as with all the other members, the set of cluster guides in the form of an index of connected grid points representing the clusters. The protection comes from the loss of information caused by using an approximated kernel function instead of the original kernel. As a result, even the points at the border of any given cluster cannot be reconstructed with arbitrary precision. The precision is defined by the distance between the grid points, i.e., sampling rate τ.

Since KDEC-S produces a cluster map at the end of the mining session, our first step in the privacy analysis is to state how much an attacker can learn from the cluster map alone, without reconstruction. This privacy level is described by the $\mathbf{PR}^{range}$ measure, which is based on the cluster width (cf. Eq. 3.2).

Lemma 4.6 *Let $\mathcal{L}$ be a mining group formed by $P > 2$ peers and attackers working alone without forming collusion groups, i.e. $c = 1$. If $\tau \in \mathbb{R}$ be a sampling rate then the privacy level of KDEC-S with singleton collusion groups without reconstruction is $\mathbf{PR}^{DCP}_{KDEC[1]} = \mathbf{PR}^{range}_{KDEC[1]}(D) \geq \tau$.*

Proof. Assume that $c = 1$, and that each peer only has access to its local dataset and the cluster guides he receives from the helper. The cluster guides, produced by the helper, contain only index codes representing grid points where the threshold θ is reached. From the information in the cluster guides, it is possible to find the grid points for each cluster C_i, which has its width of least m grid points spaced by τ in each dimension. Thus $\mathbf{PR}^{range}_{KDEC-S[1]}(D) = m\tau$, with $m \in \mathbb{N}$. Therefore, $\mathbf{PR}^{range}_{KDEC-S[1]}(D) \geq \tau$, with $m > 1$. □

Another option to determine the privacy level of KDEC-S is to model each point in a cluster as a random variable X with probability density function $f_X(x)$.

Lemma 4.7 *Let $\mathcal{L}$ be a mining group formed by $P > 2$ peers and attackers working alone without forming collusion groups, i.e. $c = 1$. If $\tau \in \mathbb{R}$ be*

a sampling rate then the privacy level of KDEC-S with singleton collusion groups is $\mathbf{PR}^{DCP}_{KDEC[1]}(D) = \mathbf{PR}^{BK}_{KDEC[1]}(D) \geq \tau$.

Proof. Recall that a probability density estimates is not available to parties in the mining group, but only the cluster guides. In this case, the attacker needs to assume that the points in a given cluster follow a uniform distribution, i.e. for a given cluster $\mathcal{C}_i$ let $X_i \sim U(min_i, max_i)$. Clusters guides contain at least one grid point by construction. Each grid point represents a region in the data space of width greater or equal to τ by construction. Thus, each cluster guide represents a cluster with at least width τ. By construction, the cluster width is a multiple of τ. As a consequence, it is possible to model X_i as $X_i \sim U(0, m\tau)$, with $m > 1$. The entropy of X_i as a uniform distribution[5] is $h(X_i) = -\log_2 \frac{1}{(m\tau)} = \log_2 (m\tau)$. Now, using Eq. (3.9) it results in $\mathbf{PR}^{BK}_{KDEC-S[1]}(D) = \min\{2^{h(X_i)}\} = 2^{\log_2 (m\tau)} = m\tau \geq \tau$, for $m \geq 1$ (since no cluster is smaller than τ in any dimension). □

Finally, let us to analyze the privacy level of KDEC-S using pointhunt as reconstruction function.

Lemma 4.8 *For a given sampling rate τ, a single insider attack against KDEC-S using pointhunt results in a privacy level of*

$$\mathbf{PR}^{DCP}_{KDEC-S[1]}(D) = \mathbf{PR}^{rec}_{KDEC-S[1]}(D) \geq \tau$$

.

Proof. Each peer inside the mining group knows all parameters negotiated before the mining session begins. Additionally, it receives the global cluster guides CG_θ at the end of the mining session. Cluster guides, however, do not contain all information needed by pointhunting. Therefore, this analysis only makes sense considering a malicious helper attack. A malicious helper knows sample sets from all peers and cluster guides. Sampling sets contain only density estimates approximations $\tilde{\varphi}[D](z)$, which are computed with Eq. 4.13 and the transformation function $\psi_{f,\mathbf{v}}$ (see Def. 4.12). By lemma

[5] The entropy of a uniformly distributed random variable X $\sim U(0, a)$

$$h(X) = -\int_0^a \frac{1}{a} \log_2 \frac{1}{a} dx = -\left[\left(\frac{1}{a} \log_2 \frac{1}{a}\right) x\right]_0^a = -\log_2 \frac{1}{a} = \log_2 a$$

4.5, for any value $\tilde{\varphi}[D](z)$ in a sampling set, the process of reconstruction using pointhunt cannot locate the original point in an interval smaller than τ. Therefore, $\mathbf{PR}^{rec}_{KDEC-S[1]}(D) \geq \tau$. □

Collusion Attack

The collusion attack involves extra communication between the subgroup forming the malicious peers. It is assumed that attackers have a local dataset and know the parameter values used to compute the mining results. Again, the attacker receives the cluster guides as an index of connected grid points representing the clusters, as with all the other members. Following the conclusion of the normal protocol's execution, colluders may exchange sampling sets S_j to improve reconstruction precision. However, if the collusion group includes the helper, it already knows this information. Thus, this scenario reduces to the malicious helper attack. Similar to the single attack scenario, the protection comes from the loss of information incurred by using an approximated kernel function instead of the original kernel. Once more, the reconstruction precision is given by the sampling rate τ.

Since KDEC-S produces a cluster map at the end of the mining session, we analyze the privacy level as described by the $\mathbf{PR}^{range}$ measure in a collusion scenario.

Lemma 4.9 *Let $\mathcal{L}$ be a mining group formed by $P > 2$ peers and $1 < c \leq p - 1$ malicious peers, which can form collusion groups. If $\tau \in \mathbb{R}$ be a sampling rate then the privacy level of KDEC-S with collusion groups without reconstruction is $\mathbf{PR}^{DCP}_{KDEC-S[c]}(D) = \mathbf{PR}^{range}_{KDEC[c]}(D) \geq \tau$.*

Proof. $\mathbf{PR}^{range}$ needs only the cluster width, which can be derived from cluster guides CG_θ. This information is already known to all peers, at the end of the mining session. Thus this proof reduces to the proof of Lemma 4.6. Thus $\mathbf{PR}^{range}_{KDEC-S[c]}(D) = m\tau$, with $m \in \mathbb{N}$, since Therefore, $\mathbf{PR}^{range}_{KDEC-S[c]}(D) \geq \tau$, with $m > 1$. □

As in the single attack scenario, it is possible to determine the privacy level of KDEC-S with a bounded knowledge measure, modeling each cluster as a random variable X with probability density function $f_X(x)$.

Lemma 4.10 *Let $\mathcal{L}$ be a mining group formed by $P > 2$ peers and $1 < c \leq p-1$ malicious peers, which can form collusion groups. If $\tau \in \mathbb{R}$ be a sampling rate then the privacy level of KDEC-S with collusion groups is $\mathbf{PR}^{DCP}_{KDEC-S[c]}(D) = \mathbf{PR}^{BK}_{KDEC-S[c]}(D) \geq \tau$.*

Proof. Similar to the proof of Lemma 4.7, cluster guides are known to all peers in a mining session. □

In the next, we state the privacy level of KDEC-S using the reconstruction-based measure $\mathbf{PR}^{rec}$ with the function pointhunt.

Lemma 4.11 *Let $\mathcal{L}$ be a mining group formed by $P > 2$ peers and $1 < c \leq p-1$ malicious peers, including the helper, which can form collusion groups. For a given sampling rate τ, a collusion attack against KDEC-S using pointhunt results in a privacy level of $\mathbf{PR}^{DCP}_{KDEC-S[c]}(D) = \mathbf{PR}^{rec}_{KDEC-S[c]}(D) \geq \tau$.*

Proof. Assume the collusion group includes a malicious helper. This collusion group has all parameters values, cluster guides CG_θ, and sample sets sent from peers. All this information is already known to the helper. Because of that, this scenario reduces to a single insider attack carried out by a malicious helper. As a consequence, the proof is similar to the that of Lemma 4.8. Therefore, $\mathbf{PR}^{rec}_{KDEC-S[c]}(D) \geq \tau$, with $c > 1$. □

Outsider Attack

Outsiders need to monitor the communication to collect enough information before starting a reconstruction process. In an extreme case, it is assumed that attackers eavesdrop on all the parameter values used to compute the mining results, intermediate results, and the cluster guides distributed at the end of the mining session. As in the single attack scenario, the protection comes from the loss of information incurred using an approximated kernel function instead of the original kernel. Recall that the reconstruction precision is determined by the distance between the grid points, i.e., the sampling rate τ. However, this is the worst-case scenario for an outsider attack. A milder assumption would be that the attacker does not have access to all information on parameter values. In that case, the precision of an outsider

attack will always be worse or equal to the precision produced by a collusion attack. We expressed this result as follows:

$$\mathbf{PR}^{DCP}_{KDEC-S[0]}(D) \geq \mathbf{PR}^{DCP}_{KDEC-S[c]}(D) \tag{4.17}$$

with $c \geq 1$.

Summary of privacy analysis

Theorem 4.6 *Let $\mathcal{L}$ be a mining group formed by $P > 2$ peers, one of them being the helper, and $c < p$ malicious peers, including the helper, possibly forming a collusion group in $\mathcal{L}$. If $\tau \in \mathbb{R}$ is a sampling rate then $\mathbf{PR}^{DCP}_{KDEC-S[c]}(D) \geq \tau$ for all $c \geq 1$.*

Proof. From Lemmas 4.6, 4.7, 4.8, 4.9, 4.10 and 4.11 it was shown that smallest interval a single point can be located has size of at least τ units in a given cluster dimension, for all privacy measures in all attack scenarios. Therefore, $\mathbf{PR}_{KDEC-S[c]} \geq \tau$, for all $c \geq 1$. □

4.3.4 Experimental Evaluation

KDEC-S was implemented in Java (JDK 1.7), primarily to keep the implementation machine-independent and because the main objective is to provide a proof of concept, as opposed to a performance-oriented implementation.

Datasets. Several datasets were utilized in this batch of experiments. Sample datasets were created, consisting of points generated from a mixture model with four Gaussians, each with a variance of σ^2 =1 in all dimensions. The idea is to simulate a dataset with four clusters (cf. Fig. 4.9). A dataset with 500 points was also generated, built to perform basic tests. Datasets with 5K, 10K, 15K, 20K, 25K, and 30K points were also created to analyze time as a function of the size of the dataset. A dataset with 400 points was also generated, 200 points of which were generated from a Gaussian with $\mu = 0$ and $\sigma^2 = 5$ and 200 points generated around the center with radius $R \sim N(20, 1)$ and angle uniformly distributed from $\sim U(0, 2\pi)$. This dataset is interesting because its clusters are not easily defined (cf. Fig. 4.10). Additionally, a synthetic dataset from [74] was utilized, containing 5000 points distributed in 15 predefined clusters (cf. Fig. 4.11). The spiral dataset from

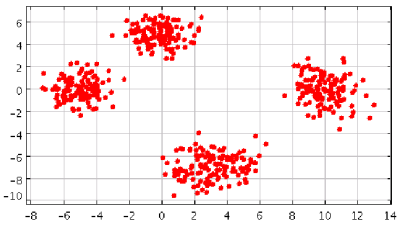


Figure 4.9: *Gaussian dataset: data points generated from a mixture of 4 Gaussians.*

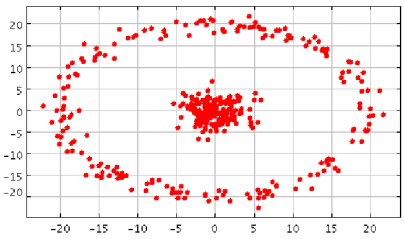


Figure 4.10: *Polar dataset: Points are distributed along an arbitrary shape, with two clusters.*

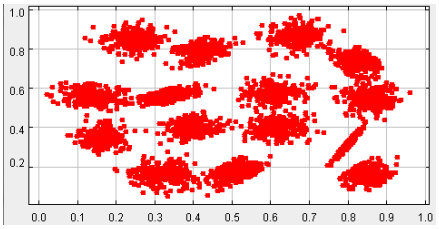


Figure 4.11: *S1 dataset: data points generated from a mixture of 15 Gaussians.*

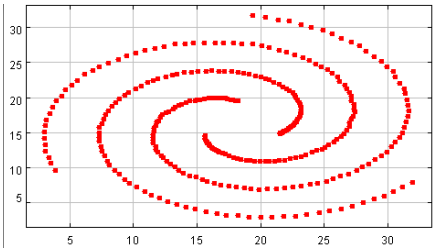


Figure 4.12: *Spiral dataset: data points form three different spirals.*

[36] was also utilized, which contains 312 points and 3 clusters (cf. Fig. 4.12).

Sanity Check. KDEC-S was applied to the datasets with several parameter configurations. In the following, the best parameter setting for each dataset is outlined. For the `Gaussian` dataset with 500 points, the Gaussian kernel with bandwidth $h = 0.5$, *neighborhood* $= 1$, $\tau = 0.5$, grid corners ((-15,-15), (15, 15)), iso step =1, and threshold $\theta = 0.5$ was used. For the `polar` dataset, bandwidth $h = 0.5$, *neighborhood* $= 4$, $\tau = 0.5$, grid corners ((-30, -30),(30,30)) and threshold $\theta = 0.1$ was used. For `s1` dataset, $h = 10^4$, *neighborhood* $= 3$, with grid corners ((0,0), (10^6,10^6)), $\tau = 10^4$, iso step $=$ 1, and threshold $\theta = 2.0$ was utilized. For `spiral` dataset, the kernel Gauss with $h = 0.5$, *neighborhood* $= 1$, corners ((0,0),(100,100)), $\tau = 1$, iso step $= 1$, and $\theta = 0.1$ was employed.

Figures 4.13 to 4.16 depict clusters found on different datasets with KDEC-S. This worked as a sanity check to validate the approach. As is clearly evident, KDEC-S was able to find all of the clusters, even arbitrary-shape clusters as in `polar` and `spiral` datasets.

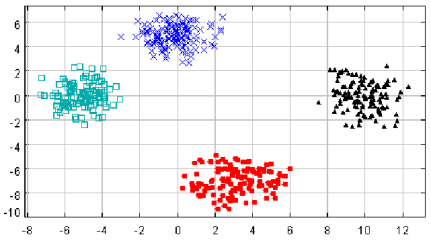


Figure 4.13: *Four clusters found in* `Gaussian` *dataset.*

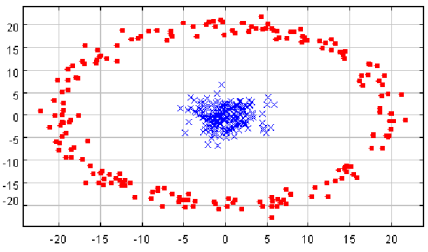


Figure 4.14: *Two clusters with arbitrary shape found in* `polar` *dataset: a central cluster and an external circular cluster.*

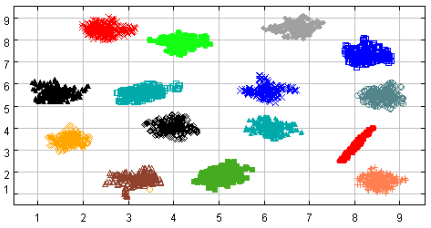


Figure 4.15: *15 clusters found in* `S1` *dataset (colors may be repeated by the plotting software).*

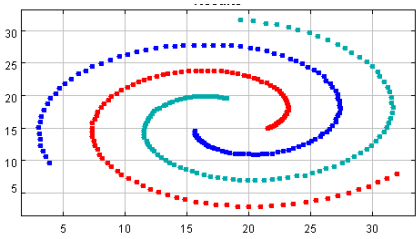


Figure 4.16: *3 clusters found in* `Spiral` *dataset.*

Clustering Error. The mislabeling error was accounted for, considering as correct cluster mapping the one obtained with $\tau_{ref} = \frac{h}{2}$, which represents a sampling rate with minimal information loss. The clustering error in terms of differences between the reference cluster mapping and a new one, considering all pairs of objects in D, is computed as defined in [120], as follows:

$$E = \frac{2}{|D|(|D|-1)} \sum_{i,j \in \{1,2,\ldots,N\}, i<j} e_{ij} \tag{4.18}$$

where $|D|$ is the size of the dataset and e_{ij}, the mislabeling error, is defined as:

$$e_{ij} = \begin{cases} 0 & \text{if } (label(x_i) = label(x_j) \wedge label'(x_i) = label'(x_j)) \quad \vee \\ & (label(x_i) \neq label(x_j) \wedge label'(x_i) \neq label'(x_j)) \\ 1 & \text{otherwise} \end{cases} \tag{4.19}$$

where x_i and x_j represent two different data points in D, $label()$ denotes the reference cluster label assigned to objects, i.e. the label found using τ_{ref}, and $label'()$ denotes the new label found with $\tau > \tau_{ref}$. Given two data

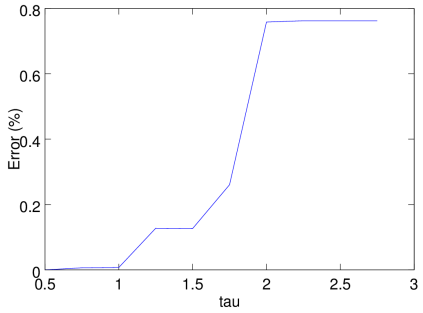


Figure 4.17: *Error rate in* `Gaussian` *dataset in function of* τ*, with* $h = 0.5$*.*

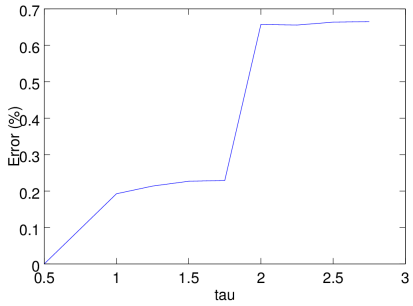


Figure 4.18: *Error in* `Polar` *dataset in function of* τ*, with* $h = 0.5$*.*

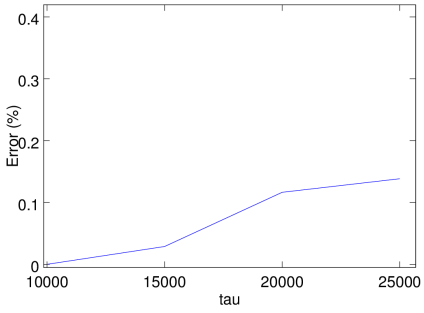


Figure 4.19: *Error in* `S1` *dataset in function of* τ*, with* $h = 10^4$*.*

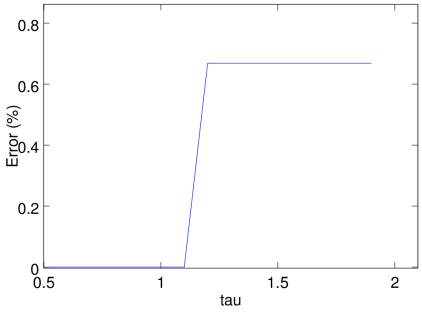


Figure 4.20: *Error in* `Spiral` *dataset in function of* τ*, with* $h = 0.5$*.*

points x_i and x_j, the error is 0 if their relationship does not change, i.e. they have the same label after and before or have different label after and before. Otherwise, 1 error is counted.

The experiments suggest that the value of τ up to h can be set with no error in the clustering results generated by KDEC-S (cf. Figs. 4.17 to 4.20). In the `Gaussian` dataset, significant error rate appears just after $\tau = 2.5h$ and in the polar datasets, the rate of error becomes large after $\tau = 2h$. In general, it was observed that, as the value of sampling rate τ reaches values beyond the kernel bandwidth, there is an increase in the error rate because more points are considered outliers. With the Gaussian kernel, it is known that the kernel goes to zero around $3 * h$, therefore meaning that the iso-levels summation does not reach the given threshold. Consequently, the corresponding grid point is omitted from the cluster guides, i.e., it is considered an outlier.

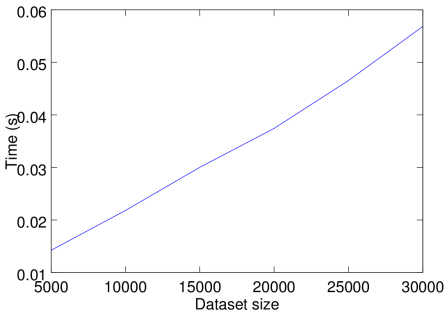


Figure 4.21: *Running time (in sec) as a function of dataset size for generating estimates at a local peer. With neighborhood* $= 1$ *and* $h = 1$

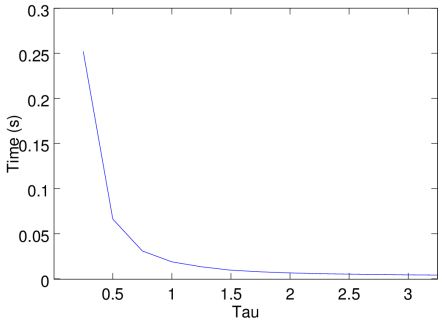


Figure 4.22: *Running time (in sec) as a function of privacy parameter* τ*.* `Gaussian` *dataset (with neighborhood* $=$ 1 *and* $h = 0.5$*)*

Performance. Empirical results also confirm theoretical performance results (cf. 4.3.2). Estimation only takes a fraction of the dataset and seems almost linear at the local peer. Synthetic datasets with different sizes were used, ranging from 5.000 to 30.000 points, with *neighborhood* $= 1$ and bandwidth $h = 1$ (cf. Fig. 4.21). The privacy parameter plays a crucial role in the performance since it determines the number of points in the sampling grid. Thus, the small values of τ require more computational effort since the grid has more points. Figure 4.22 illustrates the results of experiments for different values of τ in the `Gaussian` dataset, with *neighborhood* $= 1$ and $h = 0.5$. Therefore, for the sake of both privacy and time, this study recommends values of τ greater than the kernel bandwidth, i.e., $\tau > h$.

4.4 Related Work and Discussion

Previous sections presented KDEC-S and how it handles privacy. This section compares the KDEC-S approach with other privacy-preserving distributed clustering algorithms. We first compare KDEC and KDEC-S and then discuss Generative Models and SMC k-means clustering.

KDEC

KDEC and KDEC-S are based on density estimates, reducing the clustering problem to finding local maxima in a density function. However, KDEC-S works on a modified version of the density estimate (cf. Sec 4.3.1), one which

does not carry all the information needed to reconstruct the original dataset but still has enough information to find clusters. Since KDEC and KDEC-S have already been studied in different attack scenarios in previous sections, the comparison between the two approaches is, quite simply, stated in the following theorem.

Theorem 4.7 *Let $\tau \in \mathbb{R}$ be a sampling rate parameter. For a given value of τ we have $\mathbf{PR}^{DCP}_{KDEC-S[c]} > \mathbf{PR}^{DCP}_{KDEC[c]}$, for all $c > 0$.*

Proof. By Lemma 4.6 we know that $\mathbf{PR}_{KDEC-S[c]} \geq \tau$ and by Lemma 4.4 we have $\tau \gg \mathbf{PR}_{KDEC[c]}$, for all $c > 0$. Therefore, the assertion holds. □

The Theorem 4.7 states that with the same sample rate τ, KDEC-S will always provide more privacy than KDEC, no matter how many parties are malicious. In fact, by Lemma 4.4, the privacy level of KDEC is much less than τ. On the other hand, τ is a lower bound for KDEC-S' privacy level.

Both algorithms make no assumptions concerning trusted third parties. Therefore, collusion groups may contain up to $P - 1$ parties, with P parties in the mining group. With regards to time complexity, KDEC-S has the same worst-case complexity as KDEC, which is $O(N \log N)$ on the size of the dataset. Both algorithms require $O(|G|)$ space, where $|G|$ is the size of the sampled density estimate (or sample sets in KDEC-S).

KDEC-S, just like KDEC, needs only two rounds of message exchange to complete the protocol. Each message has size $O(|G|)$ on the number of samples used in the process.

In summary, KDEC-S performs to the same level as KDEC, only with the added benefit of an increased level of control over privacy preservation.

Generative Mixture Models

The generative models approach represents an excellent technique for privacy-preserving distributed data clustering since the privacy level can be controlled by both the number of models in the mixture and the range of cluster modeled (cf. Sec. 3.4.3). Therefore, even if the central peer is malicious, the user can control the desired level of privacy. Unfortunately, the generative models approach loses information about the natural separation in the dataset. For example, consider a dataset with k natural clusters. Any mixture model with less than k base models will lose information about

the natural separation of data points. In general, the number of clusters is the very information one is looking for when using a clustering algorithm. KDEC-S also allows the user to control the desired level of privacy without first asking the number of clusters since it follows a density-based approach to discover clusters. Consequently, the original information about the natural clusters is preserved.

Generative models clustering requires no trusted parties since only models are transmitted, and unless models are too detailed, data reconstruction is bounded to a given level. For the same reasons, KDEC-S also assumes no trusted parties.

Theorem 4.8 *For a given dataset D and assuming possible collusion, $c > 1$, KDEC-S may provide the same amount of privacy level as generative models with k components:*

$$\mathbf{PR}^{DCP}_{KDEC-S[c]}(D) \approx \mathbf{PR}^{DCP}_{GM[c]}(D)$$

The privacy level, $\mathbf{PR}^{DCP}$, was defined in Section 3.3.1 (cf. Def. 3.4)

Proof. To show that the assertion holds, we need to show that it is always possible to configure KDEC-S to provide the same level of privacy as generative models. The key to this proof is to choose τ in $KDEC-S$ that provides the same privacy as the cluster range in generative models. Thus, we must pick $\tau = \min|\max C_i - \min C_i|$, with $\max C_i$ and $\min C_i$ being the extreme values in a cluster. It is always possible to compute $\min C_i$ and $\max C_i$ for a bounded generative model, such as a uniform distribution, since these extremes are well defined as part of the model. For unbounded models, like Gaussians, it is possible to pick points defining a high confidence interval, like $(\mu - 3\sigma, \mu + 3\sigma)$, which covers more than 99.73% of the true range (by the three-sigma rule of normal distributions). Therefore, for a given k exists a value of τ that makes KDEC-S provide nearly the same amount of privacy provided by generative models with k components. □

The time complexity of Merugu's generative models is based on two main steps: (i) building the models and (ii) clustering each data point locally. The model building step follows an optimization approach involving several iterations until the algorithm converges. At the local site, each iteration builds a candidate model until a given stop criterion is reached. KDEC-S

builds sampling sets locally in $O(|G|n)$, where $|G|$ is the size of the sampling grid and n is the number of neighbors of each sampling point, with $n \ll |D_j|$. Since the generative model approach is iterative, it needs several passes on the dataset to build candidate models until convergence is reached. In contrast, KDEC-S only needs one pass to build local sampling sets, i.e., KDEC-S has much lower time complexity to build its clustering model than generative models. When it comes to the clustering step, both generative models and KDEC-S have the same time complexity.

The generative models approach has space complexity $O(|F|)$, where $|F|$ denotes the size of the model used to represent each cluster. The authors propose using Gaussians, which have little representation costs – typically only the mean and variance. KDEC-S needs more space since it stores the density for each point in the sampling grid, i.e., it is linear in the size of the grid $O(|G|)$.

Just as in the case of KDEC-S, only two rounds of messages are necessary to complete the generative models protocol. First, each peer sends local models to the aggregator, and then the aggregator sends the global model to each peer. In the worst-case scenario, the message has size $O(F)$, where F is the size of the generative model.

More recent approaches to model a Gaussian mixture from data, e.g., [22, 125], still needs several rounds to converge. Moreover, Gaussian Mixtures approaches fail to discover arbitrary-shaped clusters, favoring spherical ones similarly to Merugu's approach. On the other hand, density-based algorithms like KDEC-S can find arbitrary-shaped clusters.

Secure Multiparty k-Means

Several SMC algorithms for clustering were discussed in Sec. 2.3.4 and its security properties under inference attacks were discussed in Sec. 3.4.1. These algorithms differ from KDEC-S in its underlying privacy-preserving approach, focusing its security heavily on cryptographic techniques, aiming to ensure that an outsider will not learn anything by eavesdropping on the conversation. In the following, we compare KDEC-S with a representative algorithm of the SMC-based clustering approaches, namely Vaydias' approach [213], here on referred to as VC-k-Means.

VC-k-Means assumes at least three trusted parties to run this protocol. These trusted parties play an essential role in the SMC protocol, ensuring

no other parties will learn anything beyond the mining results. Under collusion, however, the attacker may reconstruct all of the data points from the remaining parties in the mining group. KDEC-S assumes there are no trusted parties.

Theorem 4.9 *With no trusted parties, KDEC-S provides more privacy than VC-k-Means:*

$$\mathbf{PR}^{DCP}_{KDEC-S[c]} > \mathbf{PR}^{DCP}_{VCkMeans[c]}$$

with $c > 1$. *The privacy measure* **PR** *was defined in Section 3.3.1 (cf. Def. 3.4).*

Proof. Privacy in KDEC-S is given by the sampling rate τ, as shown in Lemma 4.6. According to Lemma 3.11 $\mathbf{PR}^{DCP}_{VCkMeans[2]} \approx 0$. Therefore, if we choose any $\tau > 0$, we have that the assertion holds. □

The message size in SMC k-Means is $O(Npk)$, where N is the number of parties, and k is the number of clusters. The total number of rounds is $O(P + k)$. KDEC-S requires a constant number of rounds and message size defined by the size of sampling sets, independent from the number of parties or the number of clusters.

Other Approaches

Shen and Li [184] proposed an information-theoretical approach to distributed clustering. They assume a peer-to-peer network where each node solves a local clustering problem and updates its neighbors. The clustering problem is defined as a linear program that maximizes the mutual information between cluster labels and data points. This general optimization framework is instantiated as two concrete algorithms, one using a linear discriminative model and another using a discriminative kernel model. With low communication, local clusters are formed based on global information spread through the network. This work is closely related to KDEC and KDEC-S since they also use statistical information in the sampled form to keep low communication costs. However, when compared to KDEC and KDEC-S, the linear program needs several rounds of iterations to converge, while KDEC and KDEC-S only need one round of messages. When it comes to privacy, the authors do not investigate how the algorithm would behave under inference attacks and do not investigate how much privacy this approach does provide.

Liu et al. [134, 135] proposed a distributed version of DBSCAN with privacy-preserving guarantees. The authors use Paillier encryption to develop a secure multiplication protocol to compute the distance between two points. The theoretical analysis proves the efficiency and security of the proposed solution. However, no experimental results were presented. KDEC and KDEC-S are also based on density, but instead of using density to discover a density path between core points, our approaches assign points to density regions by hill climbing on the density estimates. Moreover, KDEC and KDEC-S are not iterative and need only one pass over the local data to estimate density and one pass to assign labels.

Almutairi et al. [8] generates local chain distance matrices (CDM) and uses Paillier encryption to produce an order-preserving encrypted version of it. A central semi-trusted party coordinates the encryption process and runs a modified DBSCAN algorithm to find global clusters. Original data is never sent in plain text, and the output is decrypted only at local parties to find local clusters. The scheme is linear with the number of parties. Nevertheless, it presents a computational overhead to encrypt the data at local parties. KDEC-S does not use encryption, but an approximation of density estimates that limits original data reconstruction by design. Therefore, KDEC-S does need to assume a fixed semi-trusted central server.

Bozdemir et al. [27] developed a two-party full private DBSCAN based on the ABY2 secret sharing framework without encryption. Data from all parties and parameters are split into shares and processed by two non-colluding cluster servers. The results are precise compared with the original DBSCAN. However, there is a computational overhead to split original data into shares during private distance computation in the innermost loop. Moreover, the protocol requires several rounds of messages to compute all distances. By comparison, KDEC-S does not assume any central server and has better running time complexity with only a few rounds of messages.

Wang et al. [217] developed a distributed spectral clustering algorithm (FMSC) based on homomorphic encryption. FMSC needs a trusted central server to generate aggregate statistics and two or more data providers holding a different view of the data (heterogeneous partition or vertical partition). The central server uses additive homomorphic encryption to aggregate statistics without decryption. The central server adds noise to the statistics before sending them back to data providers to avoid data leakage. It is an

iterative algorithm that requires many rounds of messages until convergence. One main drawback compared with KDEC-S is the assumption of a central trusted party as KDEC-S assumes no trusted central party. Furthermore, homomorphic encryption implies a computational overhead when compared to our approach.

Xia et al. [227] follows a random perturbation approach and presents a distributed k-means algorithm. The proposed algorithm assumes a set of users providing data to an untrusted server. They first add noise to local data to achieve differential privacy and collaboratively build a set of global centroids. This algorithm is iterative and takes several rounds until convergence. It does not assume a trusted third party and benefits from the simplicity of its base algorithm k-means. However, unlike KDEC and KDEC-S, it needs to known the number of clusters a priori. Moreover, it is not able to find arbitrary shaped clusters.

4.5 Summary

This chapter showed that kernel-based density estimation is a feasible technique for privacy-preserving distributed data clustering. First, it provides a generic method for finding clusters with no assumption about the underlying probabilistic model or the number of clusters. Secondly, it allows for low communication costs since the density estimates are much smaller than the original dataset from which it is computed. Finally, since density estimates are additive, it is possible to utilize them in distributed algorithms. This chapter also showed that one attacker could reconstruct original sensitive data during a KDEC mining session under certain circumstances. To overcome this form of attack, we presented KDEC-S, which uses a density approximation instead of the original density estimate in constructing a density landscape of the union of the datasets. This approach enables the data holders to control the privacy level by choosing the sample rate and a suitable kernel transformation that meets their privacy needs.

Compared to generative models, KDEC-S has similar performance and allows the user to control how much privacy preservation is needed. The number of mixture models used to generate the global model can control the privacy level provided by generative models. In contrast, KDEC-S does not need the number of clusters a priori and controls privacy via a sampling

parameter τ.

Differently from VC-k-Means, KDEC-S assumes no trusted parties. Notice that if the assumption of trusted parties is not met, the privacy of the VC-k-Means algorithm drops radically since the colluding parties can reconstruct the original data points. KDEC-S trades off privacy level for communication and time efficiency, while SMC protocols (in general) take an all or nothing approach to privacy. However, secure multi-party computation does not avoid attacks by insiders.

VC-k-Means has low time complexity at local clusters in each iteration since it is a distributed computation. That contrasts with KDEC-S, which computes local models at each site and sends them to other parties.

Table 4.1 summarizes the comparative analysis of the algorithms discussed in previous sections.

	KDEC	**KDEC-S**	**Gen. Models**	**Vaidya's k means**
Reference	[118]	Sec. 4.3	[149]	[213]
Clustering approach	density-based	density-based	generative models	k-means
Privacy with outsider $\mathbf{PR}_{[0]}^{DCP}$	≈ 0	τ	$\min\{\max C_i - \min C_i\}$	∞
Privacy with single insider $\mathbf{PR}_{[1]}^{DCP}$	≈ 0	$\tau > 0$	$\min\{\max C_i - \min C_i\}$	$\min\{\max C_i - \min C_i\}$
Privacy with collusion $\mathbf{PR}_{[c>1]}^{DCP}$	≈ 0	τ	$\min\{\max C_i - \min C_i\}$	≈ 0
Num. of trusted parties required	0	0	0	2
Time Complexity [a]	$O(Nlog(N))$	$O(Nlog(N))$	$O(N+F)$	–
Space Complexity [b]	$O(G)$	$O(G)$	$O(kF)$	$O(F)$
Number of rounds [c]	2	2	2	$O(P+k)$
Message size	$O(G)$	$O(G)$	$O(kF)$	$O(Nrk)$

Table 4.1: *Summary of comparative analysis of privacy from different PP-DDC algorithms.*

[a] We simplified complexity expressions. In time, space and message complexity, N is the size of the dataset and G is the size of the sampled data, and k is the number of clusters, and r denotes the number of rounds.

[b] F denotes the model size. More precisely, it denotes the minimum number of bits necessary to describe a given model.

[c] P denotes the number of parties.

Chapter 5

Privacy-Preserving Distributed Time Series Mining

> "Time is free, but it is priceless. You can't own it, but you can use it. You can't keep it, but you can spend it. Once you've lost it, you can never get it back."
>
> (Harvey Mackay)

The previous chapter discussed privacy issues in the context of distributed data clustering. In this chapter, we continue the investigation of privacy issues in the context of distributed time series mining.

Time series data is a prevalent type of data, one that is recurrent in practically every field of human activity, spanning such wide-ranging fields as science, industry, business, and medicine, among others. Mining this kind of data has been an active area of research [69, 205] and has produced several specialized algorithms including association rules discovery [166], classification [17], clustering [158], anomaly detection [183], motif discovery [205, 235], to name but a few.

Many time series mining algorithms have pattern-related problems as core activities. A pattern is an "interesting" subsequence of a time series. Here, the term "interesting" may contain a different meaning, depending on the pattern problem one wants to address. A well-investigated problem lies in finding patterns similar to a given subsequence, which is known as the *query-by content* problem. Another class of pattern problems occurs when the pattern is unknown. In this case, if the goal is to identify a set of previously unknown frequently occurring patterns, the problem is called *motif discovery* [235]. On the other hand, the problem of finding non-frequent

patterns is defined as *surprise, discord, outlier* or *anomaly* detection [25]. This thesis is only concerned with *motif discovery*, i.e., finding previously unknown frequent patterns.

There are several studies on how to find patterns in time series. In a seminal study, Lin et al. [128] proposed EMMA, a discretization-based algorithm that generates a string of symbols for each subsequence in the time series and stores them in a lookup table. By counting the number of identical strings, EMMA identifies the most frequent subsequences of a given time series. The approach employs a technique called ADM to avoid computing a full distance matrix. However, it requires access to the original data points to compute the distance between subsequences. Chiu et al. [42] address the scalability limitations of EMMA and its inability to discover motifs in the presence of noise and propose a probabilistic approach.

Minnen et al. [151] formulate pattern discovery as a problem of locating regions of high density in the space of all subsequences of size n from a time series. Mueen et al. [153] introduced the MK algorithm to find the exact closest pair of subsequences in massive time series databases. MK can be used as a primitive to find all frequent patterns by finding all subsequences similar to the first closest pair found on the series and applying it $k-1$ times until k patterns have been found. Matrix profile [5, 235] computes all distances between subsequences of a given size in a time series and has been shown to find the closest pair of subsequences even faster than the MK algorithm.

Notice that none of the current state of the art alternatives was designed to address a distributed data context. Additionally, none of these approaches address privacy issues. Hence, it is still necessary to investigate pattern discovery without compromising data privacy when time series are distributed.

In the following sections, the problem of frequent pattern discovery in distributed time series is formalized, privacy and quality metrics for time series data are discussed, and proposed algorithms are presented. The related work in the broader field of privacy-preserving distributed time series is surveyed at the end of this chapter.

5.1 Distributed Pattern Discovery in Time Series

There are numerous application scenarios for distributed time series mining. For example, disease screening for public health from a network of contributing hospitals [138], dynamic power load in smart grid applications with smart meter data [222] or real-time data analytics from smart devices [224], all generate distributed time series data.

In all example scenarios, the underlying problem can be stated informally as the task of *finding all frequent, or surprise, patterns given the union of the individual data owned by the different peers.* Additionally, considering that participating peers may be malicious (as recalled from the previous chapter), preserving the privacy of sensitive data is imperative.

Usually, the value of each point in a time series is the sensitive information to be protected. For example, if a time series represents the total purchase made from a given credit card per month, the credit card holder might want to keep each point undisclosed from other people. Moreover, other aspects of time series may also be considered sensitive such as peaks, trends, and periodicity [245]. Additionally, it is worth stressing that the identity of data owners and data holders is also sensitive. For example, if the data points represent the doctor-patient mortality ratio at hospitals, no doctor or hospital wants to be identified as a high mortality ratio doctor or institution.

In the following, we introduce the notation and define the problem of privacy-preserving pattern discovery from a distributed time series. Then, we present algorithms to solve the said problem. Finally, a theoretical analysis is presented based on the algorithms' performances and privacy properties.

Definition 5.1 (Time Series) *Let $f : \mathbb{N} \rightarrow \mathbb{R}$ be a function from time stamps to reals. We define a* time series *T as a finite sequence of reals x_t coming from some measurement function f. We denote a time series as $T = \langle x_1, x_2, x_3, \ldots, x_m \rangle$, with $x_t = f(t)$, $1 \leq t \leq m$. We denote the length of the time series T by $|T|$. Furthermore, we denote by $T[t]$ the element x_t of T, i.e $T[t] = x_t$.*

□

Definition 5.2 (Subsequence) *Let $T = \langle x_1, x_2, \ldots, x_m \rangle$ be a time series of length $m \geq 1$. We say that S is a* subsequence *with size n starting at time t of T, denoted $S_t^n \sqsubseteq T$, if $S = \langle x_t, \ldots, x_{t+n-1} \rangle$, for given integers*

$1 \leq n \leq m$ *and* $1 \leq t \leq m-n+1$. *If* t *and* n *are clear from context, we may simply write* S, *or* S_t, *instead of* S_t^n. *We use the notation* $T[i:j]$ *to indicate a subsequence from* T *starting at position* i *and ending at position* j, *with* $i \leq j \leq m$. *In particular,* $T[i:i]$ *denotes the single element subsequence* $\langle x_i \rangle$. *Given a subsequence* S_t^n *of* T, *we access its elements by* $S_t^n[i]$, $1 \leq i \leq n$, *i.e.* $S_t^n[i] = T[t+i-1] = x_{t+i-1}$. □

Example 5.1 *Given a time series* $T = \langle 1, 2, 2, 8, 0, 5, 3, 4, 1 \rangle$, *a subsequence size* $n = 4$ *starting at* $t = 3$ *is* $S_3^4 = T[3:6] = \langle 2, 8, 0, 5 \rangle$, *a subsequence size* $n = 6$ *starting at* $t = 2$ *is* $S_2^6 = T[2:7] = \langle 2, 2, 8, 0, 5, 3 \rangle$, *a subsequence size* $n = 1$ *starting at* $t = 8$ *is* $S_8^1 = T[8:8] = \langle 4 \rangle$. *Additionally,* $S_3^4[1] = 2$, $S_3^4[2] = 8$, $S_8^1[1] = 4$, *and so on.*

Next, we define an occurrence set as the set of time stamps where subsequences start. This definition is constructive, showing how to build the occurrence set of a given subsequence set.

Definition 5.3 (Occurrence Set) *Let* $T = \langle x_1, \ldots, x_m \rangle$ *be a time series and let* $\mathcal{S}^n = \{S_t^n\}$, $1 \leq t \leq m-n+1$, *be a set of subsequences from* T *of equal length* n *with* $S_t^n = \langle x_t, \ldots, x_{t+n-1} \rangle$. *The occurrence set, given* $\mathcal{S}^n$, *is the set of indices* $I = \{t : S_t^n \in \mathcal{S}^n\}$. □

Since the subsequences in $\mathcal{S}^n$ all have the same length n, for each $t \in I$ there is a unique $S_t^n \in \mathcal{S}^n$ such that $S_t^n = \langle x_t, \ldots, x_{t+n-1} \rangle$. An occurrence set marks the beginning of each sequence in $\mathcal{S}^n$.

Example 5.2 *Let* $T = \langle 1.3, 2.1, 5.8, 3.9, 0.1, 0.6, 2.7, 2.2, 0.3, 4.1 \rangle$ *and a set of subsequences* $\mathcal{S}^3 = \{\langle 1.3, 2.1, 5.8 \rangle, \langle 0.1, 0.6, 2.7 \rangle, \langle 2.7, 2.2, 0.3 \rangle\}$, *and sequence length* $n = 3$. *The corresponding occurrence set is* $I = \{1, 5, 7\}$, *the position where each subsequence starts (cf. Fig. 5.1).*

Definition 5.4 *An occurrence set* I *of a set of subsequences* $\mathcal{S}^n$ *in a time series* T *is non-overlapping if and only if*

$$\min\{|i-j| : i \in I, j \in I,\ i \neq j\} \geq n,$$

where n *is the length of each* $S \in \mathcal{S}^n$. □

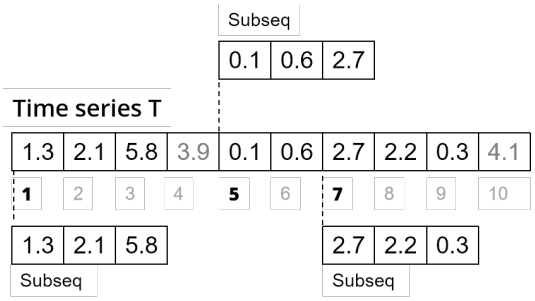


Figure 5.1: *Occurrence set* $\{1, 5, 7\}$ *relative to given subsequences in a time series.*

Example 5.3 *Let* T *as in Example 5.2 and all sequences of length* $n = 3$*. Assume the set of subsequences* $\mathcal{S}^3 = \{\langle 1.3, 2.1, 5.8\rangle, \langle 3.9, 0.1, 0.6\rangle\}$*. The corresponding occurrence set* $I = \{1, 4\}$ *is non-overlapping, since* $|1 - 4| \geq n = 3$*, i.e, the subsequences do not share any elements. On the other hand, with the set of subsequences* $\mathcal{S}^3 = \{\langle 2.1, 5.8, 3.9\rangle, \langle 3.9, 0.1, 0.6\rangle\}$ *the resulting occurrence set* $I = \{2, 4\}$ *is overlapping, since* $|2 - 4| = 2 \ngeq n = 3$*. This means that at least one element is shared among the subsequences. In this case, the element* $T[4] = 3.9$ *is shared by two sequences.*

Definition 5.5 (Distance) *Let* $d : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}$ *be a function measuring the* distance *between two vectors in* $\mathbb{R}^n$*, identified here as subsequences of a time series.* □

Example 5.4 *A well-known distance function is the Euclidean distance, defined as*

$$d(S, Q) = \sqrt{\sum_{i=1}^{n} (S[i] - Q[i])^2}$$

given two subsequences S *and* Q *from* T*, where* n *is the length of subsequences.*

Definition 5.6 (Match) *Let* T *be a time series with size* m*,* d *a distance function, and threshold* $r \in \mathbb{R}$*. Given a query* $Q = \langle q_1, \ldots, q_n\rangle$ *with* $q_i \in \mathbb{R}$*,* $1 \leq i \leq n < m$*, a* match *for* Q *with radius* r *is a subsequence* $S \sqsubseteq T$ *such that* $|S| = n$ *which satisfies* $d(Q, S) \leq r$*.* □

Intuitively, the threshold r defines a ball of radius r around Q in the $\mathbb{R}^n$ space. Therefore, matches are subsequences inside the ball defined by r (cf. Figure 5.2).

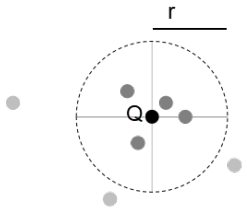


Figure 5.2: *All subsequences from T that are inside a ball radius r are matches to Q, located at the center.*

Example 5.5 *Given a time series* $T = \langle 1.1, 2.1, 2.9, 8.3, 0.1, 5.4, 3.7, 4.2, 1.7, 2.6, 8.1, 0.2, \rangle$, *a query* $Q = \langle 3.0, 8.0, 0.0 \rangle$, *a threshold* $r = 1.0$ *and* d *is the Euclidean distance. The subsequence* $S_3^3 = T[3:5] = \langle 2.9, 8.3, 0.1 \rangle$ *occurring at time stamp* $t = 3$ *has distance* $d(Q, S_3^3) = 0.33166$. *Further,* $S_{10}^3 = T[10:12] = \langle 2.6, 8.1, 0.2 \rangle$ *starting at time stamp* $t = 10$ *has distance* $d(Q, S_{10}^3) = 0.45826$. *Therefore, both* S_3^3 *and* S_{10}^3, *with occurrence set* $I = \{3, 10\}$, *are matches for* Q *in* T *with* $r = 1.0$.

The definition of a match helps us to capture the notion of pattern.

Definition 5.7 (Pattern) *Given a time series T, a distance function d, a radius r and length n, a* pattern *is a pair $(\mathcal{S}^n, I)$, where $\mathcal{S}^n$ is a set of subsequences of length n from T such that any two elements of $\mathcal{S}^n$ are a match for each other, for d and r, and I is non-overlapping occurrence set for $\mathcal{S}$ in T.* □

A pattern is a set of subsequences from time series T similar to each other w.r.t. distance function d and radius r.

The non-overlapping requirement aims to avoid *trivial matches*. A trivial match starts only a few points before or after a true match for a given query Q. Trivial matches occur when the sliding window is large enough so that one point in time does not amount to a difference from Q greater than the radius r. Figure 5.3 shows a given query Q and its best matches, all differing by one single point in time. Trivial matches may falsely indicate a frequent subsequence [42, 128, 153]. This constraint helps us to eliminate these false positives.

Definition 5.8 (Neighborhood) *By* neighborhood *of a subsequence S of length n from a time series T with respect to a distance function d and a radius $r \in \mathbb{R}$, we mean a non-overlapping occurrence set for subsequences in*

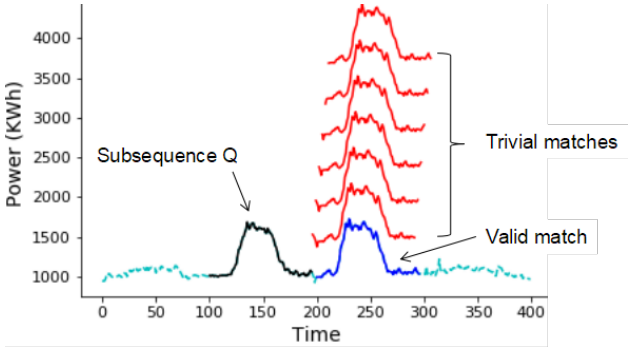


Figure 5.3: *Trivial matches for a given subsequence Q may falsely indicate a frequent subsequence.*

$\mathcal{S}^n$ of T, all with the same size n, such that every element of $\mathcal{S}^n$ is a match for S, for d and r. Further, S is in its own neighborhood, by default, since each subsequence S of T is a match for itself. We denote the neighborhood by $Neigh_{[T,d]}(S,r)$. When the parameters are clear from context, we use the simplified notation $Neigh(S)$ or $Neigh(S,r)$. □

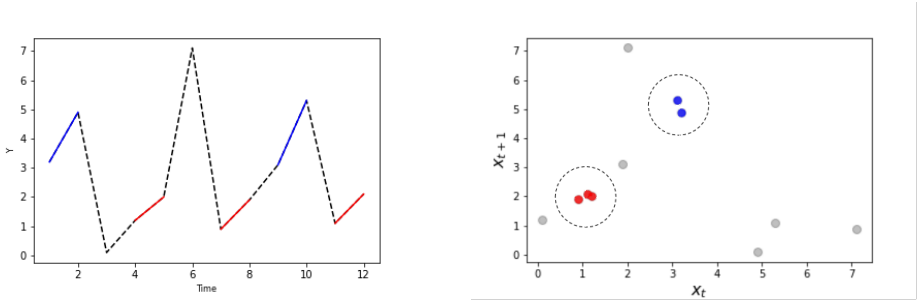
Example 5.6 *Given a time series $T = \langle 1.1, 2.1, 2.9, 8.3, 0.1, 5.4, 3.7, 4.2, 1.7, 2.6, 8.1, 0.2, \rangle$, with subsequences of size $n = 4$. Consider the threshold $r = 1.0$ and d is the Euclidean distance. For subsequence $S_2^4 = T[2:5] = \langle 2.1, 2.9, 8.3, 0.1 \rangle$ the neighborhood $Neigh(S_2^4)$ is the occurrence set $\{2, 9\}$, which refers to subsequence set $\{\langle 2.1, 2.9, 8.3, 0.1 \rangle, \langle 1.7, 2.6, 8.1, 0.2 \rangle\}$. Subsequence S_2^4 is included by default (self match) and S_9^4 is included because it is close to S_2^4. Indeed, the Euclidean distance between them is $d(S_2^4, S_9^4) = 0.54772$, which is less than $r = 1.0$.*

Definition 5.9 (Rank) *By* rank of a pattern *$P = (\mathcal{S}^n, I)$, from a time series T with distance function d, radius $r \in \mathbb{R}$, we mean the cardinality of its occurrence set I, i.e. $rank(P) = |I|$.* □

Definition 5.10 (k most frequent patterns) *By set of the* k most frequent patterns *of length n of a time series T, we mean a set of size k of patterns $(\mathcal{S}, I)$ of T having the highest k ranks with respect to a distance function d and radius r.* □

Example 5.7 *Let a time series $T = \langle 3.2, 4.9, 0.1, 1.2, 2.0, 7.1, 0.9, 1.9, 3.1, 5.3, 1.1, 2.1 \rangle$, with subsequences of length $n = 2$. We have the following subse-*

quences $S_1 = \langle 3.2, 4.9 \rangle$, $S_2 = \langle 4.9, 0.1 \rangle$, *and so on, until* $S_{11} = \langle 1.1, 2.1 \rangle$. *With Euclidean distance and radius* $r = 1$, *we have two sets of non-overlapping subsequences that are matches to each other. The first one is* $\mathcal{S}_1 = \{S_1, S_9\}$ *with occurrence set* $I_1 = \{1, 9\}$ *and* $\mathcal{S}_2 = \{S_4, S_7, S_{11}\}$ *with occurrence set* $I_2 = \{4, 7, 11\}$. *Therefore we have two patterns* $P_1 = (\mathcal{S}_1, I_1)$ *and* $P_2 = (\mathcal{S}_2, I_2)$. *The cardinalities of the occurrence sets are* $|I_1| = 2$ *and* $|I_2| = 3$. *Therefore, the rank of* P_1 *is* 2 *and the rank of* P_2 *is* 3. *If we set* $k = 1$, *the* k *most frequent patterns is given by the set* $\mathcal{P} = \{P_2\}$. *By the same token, if* $k = 2$, *then* $\mathcal{P} = \{P_2, P_1\}$. *Figure 5.4(a) shows the time series with highlighted pattern occurrences. Since subsequences in this example have length* $n = 2$, *we can plot them in a 2d graph, as shown in Figure 5.4(b).*



(a) Pattern P_1 (in red) has three occurrences and pattern P_2 has two occurrences (in blue). Other subsequences are plotted in dashed line.

(b) Occurrences of two patterns plotted in 2d. Subsequences in light gray are not close enough to each other to form any pattern.

Figure 5.4: *A toy time series with pattern occurrences in highlight.*

With the previous definitions, the problem can now be stated formally.

Underlying Architecture. The basic scenario defined in Section 2.3.3 is assumed, where there are a group sites L_i, $i = 1, \ldots, P$, in the mining group. Each party L_i holds a mining and a data agent responsible for participating in a mining session and accessing local datasets D_i, respectively. Furthermore, we assume that the time series data collected at different sites refer to the same variable and has the same time spacing, i.e., data is horizontally distributed among the parties. Datasets are sensitive and should not be disclosed to other parties. Each party may set a local privacy threshold indicating the minimum amount of privacy required to join a specific mining session. Agents are organized into a pure peer-to-peer network, meaning

that each party may act as an initiator or as an arbitrary party in any given mining session. It is also assumed that agents are semi-honest, i.e., they follow the protocol but are curious enough to try to discover any sensitive data from other parties whenever possible.

Problem 5.1 (PP-DPDTS) *Given an integer k, and a set of sites $\mathcal{L} = \{L_i\}_{1 \leq i \leq P}$, each of them with a local time series T_i, and given a distance function d and radius $r \in \mathbb{R}$, find a set $\mathcal{P}$ of the k most frequent patterns of length n found on time series $(T_1, T_2, \ldots, T_P)$, such that:*

1. *The total communication cost regarding the number of messages and size of each message is minimized;*
2. *The result equals the one obtainable on a problem instance in which $\mathcal{L} = \{L\}$ and L holds the time series $(T_1, T_2, \ldots, T_P)$;*
3. *for all $L_i, L_j \in \mathcal{L}$, it can be shown that L_i learns nothing about the data owned by L_j, with $i \neq j$.*

To illustrate the problem statement, consider the following example.

Example 5.8 *Let a group of sites $\mathcal{L} = \{L_1, L_2, L_3\}$, and its respective local time series T_1, T_2, T_3. Let $T_1 = \langle 3, 4, 1, -2, 1, 5, 4, -7, 3\rangle$, $T_2 = \langle 9, -1, 0, -3, 0, 2, 3, 4, 1, 0, -5, 1\rangle$, and $T_3 = \langle 1, 4, -7, 3, -5, 3, 4, 1, 2\rangle$. The problem is to find subsequences that globally are reoccurring, even if at local time series it does not appear to be. In this example, the most frequent reoccurring pattern is formed by subsequence set $\{\langle 3, 4, 1\rangle\}$, which occurs in T_1 at $t = 1$, T_2 at $t = 7$, and T_3 at $t = 6$. The second most frequent pattern is formed by subsequence set $\{\langle 4, -7, 3\rangle\}$, which occurs in T_1 at $t = 7$ and in T_3 at $t = 2$ but does not occur in T_2. Each local site would learn which patterns are globally frequent, the occurrence set of each pattern at local time series (if any), but should know nothing about occurrence of global patterns in other sites.*

The PP-DPDTS problem requires that the privacy level from data owned by L_j does not decrease. Privacy measure for time series were discussed in Section 3.3.2. Subsequent sections will introduce approaches geared to solve the PP-DPDTS problem.

The following sections define three algorithms to discover frequent patterns in distributed time series. These are the first algorithms proposed in the distributed setting to the best of our knowledge. We first introduce the DPD-TS algorithm, swiftly followed by the presentation of the DPD-HE and DPD-FS algorithms, which is based on a heuristic to prune the number of subsequences in the search space. Each section provides inference attack analysis and experimental evaluation of the said algorithms.

5.2 DPD-TS Algorithm

In this section we present our DPD-TS algorithm [50, 45], which is the first step towards a solution to the PP-DPDTS problem (as stated in Section 5.1).

DPD-TS first computes the density of each local time series subsequences and outputs a list with the top k most dense subsequences. Our approach exploits the fact that a density estimate can be used to find overcrowded regions in a hyperspace. This study's central hypothesis is that if subsequences of time series are represented as points in a multidimensional space, it is possible to reduce the search for frequent subsequences to the search for the densest points in a multidimensional space. Consequently, the search for the most frequent patterns of size n reduces to the search for the densest points in a $\mathbb{R}^n$ space. To cope with the high dimensionality of time series, we first transform the original data to space $\mathbb{R}^w$ with reduced dimensionality, i.e., with $w \ll n$. Additionally, the resulting time series is discretized into a set of symbols from a given alphabet Σ to decrease the computation complexity of comparing subsequences. Within the discrete space, the algorithm identifies the densest strings, which are used back in the original $\mathbb{R}^n$ space to locate occurrences of frequent patterns.

Definition 5.11 (Alphabet and strings) *An alphabet Σ is a finite, totally ordered set of symbols. A string S of size w is an element of Σ^w.* □

Definition 5.12 (String distance) *Let Σ be an alphabet and two strings S and Q from Σ^w. A string distance for two strings S and Q is defined as a function $d(S, Q) : \Sigma^w \times \Sigma^w \rightarrow \mathbb{R}_+ \cup \{0\}$.* □

Example 5.9 (Manhattan String distance) *Manhattan distance for two*

strings S and Q is defined as $d(S,Q) = \sum_{i=1}^{n} |(ord(S[i]) - ord(Q[i]))|$, where $ord(x)$ returns 1 for the first symbol, 2 for the second symbol, and n for the n-th symbol in Σ.

DPD-TS computes the density over a distributed dataset. A local density estimate is computed at each peer, and, together, the peers sum up local densities to produce a global density estimate. Then, each local mining agent can independently discover frequent subsequences on their local dataset using the global density estimate.

5.2.1 Algorithm Overview

DPD-TS computes a set of frequent patterns occurring in the collection of local time series T_i owned by peers in the mining group $\mathcal{L}$. Peers are assumed to form a peer-to-peer network. DPD-TS needs the following parameters: T_i is the local dataset, n is the size used to generate subsequences, w is the number of symbols per string, i.e., the string size, Σ is the alphabet used to generate strings, and $\mathcal{L}$ is the set of peers forming the mining group. The parameter r defines the radius of the hypersphere to be used in the second step. As its output, DPD-TS returns a set $\mathcal{P}$ with the *globally* most frequent patterns. The pseudocode for DPD-TS is outlined in Algorithms 5.1 (initiator) and 5.2 (arbitrary party). Details are discussed in the following.

Negotiation. The first step in DPD-TS involves a negotiation concerning the value of the parameters. In this phase, any given peer may decide not to engage in the mining session if said negotiation is not in accordance with its local policy. The other peers may decide to continue with the mining session, even if some original first responders decide not to join the group. All further steps in the algorithm assume that the negotiation was successful.

As we remarked in Chapter 4, we do not explore the **negotiation protocol** in further details in this thesis. We assume it to be a multiplayer negotiation protocol that seeks to find a consensus agreement about the parameters' values. To ensure no outsiders could eavesdrop on the negotiation, we assume an asymmetric key system, like RSA or ElGamal [148, Ch. 8], is in place. The basic form of the negotiation consists of two steps: (a) the initiator broadcasts a call for a mining session with proposed parameter values; (b) interested peers answer to the call accepting the proposed values. A more

Algorithm 5.1 DPD-TS: Initiator

Input: k, T_1, n, w, Σ, r, $\mathcal{L}$;
Output: $\mathcal{P}$;

At the initiator party L_1 do:

1: negotiate($\mathcal{L}$,k,n,w, Σ,r)
2: $\hat{\varphi}_1 \leftarrow 0$
3: $\mathcal{S}_1 \leftarrow \emptyset$
4: **for** $(t \leftarrow 1;\ t \leq |T_1| - n;\ t \leftarrow t + n)$ **do** // *All non-overlapping subsequences*
5: $S \leftarrow T_1[t : t + n]$
6: $S' \leftarrow$ reduceDim(S, n, w); // *Using Eq.* (5.1)
7: $S'' \leftarrow$ discret(S', w, Σ); // *Local dim. reduction and discretization*
8: $\mathcal{S}_1 \leftarrow \mathcal{S}_1 \cup \{S''\}$
9: **end for**
10: **for all** $S'' \in \mathcal{S}_1$ **do**
11: $\hat{\varphi}_1(S'') \leftarrow$ estimateDensity(S'', w, r) // *Update local density estimation*
12: **end for**
13: **send** $\hat{\varphi}_1$ **to** L_2; // *Cooperative sum*
14: **receive** $\hat{\varphi}_P$ **from** L_P
15: $\hat{\varphi} \leftarrow \hat{\varphi}_P$ // *Global density estimate*
16: $\mathcal{M} \leftarrow$getCenters($\hat{\varphi}$,k, r) // *Discrete space*
17: **send** $\mathcal{M}$ **to** all agent $L_i \in \mathcal{L}$
18: $\mathcal{P}_1 \leftarrow$findLocalOccurrences(T_1,$\mathcal{M}$) // *Original space*

flexible approach would allow several rounds of counter-proposals until an agreement is or a deadline is reached. There are several sophisticated negotiation protocols for multi-party negotiation, e.g. [16, 137, 10, 146] to name a few. After the agreement, the peers in the mining session are coordinated as indicated in the algorithm using direct messages between peers.

Dimension Reduction. The for loop splits the original time series T_i in various non-overlapping subsequences S of size n. Function reduceDim() takes a subsequence $S \sqsubseteq T_i$ and computes a reduced subsequence S'. Each point of S' is the average of a small subsequence of S of size $\frac{n}{w}$. This operation (proposed elsewhere [114]) is known as piecewise aggregate approximation (PAA):

$$S'[j] = \frac{w}{n} \left(\sum_{k=\frac{n}{w}(j-1)+1}^{\frac{n}{w}j} S[k] \right) \tag{5.1}$$

where $S[k]$ is an element of subsequence S. This transformation reduces the dimensionality of a given subsequence S from n to w, where n is the size of S and w is the size of the resulting subsequence S'. The resulting time series T' is a concatenation of all reduced subsequences S' computed from T. Figures

Algorithm 5.2 DPD-TS: Arbitrary Party

Input: k, T_i, n, w, Σ, r, $\mathcal{L}$
Output: $\mathcal{P}$

At an arbitrary party L_i do:

1: negotiate($\mathcal{L}$,k,n,w, Σ,r);
2: $\hat{\varphi}_i \leftarrow 0$
3: $\mathcal{S}_i \leftarrow \emptyset$
4: **for** $(t \leftarrow 1;\ t \leq |T_i| - n;\ t \leftarrow t + n)$ **do** // *All non-overlapping subsequences*
5: $S \leftarrow T_i[t : t + n]$
6: $S' \leftarrow$ reduceDim(S, n, w); // *Using Eq.* (5.1)
7: $S'' \leftarrow$ discret(S', w, Σ); // *Local dim. reduction and discretization*
8: $\mathcal{S}_i \leftarrow \mathcal{S}_i \cup \{S''\}$
9: **end for**
10: **for all** $S'' \in \mathcal{S}_i$ **do**
11: $\hat{\varphi}_i(S'') \leftarrow$ estimateDensity(S'', w, r) // *Update local density estimation*
12: **end for**
13: **receive** $\hat{\varphi}_{i-1}$ **from** L_{i-1};
14: $\hat{\varphi}_i \leftarrow \hat{\varphi}_{i-1} + \hat{\varphi}_i$; // Updating with local density
15: **send** $\hat{\varphi}_i$ **to** $L_{(i \mod P)+1}$; // Send to next peer and the last one sends to initiator
16: **receive** $\mathcal{M}$ **from** L_1;
17: $\mathcal{P}_i \leftarrow$ findLocalOccurrences$(T_i$,$\mathcal{M})$

5.5 and 5.6 illustrate the dimension reduction of a single subsequence.

The following example illustrates the dimension reduction process.

Example 5.10 *Let* $T = \langle 1.0, 2.0, 3.0, 5.0, 3.0, 4.0, 8.0, 9.0, 5.0, 6.0, 7.0, 2.0\rangle$, *subsequence size* $n = 6$, *and word size* $w = 3$. *With these parameters, let extract the first subsequence* $S_1 = \langle 1.0, 2.0, 3.0, 5.0, 3.0, 4.0\rangle$ *from* T. *Since we want to reduce the size from* 6 *to* 3, *we compute three averages using two values at time. Therefore we have the reduced subsequences* $S'_1 = \langle 1.5, 4.0, 3.5\rangle$.

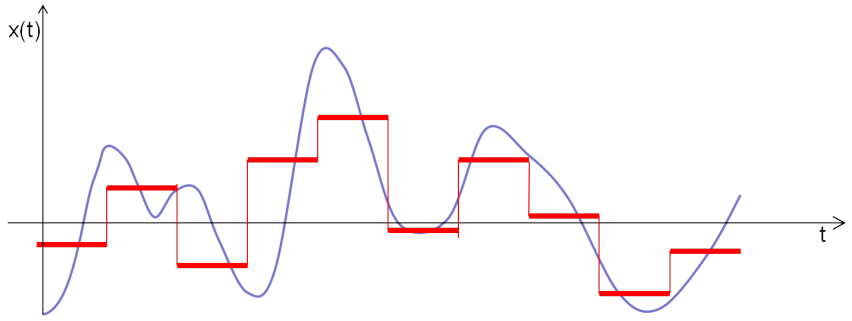


Figure 5.5: *A subsequence* S *of* T *is transformed in a subsequence of size* w. *Each point is the average value of* n/w *points of the original sequence* S.

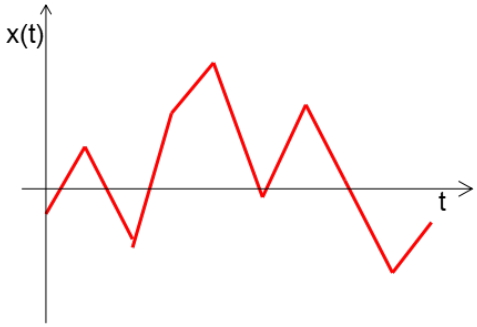


Figure 5.6: *Reduced subsequence is composed of w average values.*

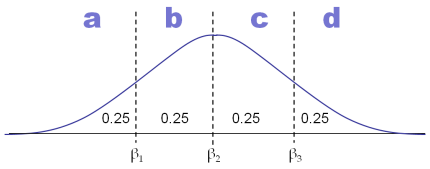


Figure 5.7: *Breakpoints for a 4-symbol alphabet under the normal distribution.*

Discretization. Function discret() produces a discretized version of a subsequence S' — a sequence of symbols in a given alphabet. The discretized version is referred to as S''. For each element of S', a symbol $\sigma_a \in \Sigma$ will be chosen at a corresponding position in the string S''. The substitution procedure subst() is accomplished by choosing breakpoints $\{\beta_a\}$ in the values dimension of a given subsequence S, such that $|\{\beta_a\}| = |\Sigma| + 1$, and such that each occurrence of a given value of S'' has the same probability [128], assuming they are typically distributed. For example, when considering a 4-symbol alphabet, 5 breakpoints are needed (including $\beta_0 = -\infty$ and $\beta_5 = +\infty$), and each region will have a 0.25 probability of appearing in the time series (cf. Fig. 5.7).

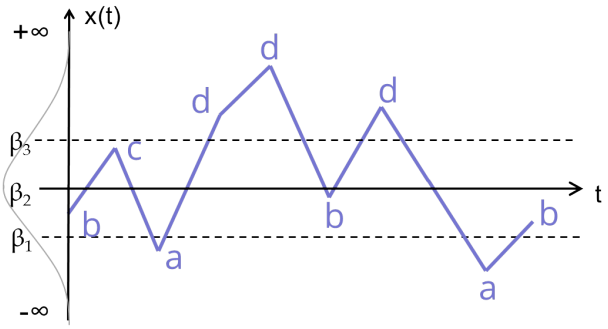


Figure 5.8: *A 4-symbol alphabet is used to discretize a given subsequence, generating the word "bcaddbdab".*

Then, the substitution rule is applied:

$$S''[j] = subst(S'[j]) = \begin{cases} \sigma_1 & \text{if } S'[j] \leq \beta_1 \\ \sigma_a & \text{if } \beta_{a-1} < S'[j] \leq \beta_a, \text{with } 1 < a < |\Sigma| \\ \sigma_{|\Sigma|} & \text{otherwise.} \end{cases} \tag{5.2}$$

Example 5.11 *Figure 5.8 illustrates the discretization process. Each value in the reduced time series is mapped to a symbol in the alphabet. Considering the alphabet* $\Sigma = \{a, b, c, d\}$ *and the reduced time series in Fig. 5.6, the discretized sequence of symbols is* S'' = "*bcaddbdab*". *Notice that breakpoints are chosen in the value (amplitude) dimension of time series and include both* $-\infty$ *and* $+\infty$.

Estimating Density of Strings. Function estimateDensity() computes the density estimates of each string S'' generated from time series T. An important requirement is that the density estimate function $\hat{\varphi}$ be a non-negative function over $\mathbb{R}$ and that the local maxima represent the densest regions in the feature space. In practice, the function does not even need to be a true estimation; an approximation will do.

A general approach to compute the data density function is kernel-based density estimation (as discussed in a previous chapter, Sec. 4.2.1). For a given kernel function K such that $\int_{\infty}^{+\infty} K(x)dx = 1$, an estimate of the

density, for a specific dataset D, is given by:

$$\hat{\varphi}[D,r](x) = \frac{1}{N\nu_{K,d}(h)} \sum_{i \in Neigh(x,r)} K\left(\frac{d(x,x_i)}{h}\right) \tag{5.3}$$

where N is the total number of points, d is a distance function. The parameter h is a bandwidth parameter and controls the smoothness of the density estimates[1]. The term $\nu_{K,d}(h)$ is a normalization factor for a given kernel K and distance d. $Neigh(x,r)$ is the set of indices of points in a neighborhood of point x in a given dataset D, considering only neighbors inside a ball of radius r computed with distance d.

The triangle kernel $K(u) = (1-|u|)I\{|u| \leq 1\}$ is employed, where I is the indicator function[2]. This kernel is chosen for its simplicity, but any other kernel can be used instead. The kernel bandwidth parameter is set out as equal to the neighborhood radius, i.e. $h = r$. The Manhattan distance, denoted d, is utilized. An arbitrary point x is a string S'' from T''_j, i.e. $S'' \sqsubseteq T''_j$. For a given discretized sequence T'', there are $|T''|/w$ non-overlapping strings with no gaps to consider. In the Kernel expression we will use the substitution $u = d(S'', S''_i)/h$.

The set of indexes $Neigh(x,r)$ refers to points in the neighborhood of subsequence S'' with radius r, which is a non-overlapping occurrence set for subsequences from a given time series (see Definition 5.8).

Putting everything together, we have:

$$\hat{\varphi}(S'') = \frac{w}{|T''|r} \sum_{i \in Neigh(S'',r)} \left(1 - \left|\frac{d(S'', S''_i)}{r}\right|\right) I\left\{\left|\frac{d(S'', S''_i)}{r}\right| \leq 1\right\} \tag{5.4}$$

$$= \frac{w}{|T''|r} \sum_{i \in Neigh(S'',r)} \left(1 - \left|\frac{d(S'', S''_i)}{r}\right|\right) \tag{5.5}$$

The indicator function was removed in Eq. (5.5) because the set $Neigh()$ contains only subsequences inside a ball of radius r centered at S''.

Computing Global Density. The mining group cooperatively computes the global density by summing up all local densities. This is made possible

[1] Small values of h yields spikier densities and larger values o h yields smoother density surfaces.

[2] The indicator function, also known as characteristic function, returns 1 if the expression in curly brackets holds and 0 otherwise.

by the fact that density estimates are additive, which means that the global density of a given subsequence is the sum of local estimates at each peer L_i:

$$\hat{\varphi}(S'') = \sum_{i=1}^{|\mathcal{L}|} \hat{\varphi}_i(S'') \tag{5.6}$$

Initially, peer L_1 sends its local density to L_2. After that, each peer L_j receives a partial density estimate $\hat{\varphi}_i$ from its neighbor L_{j-1}, with $j > 1$. L_j adds its own local density to partial global density, i.e. $\hat{\varphi}_j = \hat{\varphi}_j + \hat{\varphi}_{j-1}$. Then, site L_j sends partial global density $\hat{\varphi}_j$ to the next neighbor L_{j+1} in the mining group. This protocol continues until the partial sum is sent to L_1, which then broadcasts the global density estimate $\hat{\varphi}$ to all members of the mining group.

Finding Patterns by Locating Centers. To find the discrete patterns (dense strings), the initiator uses the function getCenters() in Alg. 5.1, line 16, which works as follows.

Consider the set D'' of all strings S'' generated from non-overlapping subsequences of T with non-zero density, i.e. $\varphi(S'') > 0$. Pick the most dense $S'' \in D''$ and call it S''_1. Now, remove S''_1 and its neighbors from D'', i.e., $D'' \leftarrow D'' \setminus (\{S''_1\} \cup \{S''_i\})$ with $i \in Neigh(S''_1, r)$. Again, pick the most dense S'' and call it S''_2. Additionally, $d(S''_1, S''_2) > 2r$ to avoid any intersections of neighborhoods. Remove S_2 and update D'' again. Repeat the process $k-2$ times or when only singletons neighborhoods are found. Ties are broken by the cardinality of neighborhood.

Let $\mathcal{M} = \{S''_i\}$, with $|\mathcal{M}| \leq k$, represent the set of local maxima in the global density estimate, i.e. the densest strings. For a given density estimate $\hat{\varphi}$, distance function d, we have:

$$\mathcal{M} = \{S''_i \mid \forall j \in Neigh(S''_i, r) : \hat{\varphi}(S''_i) > \hat{\varphi}(S''_j)\} \tag{5.7}$$

Finding Occurrences of Patterns in Original Local Time Series. Subsequences of the original time series are in $\mathbb{R}^n$, while the set of global patterns is in Σ^w. Therefore we need to identify occurrences of a pattern in the original local time series T_i (Problem 5.1). This process is carried out as follows. For each non-overlapping subsequence of $S_j \sqsubseteq T_i$, generated in the density estimation step, discretize it to S''_j and try to match it against

a string $S''_p \in \mathcal{M}$. Since $\mathcal{M}$ has no repeated elements, there should be only one match of S''_j and S''_p if any. All subsequences that matched to the same string $S''_p \in \mathcal{M}$ are identified as local occurrences of a given frequent pattern $P_p \in 2^{\mathbb{R}^n}$.

5.2.2 Complexity Analysis

Time. DPD-TS has a worst-case time complexity linear in the size of the local time series T_i, at each party.

Theorem 5.1 *Algorithm DPD-TS takes $O(|T_i|)$ steps at each party, where $|T_i|$ means the size of the original local time series at peer L_i.*

Proof. DPD-TS generates $\lfloor \frac{|T_i|}{n} \rfloor$ non-overlapping subsequences from T_i. The reduction step takes each subsequence and compute w averages for each one, by summing $\lfloor \frac{n}{w} \rfloor$ points for each mean, i.e. $w\lfloor \frac{n}{w} \rfloor = n$ steps. Discretization is a straightforward application of the transformation where each point in the reduced series is substituted by a symbol, which takes w steps. In particular, the density estimation step requires $O(N\lfloor \frac{|T_i|}{n} \rfloor)$ calls of the distance function, where N is the average number of neighbors and $\frac{|T_i|}{n}$ is the maximum number of subsequences. The discovery step scans $O(\lfloor \frac{|T_i|}{n} \rfloor)$ non-overlapping strings generated from the time series searching for the k-most dense regions in the density estimates. Finally, to find the instances on the original time series, each $\lfloor \frac{|T_i|}{n} \rfloor$ non-overlapping subsequences is tested, which takes $O(|T_i|)$ steps. Therefore, the overall time complexity is $O(\lfloor \frac{|T_i|}{n} \rfloor (n + w) + N|T_i| + |T_i|)$. Notice that normally $w < n \ll |T_i|$ and also $N \ll |T_i|$. Consequently, the time cost can be simplified to $O(|T_i|)$ in the worst case. □

Space. The density of each string is stored in a lookup table, which is a very sparse structure requiring one entry for each string corresponding to a non-overlapping subsequence of T_i, at each party.

Theorem 5.2 *Algorithm DPD-TS requires $O(|T_i|)$ space at each party.*

Proof. Each subsequence $S \sqsubseteq T_i$ has fixed length n. Consequently, there are $\lfloor \frac{|T_i|}{n} \rfloor$ possible strings to be generated from T_i. Therefore, in the worst-case scenario, all subsequences S are not similar to any other subsequence in T_i and all strings have to be stored in the lookup table. Therefore, DPD-TS

requires $O(\lfloor \frac{|T_i|}{n} \rfloor)$ space. Given that $n \ll |T_i|$, we can write simply $O(|T_i|)$. □

However, only a few variations of all possible patterns appear in practice unless the time series is entirely random.

Communication. The size of messages exchanged among peers is linear in the size of local time series T_i, at each party.

Theorem 5.3 *Algorithm DPD-TS generates messages with size $O(|T_i|)$ at each party.*

Proof. Message has size $O(|T_i|)$. Each peer sends one message to a neighbor and receives one from another neighbor. There are only two rounds of messages, one of which informs the mining results. In the worst case, when all subsequences are dissimilar, it is necessary to inform the density of every subsequence in T_i. There are $\frac{T_i}{n}$ subsequences in T_i. Therefore, each message has a size of $O(|T_i|)$. □

5.2.3 Inference and Collusion Attacks Analysis

This section analyzes the privacy properties of DPD-TS in insider and outsider attack scenarios.

Insider Attacks

Malicious Initiator Attack. Recall that the initiator peer knows all of the parameters' values, the set of global patterns, its local density estimates, and the global density estimates. It does not know any local data from other peers since, by construction, the DPD-TS scheme does not require the parties to transmit raw data. The only information transmitted by the peers during the mining session concerns the partial density estimates of subsequences. Therefore, a malicious initiator will not receive any piece of original sensitive data from other peers.

The initiator also has access to the global density estimate. However, the density estimate has no information on the order of each time series subsequence's occurrence, which is necessary to reconstruct the original time series.

The initiator computes the set of global patterns $\mathcal{P}$, from which it can attempt to infer the original values of the entire time series T. Since the discretization step is primarily responsible for the privacy level in DPD-TS, it is possible to deduce (from intuition alone) that the more symbols in the alphabet Σ, the less privacy is preserved, since the discretized version tends to ascertain the "shape" of the original time series data. The following result shows how the size of Σ influences the privacy level of a single point in T.

Lemma 5.1 *Let T be a time series and $S \sqsubseteq T$ a subsequence of length n. Let Σ be an alphabet of symbols used by DPD-TS scheme. Let $\{\beta_j \in \mathbb{R}\}_{j=1}^{|\Sigma|+1}$ be a set of breakpoints which divides the normal curve in $\mid \Sigma \mid +1$ equiprobable regions. Let $S'' \in \Sigma^w$ be the transform of S according to the discretization step of Algorithm DPD-TS. For a given point $x_t \in S$ if its transformed counterpart $x''_u \in S''$ is known, under a single insider attack, DPD-TS has privacy level given by:*

$$\mathbf{PR}^{TBK}_{DPD-TS[1]}(x_t) = (\beta_{j+1} - \beta_j)\frac{n}{w} \tag{5.8}$$

with $x''_u = subst(x_t)$, and $\beta_{j+1} < x_t < \beta_j$, $1 < j < |\Sigma| + 1$, and with $\mathbf{PR}$ as defined in Section 3.3.2 (cf. Def. 3.5).

Proof. (Amplitude) First, we show that the privacy of each point x_t in T, regarding its amplitude, is $\mid \beta_{j+1} - \beta_j \mid$. This is a consequence of the discretization step. Let $\sigma_j \in \Sigma$ be the symbol at point x''_u. After discretization, each original point x_t corresponds to a discretized point x''_u. Since the symbol σ_j comes from the substitution rule (cf. Sec. 5.2), x''_u corresponds to a value x'_u in the reduced subsequence S' and the amplitude value of x'_u lies in the interval (β_j, β_{j+1}). In the absence of further information, the only suitable option is to model the amplitude of x'_u as a random variable X uniformly distributed in the given interval, i.e. $X \sim U(\beta_j, \beta_{j+1})$. Now, using the bounded knowledge privacy Equation (3.9):

$$\begin{aligned}\mathbf{PR}^{BK}_{DPD-TS[1]}(X) = 2^{h(X)} = 2^{\int_{\beta_j}^{\beta_{j+1}} p(x) log_2 p(x) dx} = 2^{log_2(\beta_{j+1} - \beta_j)} \\ = \beta_{j+1} - \beta_j\end{aligned}$$

(Time) Each point x''_u in the discretized sequence is associated with a point x'_u in a reduced subsequence S'. Further, each point in the reduced

subsequence S' is the average value of $\frac{n}{w}$ points in S. Similarly to the argumentation for the amplitude, in the absence of further information, the only suitable option is to model the time stamp t of a given x_t used to compute the average x'_u as a random variable V uniformly distributed in the reduction interval, i.e., $V \sim U(t, t + n/w)$. Applying bounded knowledge privacy Equation (3.9) to V:

$$\begin{aligned}\mathbf{PR}^{BK}_{DPD-TS[1]}(V) &= 2^{h(V)} = 2^{\int_t^{t+n/w} p(v)log_2 p(v)dv} \\ &= 2^{log_2((t+(n/w))-t)} \\ &= \frac{n}{w}\end{aligned}$$

(Combining) Using Eq. (3.12) from Definition 3.5, we can write:

$$\begin{aligned}\mathbf{PR}^{TBK}_{DPD-TS[1]}(x_t) &= \mathbf{PR}^{BK}_{DPD-TS[1]}(X)\mathbf{PR}^{BK}_{DPD-TS[1]}(V) \\ &= (\beta_{j+1} - \beta_j)\frac{n}{w}\end{aligned}$$

□

Lemma 5.1 give us a direct relationship between the alphabet size and a bounding box (amplitude and time) protecting the true value of given point x_t. Breakpoints give the size of this box $\{\beta_j\}$ and the dimension reduction factor $\frac{n}{w}$. So, larger alphabets mean less privacy, and smaller alphabets mean more privacy. The same holds for w.

Using Lemma 5.1, peers in the mining group can define the minimum amount of privacy they require to join the mining session. In other words, if the discretization interval in amplitude and in the time dimension is smaller than a given local preference, the peer will not join the mining session. The local peer sets the desired size for the alphabet Σ and, consequently, the size of the intervals defined by the breakpoints $\{\beta_j\}$ (cf. Fig. 5.8). Thus, the minimum allowed size of the alphabet becomes a critical decision factor before joining a mining session. Of course, σ leaks some information about the amplitude of points (restricted to breakpoints), but no peer can reconstruct original points with arbitrary precision because the discretization process loses information, i.e., it is not invertible.

Single Arbitrary Party Attack. Each party in a DPD-TS mining session knows all of the parameters' values agreed in the negotiation step[3], the set of global patterns received from the initiator, and its local density estimates. Without collusion, an arbitrary party has no access to the density estimates of other peers and, as a consequence, may only try to infer the actual values x_t associated with the symbols in the global patterns. As in the previous attack, the privacy level is provided by discretization applied to the original time series T. Therefore, the privacy level of each point in the time series is the same as in the malicious initiator attack.

Collusion Attack (including initiator). Any collusion group that includes the initiator peer has information on the parameter values, the local density estimates of the attacker, and the global density estimates from the initiator. Thus, any collusion group that includes the initiator has enough information to isolate the partial density estimates of a group of victims. In the extreme case where $P-1$ peers collude against one single victim peer, the attackers can discover the victim density estimate. Nevertheless, the density estimate can only be used to find the most frequent patterns at any given peer or group of peers. As discussed in the first attack scenario, the density estimates reveal no information concerning each pattern's order. Because of that, the attackers cannot reconstruct the entire local times series of any victim. Again, the privacy of each point is preserved, as in the case of a single insider attack. This fact can be expressed as:

$$\mathbf{PR}^{TBK}_{DPD-TS[c>1]}(x_t) = \mathbf{PR}^{TBK}_{DPD-TS[1]}(x_t) \tag{5.9}$$

Collusion Attack (without initiator). Any collusion attack without the initiator has no access to the global density estimates. Thus, any collusion without the initiator brings no information to the collusion group beyond the public information they already have — the parameter values and the local density estimates, for instance. The privacy level of each point is preserved to the same level as in previous attacks.

[3]Cf. Negotiation protocol described at page 149.

Outsider Attacks

Under DPD-TS scheme, peers exchange parameter information during negotiation. We assume it is done through a secure channel to avoid leaks to outsiders. During the protocol, peers exchange density estimates from discretized subsequences in plain text form, i.e., not encrypted. However, as discussed in previous sections, the discretization procedure does not leak the original values. Therefore, an outsider has no information about parameter values. It can eavesdrop on local density estimates and global patterns, but this information is not enough to reconstruct the original time series precisely. From the subsequences in the density estimates, the outsider attacker may reconstruct the alphabet. On the other hand, it will not infer the values of parameters n, w, nor the number of breakpoints. Therefore, we consider that the malicious insider attack is a lower bound to an outsider attack.

$$\mathbf{PR}^{TBK}_{DPD-TS[0]}(x_t) \geq \mathbf{PR}^{TBK}_{DPD-TS[1]}(x_t) \tag{5.10}$$

Summary of privacy analysis

Theorem 5.4 *Let Σ be an alphabet of symbols, n the subsequence size, and w the pattern size. Let $\{\beta_j \in \mathbb{R}\}_{j=1}^{|\Sigma|+1}$ be a set of breakpoints which divides the normal curve in $\mid \Sigma \mid$ equiprobable regions. Let T be a time series and $T'' \in \Sigma^w$ be its transform according to the discretization step. DPD-TS has privacy level given by:*

$$\mathbf{PR}^{TBK}_{DPD-TS[c]}(T) = \min\{(\beta_{j+1} - \beta_j)\frac{n}{w} \ : \ \forall j = 1, \ldots, |\Sigma| + 1\} \tag{5.11}$$

even with collusion attacks, i.e. $c > 1$.

Proof. Using Eq. 3.13 it is possible to write:

$$\mathbf{PR}^{TBK}_{DPD-TS[1]}(T) = \min\{\mathbf{PR}^{TBK}_{DPD-TS[1]}(x_t) \mid t = 0, 1, 2, 3, \ldots, |T|\}$$

Lemma 5.1 stated that $\mathbf{PR}^{TBK}_{DPD-TS[1]}(x_t) = (\beta_{j+1} - \beta_j)n/w$. Thus, each point x_t in T has a different level of privacy, given by interval between its breakpoints. The smallest breakpoint interval gives the smallest privacy level among all points in T. Since the fraction n/w remains constant during a mining session, the smallest interval is independent of the time stamp t,

and can be found by looking at all breakpoint intervals used to discretize T. With more $c > 1$ colluders, this privacy level does not change. □

5.2.4 Experimental Evaluation

DPD-TS was implemented in JAVA 1.7, and all experiments were run in a machine with Intel Core i3, 2.1 GHz, 4 Gb RAM, with Windows 7 (64 bits). The results reported here are the averages from 50 runs with each parameter configuration.

Datasets. For the experiments reported here, we used the `power`, `sunspot`, and `tide` datasets. All datasets were downloaded from UCR Time Series Data Homepage [56], except the synthetic datasets.

- **Power** dataset presents the electricity consumption from the Netherlands Energy Research Foundation (ECN) for one year, recorded every 15 minutes. There are 35 040 data points corresponding to the year of 1997. Figure 5.9 shows an excerpt of the power data. This dataset has a pattern structure that can be observed visually.

- **Sunspot** dataset records the monthly average number of sunspots since January 1749 until 1993. There are 2 880 data points in this dataset[4].

- **Tide** data set is a collection of 12 years tide fluctuations at Crescent City, Northern California, recorded by National Ocean Service (NOA) from January 1980 to December 1991[5]. There are 8 746 data points in this series, two records per day.

- **Synthetic data** based on a random walk to evaluate the time performance of our algorithms. We generated 10 dataset with sizes 100 000 points, 200 000 points, up to 1 000 000 points.

[4] For up to date sunspot data visit `http://sidc.oma.be/`

[5] `https://tidesandcurrents.noaa.gov/sltrends/sltrends_station.shtml?id=9419750`

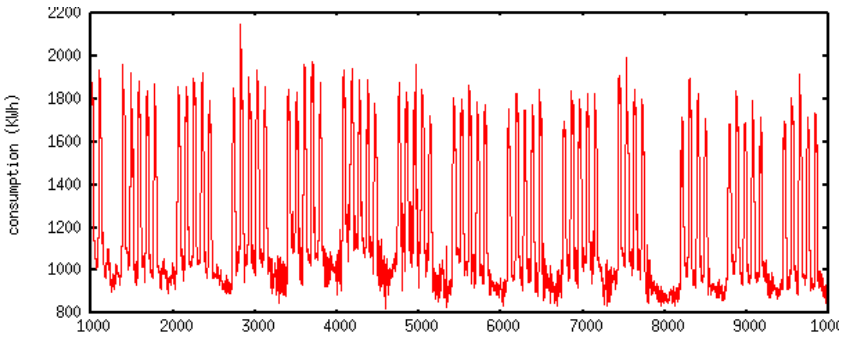


Figure 5.9: *Excerpt of power data.*

Efficacy Test. In this set of experiments, the efficacy of our proposed algorithms to find patterns is investigated. For all experiments, Σ was {a, b, c}, except for the experiments on time as a function of alphabet size. Symbol 'a' represents the lowest value and 'c' represents the highest value. The radius was set to $r = 1$. The kernel bandwidth was set to $h = 1$.

For `power` dataset, DPD-TS parameter were configured as follows. Subsequence size $n = 672$, which corresponds to 7 days (with one measurement every 15 minutes, i.e. 96 measurements per day). We choose pattern size $w = 7$, since it represents a week. Fig. 5.10(a) shows the most frequent patterns found in `power` dataset, "cccaacc", which corresponds to a normal week, showing high consumption on 5 workdays and low consumption at weekends (2 days). Figure 5.10(b) shows `power` dataset with several occurrences of the same frequent pattern "cccaacc", from position 672 to 3360, consecutively.

For `sunspot` dataset, DPD-TS was configured with $n = 100$ and $w = 10$. This dataset follows no human-made period, like the power dataset. Therefore, there is no obvious size w for a pattern in this in `sunspot` dataset. The value of w was defined after various different values were tried, guided primarily by the information loss on the dataset and visual inspection of patterns found. The most frequent pattern found with this configuration was "aaabccccb", which corresponds to a tilde like shape, shown in Fig. 5.10(c). Several occurrences of the most frequent pattern "aaabccccb" were found at 190, 400, 1000, 1400, 2320, and 2700, shown in Fig. 5.10(d).

For `tide` dataset, DPD-TS was set with $n = 480$ and $w = 8$. This configuration makes each symbol correspond to 60 measurements (i.e. one month) and therefore, a pattern corresponds to 8 months. Again, we picked this configuration by trial and error, trying to minimize information loss.

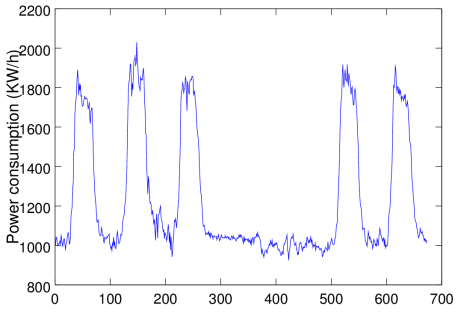
Dataset	Size (pts)	Avg time (s)	Stddev (s)
random100k	100 000	0.003647648	± 0.000618
random200k	200 000	0.007849766	± 0.001265
random300k	300 000	0.011790965	± 0.001857
random400k	400 000	0.019909966	± 0.005649
random500k	500 000	0.024471226	± 0.002877
random600k	600 000	0.029427808	± 0.007662
random700k	700 000	0.034729508	± 0.008040
random800k	800 000	0.039697080	± 0.008957
random900k	900 000	0.044820188	± 0.009452
random1000k	1 000 000	0.049096944	± 0.005584

Table 5.1: *Running time (in sec.) for different time series sizes.*

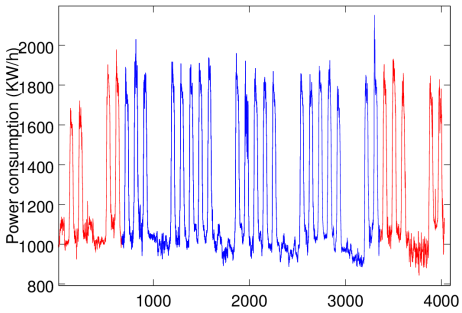
An occurrence of most frequent pattern "aaabbcbb" is shown in Fig. 5.10(e). Examples of occurrences of frequent patterns found in this dataset are presented in Fig. 5.10(f), with matches found at positions 3760, 4560 and 5280.

These results indicate that DPD-TS can find patterns expected to be found by visual inspection. However, DPD-TS is dependent on the starting point since it uses a jumping window strategy instead of a sliding window strategy. For example, in the `power` dataset, the most frequent pattern seems to start on Wednesday, with the weekend at the center ending with two workdays. On other datasets, which are not related to the human calendar and workdays, it is harder to tell if the patterns are shifted or not. Nonetheless, DPD-TS is very useful to find patterns when the pattern is aligned with the window. For example, it can easily find patterns in time series that follow a strict known period, like 24 hours or seven days. Moreover, this behavior can be improved by starting DPD-TS at different points in the dataset.

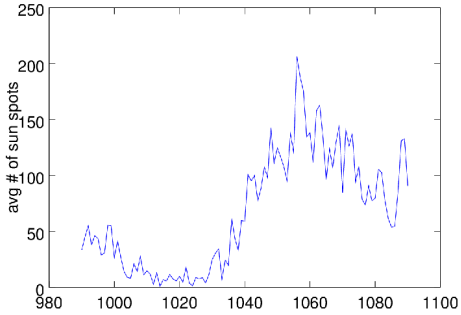
Time as a function of time series length. The running time of DPD-TS was evaluated using synthetic datasets with different sizes. Each synthetic dataset was generated with random numbers (a random walk) drawn from a uniform distribution from 0 to 1, i.e., $x \sim U(0, 1)$. Since this experiment is purely concerned with running time, no pattern was inserted in these synthetic datasets. We set $n = 10\,000$ and $w = 100$ for all synthetic datasets in this series of experiments. The result is shown in Figure 5.11 and Table 5.1. Time is indicated in seconds.



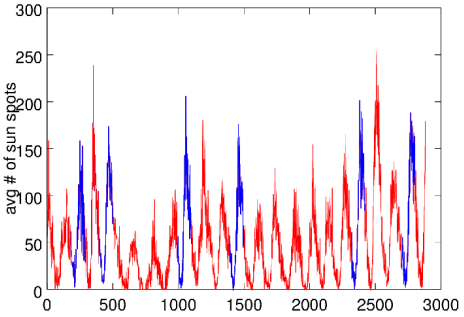
(a) An ocurrence of frequent pattern in `power` dataset



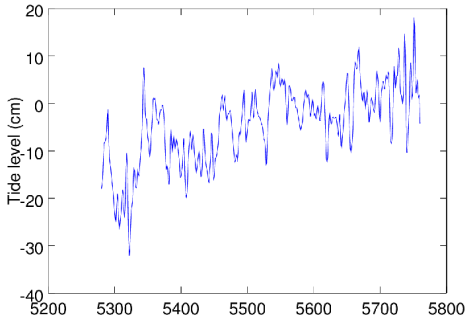
(b) Several pattern matches (blue) in `power` dataset



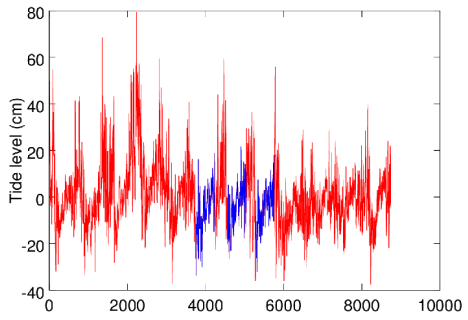
(c) An occurrence of frequent pattern in `sunspot` dataset



(d) Several pattern matches (blue) of frequent pattern



(e) An occurrence of most frequent pattern in `tide` dataset



(f) Three pattern matches (blue) in `tide` dataset

Figure 5.10: *Frequent patterns found by DPD-TS in different datasets.*

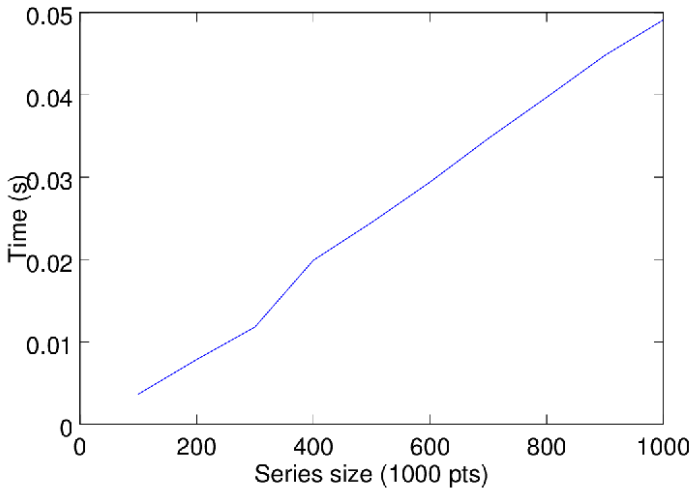
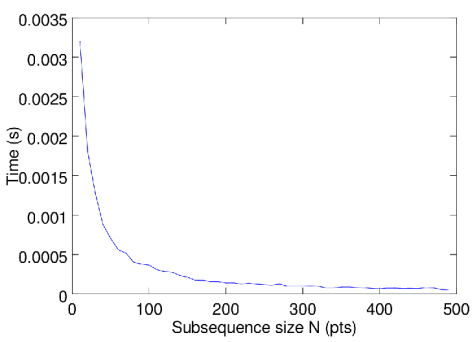


Figure 5.11: *Running time (in sec.) as a function of time series size.*

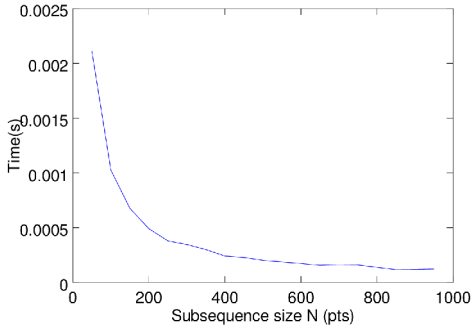
Time as a function of parameter n. In this set of experiments, the goal is to investigate the influence of the value of parameter n (the subsequence size) on the running time for a given fixed configuration. Figure 5.12 shows the results for four datasets (`sunspot`, `tide`, `powerdata`, and `random1M`). Notice that, with increasing sequence size n, there are fewer subsequences to process and, therefore, less running time. Larger values of n, with $n \geq w$, makes the ratio n/w increases and, consequently, a higher reduction rate. Therefore, higher values of n, with a fixed w, result in less overall running time.

Time as function of pattern size w. In this set of experiments, the goal is to evaluate the influence of pattern with size w on the running time of DPD-TS for a fixed subsequence size n. We set the subsequence size n with the same values used in the first set of experiments. Figure 5.13 shows the results for three datasets. In general, a higher value of w leads to a higher workload.

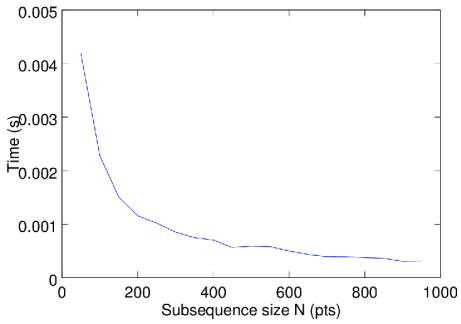
It can be noticed that the graphs follow a staircase-like shape, more clearly depicted in Figs. 5.13(b) and 5.13(c). This shape results from the reduction process that computes averages using $\lfloor n/w \rfloor$ points for each point in the reduced time series. The floor operation gives the same integer value for different n jumping to the next integer only when the remainder of n/w is 0.



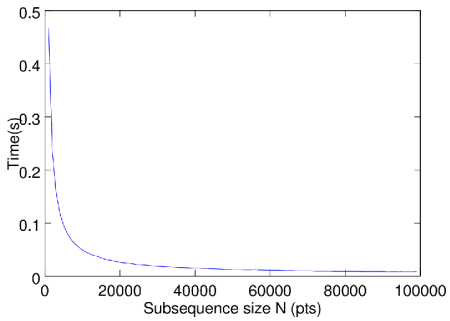
(a) `sunspot` dataset, size=2 880, $w = 10$



(b) `tide` dataset, size=8 700, w=10

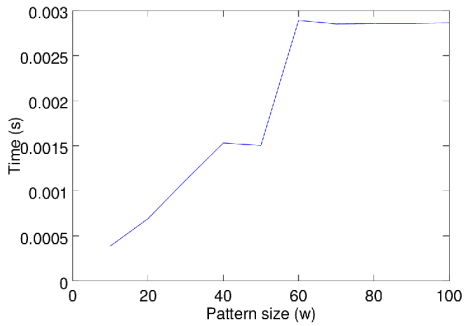


(c) `power` dataset, size = 35 000, w=7

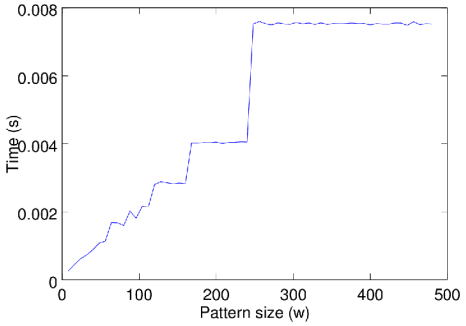


(d) `random1M` dataset, size = 1 000 000, w=100

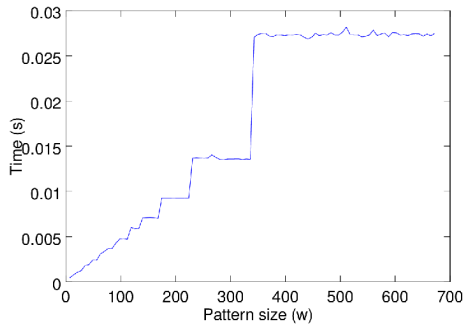
Figure 5.12: *Running time (in seconds) as a function of parameter subsequence size n. All experiments with alphabet = {abc}.*



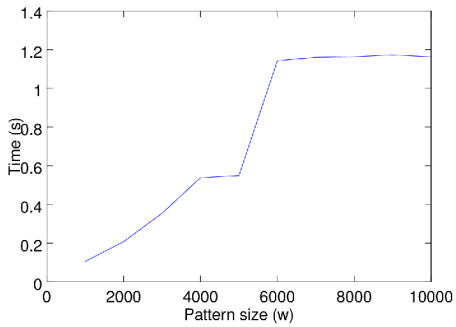
(a) `sunspot` dataset, size=2 880, $n = 100$



(b) `tide` dataset, size=8 700, $n = 480$



(c) `power` dataset, size = 35 000, $n = 672$



(d) `random1M` dataset, size = 1 000 000, $n =$ 10000

Figure 5.13: *Running time (in seconds) as a function of pattern size w. All experiments performed with alphabet* $= \{abc\}$.

In particular, with just 1 point, there is no average to compute, and therefore, two approximately constant regions are displayed at the end of the graphs[6], corresponding to values where the ratio $\lfloor n/w \rfloor$ is 2 and 1, respectively.

Privacy as a function of alphabet size. We use the interval size corresponding to each alphabet symbol in the discretization step to measure the privacy level of the amplitude.

In the initial case, with just one symbol, the size of the interval from the minimum x_{min} to the maximum value x_{max} observed in a time series was used. Assuming that *max* and *min* values are public, the attacker can compute the entropy of a random variable X over this interval and, conse-

[6]Recall that the results correspond to the average of at least 100 runs.

Alphabet	Size	Min. Interval
ab	2	0.8614
abc	3	0.6744
abcd	4	0.5066
abdce	5	0.4307
abcdef	6	0.3600
abcdefg	7	0.3186
abcdefgh	8	0.2794
abcdefghi	9	0.2533
abcdefghij	10	0.2283

Table 5.2: *Minimum interval between breakpoints with different alphabet sizes.*

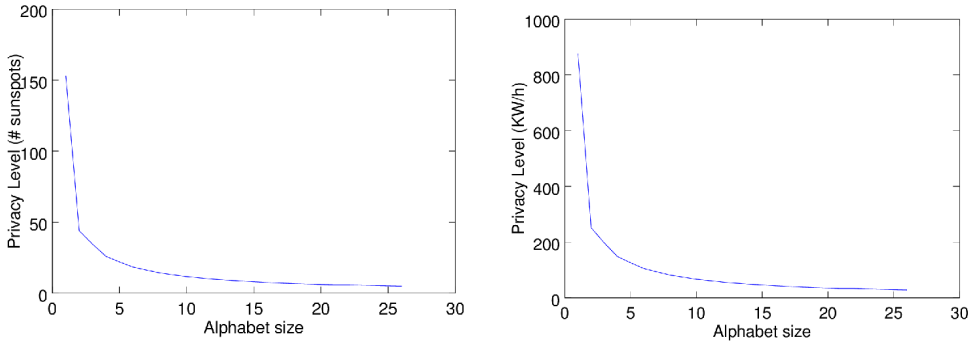
quently, the privacy level $2^{h(X)} = x_{max} - x_{min}$, using Eq. (3.13), discussed in Section 3.3.2. With more symbols in the alphabet, the interval provided by breakpoints is smaller, and the privacy decreases.

Table 5.2 shows the size of minimal intervals among breakpoints for different alphabet sizes. The values in this table are relative to breakpoints in a normalized curve. To apply these values to specific datasets, we need to normalize these sizes using the standard deviation and average value of a given dataset to get their privacy level. Normalization is necessary since breakpoints are computed over a standardized curve. Recall that our privacy level is a lower bound on the reconstruction precision. Figure 5.14 shows the privacy level applied to different datasets. For instance, with 3 symbols we can provide privacy level about 50 units in `sunspot` dataset, and a privacy level of 200 KW/h in `power` dataset. Therefore, using this table, one could choose the size of the alphabet which suits the privacy needs.

Information Loss. We compute the *information loss* as the mean average error (MAE) between the original time series and the reduced time series, as suggested elsewhere [189]:

$$IL_{MAE}(T, T') = \frac{\sqrt{\sum_{i=1}^{|T|} \left| x_i - x'_{\lfloor \frac{w}{n}(i-1) \rfloor + 1} \right|^2}}{|T|} \tag{5.12}$$

where T is the original time series, T' is the reduced time series (see Algorithm DPD-TS in Sec. 5.2), n is the size of the PAA sequence and w is the size of the pattern size. The involved index expression assures that each



(a) `sunspot` dataset (b) `power` dataset

Figure 5.14: *Privacy level (size of reconstruction interval of amplitude) as a function of alphabet size for different datasets.*

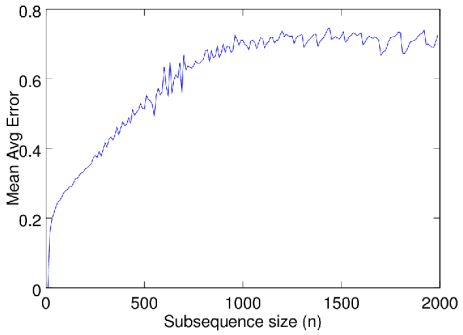
point in the original time series T is compared with the correct element in T', smaller than T. This metric is based on Euclidean distance since this distance is preserved for linear transform, e.g., PAA reduction [189].

Figure 5.15(c) shows the information loss for `sunspot`, `tide` and `power` datasets. Notice that the smallest error occurs when n is small and a positive proper divisor of the time series size. Similarly, w should be a positive proper divisor of n for less information loss. With increasing values of n, the information loss tends to be stable because the reduced time series is already so different from the original series that further reduction cannot produce much more loss.

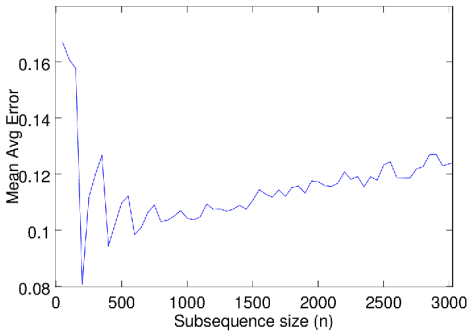
5.3 DPD-HE Algorithm

The algorithm discussed in the previous section discloses no exact values since there is no reference to raw data. Nevertheless, avoiding references to raw data may not be enough to protect privacy. In a distributed environment, an eavesdropper might be monitoring the conversation among the sites. To avoid this threat, one can apply a secure multi-party computation technique: homomorphic encryption [79]. In this section, the DPD-HE algorithm is put forth as a homomorphic encryption-based solution to the PP-DPDTS problem [49].

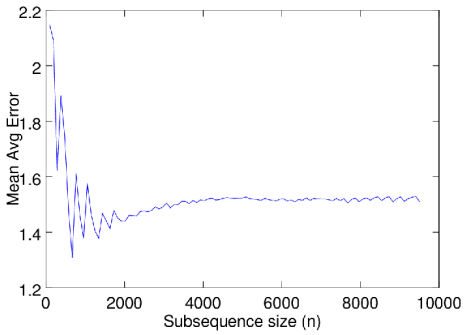
DPD-HE has the same basic assumptions as DPD-TS does, as discussed in previous sections.



(a) `sunspot` dataset ($w = 10$)



(b) `tide` dataset ($w = 8$)



(c) `power` dataset ($w = 7$)

Figure 5.15: *Information loss as a function of subsequence size n.*

Algorithm 5.3 DPD-HE: Initiator

Input: k, T_1, n, w, Σ, r, $\mathcal{L}$;
Output: $\mathcal{P}$;

At the initiator do:

1: negotiate($\mathcal{L}, k, n, w, r, \Sigma$);
2: $LDE_1 \leftarrow 0$
3: $\mathcal{S}_1 \leftarrow \emptyset$
4: **for** $(t = 1;\ t < |T_1| - n;\ t = t + n)$ **do**
5: $S \leftarrow T_1[t : t + n]$
6: $S' \leftarrow$ reduceDim(S, n, w); // *Using Eq.* (5.1)
7: $S'' \leftarrow$ discret(S', w, Σ);
8: $\mathcal{S}_1 \leftarrow \mathcal{S}_1 \cup \{S''\}$
9: **end for**
10: **for all** $S'' \in \mathcal{S}_1$ **do**
11: $LDE_1(S'') \leftarrow$estimateDensity(S'', w, r);
12: **end for**
13: $(PK_1, SK_1) \leftarrow$ generateKeyPairs();
14: **broadcast** PK_1 **to** $\mathcal{L}$;
15: $EGDE_1 \leftarrow$ encrypt$(PK_1,\ LDE_1)$
16: **send** $EGDE_1$ **to** L_2;
17: **for** $i = 2; i < |\mathcal{L}|; i++$ **do**
18: **receive** PK_i **from** L_i
19: **end for**
20: **receive** $EGDE_P$ **from** L_P
21: $GDE \leftarrow$decrypt$(SK_1, EGDE_P)$;
22: $\mathcal{P} \leftarrow$getCenters(GDE, k, r);
23: **for** $i = 2; i < |\mathcal{L}|; i++$ **do**
24: $EP \leftarrow$encrypt$(PK_i, \mathcal{P})$; // *Encrypts with each peer's public key*
25: **send** EP **to** L_i;
26: **end for**

5.3.1 Algorithm Overview

DPD-HE scheme is outlined in pseudocode shown in Algorithms 5.3 and 5.4. Its details are in the following.

The first step is the negotiation, where the mining group $\mathcal{L}$ needs to agree on the parameters used in the specific mining session. After that, each party computes the local density of strings generated from the local time series data, just like in the first step of Algorithms 5.1 and 5.2. The initiator, the peer who proposes and coordinates the mining session, creates a key pair and publicizes its public key.

The density estimate is computed using a kernel, as in the DPD-TS

Algorithm 5.4 DPD-HE: Arbitrary Party

Input: k, T_i, n, w, Σ, r, $\mathcal{L}$;
Output: $\mathcal{P}$;

At an arbitrary party L_i do:

1: negotiate($\mathcal{L}, k, n, w, r, \Sigma$);
2: $LDE_i \leftarrow 0$
3: $\mathcal{S}_i \leftarrow \emptyset$
4: **for** $(t = 1;\ t < |T_i| - n;\ t = t + n)$ **do**
5: $S \leftarrow T_i[t : t + n]$
6: $S' \leftarrow$ reduceDim(S, n, w); // *Using Eq.* (5.1)
7: $S'' \leftarrow$ discret(S', w, Σ);
8: $\mathcal{S}_i \leftarrow \mathcal{S}_i \cup \{S''\}$
9: **end for**
10: **for all** $S'' \in \mathcal{S}_i$ **do**
11: $LDE_i(S'') \leftarrow$estimateDensity(S'', w, r);
12: **end for**
13: **receive** PK_1 **from** L_1;
14: **receive** $EGDE_{i-1}$ **from** L_{i-1};
15: **send** PK_i **to** L_1;
16: $ELDE_i \leftarrow$encrypt(PK_1, LDE_i)
17: **send** $ELDE_i \cdot EGDE_{i-1}$ **to** $L_{(i \mod P)+1}$;
18: **receive** EP **from** L_1;
19: $\mathcal{P} \leftarrow$ decrypt(SK_i, EP);

algorithm. To securely estimate the global density, we apply two secure multi-party computation techniques are employed: (i) secure sum protocol and (ii) Paillier encryption scheme [156].

After the density is estimated, the initiator peer sends its public key and an encrypted density estimate to the second site. The second site produces an encrypted local density estimate, adding to the encrypted density received from the first site. The partially encrypted sum is passed from site to site until it reaches the first site again. Note that no site needs to (or can) decrypt the partial sum received from its neighbor.

Each party encrypts its local density estimate using the initiator's public key, denoted PK_1. Subsequently, each party waits until receiving the encrypted partial sum from its neighbor and adds its own local encrypted density estimate. Finally, it sends the updated encrypted partial sum to the next neighbor in the sequence. Each peer uses a Homomorphic Encryption (HE) scheme to perform addition without decryption.

A *Homomorphic Encryption* (HE) scheme allows parties to perform arith-

metical operation directly on encrypted information without decryption. In this work, we use Paillier encryption scheme [156], which is an additive homomorphic scheme. The additivity property of this scheme means that, given two messages m_1 and m_2, the following holds: $E(m_1) \cdot E(m_2) = E(m_1 + m_2)$. The Paillier scheme is also semantic secure, which means that under this scheme, no function of the plaintext can be computed given the ciphertext [82, 156].

The Paillier cryptography scheme [156] consists of the following steps:

Key Generation Let p and q be two large primes. Let $n = pq$ be a RSA modulus and g be an integer of order $\alpha n \bmod n^2$, for some integer α. The public key is (n, g) and the private key is the pair (λ, μ), where $\lambda(n) = \text{lcm}\,((p-1),(q-1))$ and $\mu = L(g^{\lambda} \mod n^2)^{-1} \mod n$.

Encryption The encryption of message $m \in \mathbb{Z}_N$ is $E(m) = g^m r^N \; mod \; N^2$, with r randomly selected from $\mathbb{Z}_N$

Decryption Given a cipher text c, the message is computed as follows:

$$m = L(c^{\lambda(N)} \; mod \; n^2) \cdot \mu \; mod \; n$$

where $L(u) = \frac{u-1}{n}$.

When all parties have added their local encrypted density estimate, it is sent back to the initiator. The initiator decrypts it and performs an algorithm searching for frequent patterns, i.e., local maxima in the global density estimate. This set of frequent patterns is then sent to the mining group, encrypted with each peer's public key.

5.3.2 Complexity Analysis

Time. The time complexity of DPD-HE is linear in the size of local time series T_i with a constant overhead due to the homomorphic encryption.

Theorem 5.5 *The time complexity of DPD-HE is $O(|T_i|E)$ at each party, where $|T_i|$ means the size of the time series at peer L_i and E denotes the overhead due to cryptography operations.*

Proof. The initiator calls decrypt() once to decrypt the partial density estimate sent by the last party in the group. The function decrypt() is applied

to each entry in the density estimate, which is a lookup table with $O(|T_i|)$ entries. The overhead due to cryptography at the initiator is $O(b^3)$, where b is the bit size of security parameter n, computed as $n = pq$, with two prime numbers p and q [156]. Similarly, each arbitrary peer calls encrypt() once to encrypt each entry in its local density estimate. The overhead is $O(b^2)$. Notice that the overhead is independent of the size of the time series and depends only on the choice of the security parameters. Consequently, the worst-case time complexity of DPD-HE is $O(|T_i|E)$, where E is the cryptographic overhead. Discarding the constant overhead E, for a given choice of security parameter, the time complexity of DPD-HE is then $O(|T_i|)$. □

Space. Ciphertext in Paillier requires the double of space required by the plaintext [156]. However, it is linear in the size of the local time series.

Theorem 5.6 *DPD-HE algorithm requires $O(|T_i|)$ space at each party, where $|T_i|$ is the size of the local time series.*

Proof. Recall that the reduction, discretization and density estimation steps are similar to DPD-TS and since DPD-TS requires $O(T_i)$ entries in the density estimate table, DPD-HE requires $O(2 \cdot |T_i|)$. This overhead is constant and independent of the density estimate's size. Therefore, DPD-HE requires $O(|T_i|)$. □

Communication. DPD-HE generates messages with a size linear to the size of local time series T_i. As a consequence of the space requirement, the messages exchanged by the peers informing the partial density estimate will require enough space to represent the density estimate.

Theorem 5.7 *DPD-HE generates messages with size $O(|T_i|)$ at each party, where $|T_i|$ is the size of the local time series.*

Proof. In DPD-HE each message will need $2 \cdot |T_i|$ space, which means that DPD-HE has communication cost given by $O(2|T_i|)$. Therefore, dropping the constant overhead, we have that the message size complexity is given by $O(|T_i|)$. □

5.3.3 Inference and Collusion Attacks Analysis

In this section, the privacy properties of DPD-HE are analyzed concerning insider and outsider attack scenarios.

Insider Attacks

Malicious Initiator Attack. Similar to DPD-TS analysis, it is assumed that the initiator in DPD-HE is aware of all of the values of the parameters, the set of global patterns, and its local density estimates. Additionally, the malicious initiator can decrypt the global density estimates. Therefore, this scenario is equivalent to the malicious initiator attack in DPD-TS. Thus, the privacy level of DPD-HE depends only on the size of the alphabet (cf. Eq. 5.8).

Lemma 5.2 *Let T be a time series and n the size of its subsequences. Let Σ be an alphabet of symbols used by DPD-HE scheme. Let $\{\beta_j \in \mathbb{R}\}_{j=1}^{|\Sigma|+1}$ be a set of breakpoints which divides the normal curve in $\mid \Sigma \mid$ equiprobable regions. Let T be a time series and $T'' \in \Sigma^w$ be its transform according to the discretization step of DPD-HE. For a given point x_t if its transformed counterpart x''_u is known, under a single insider attack, DPD-HE has privacy level given by:*

$$\mathbf{PR}^{TBK}_{DPD-HE[1]}(x_t) = (\beta_{j+1} - \beta_j)\frac{n}{w} \tag{5.13}$$

with $\mathbf{PR}^{TBK}$ as defined in Section 3.3.2 (cf. Def. 3.13).

Proof. Similar to the proof of Lemma 5.1 □

Single Arbitrary Party Attack. An arbitrary party in DPD-HE knows all of the parameter values, the set of global patterns, and its local density estimates. It does not have access to the partial global density estimates, only its encrypted version. Therefore, this scenario is equivalent to the single arbitrary party attack in DPD-TS. Again, the privacy level of DPD-HE depends only on the size of the alphabet (cf. Eq. 5.8).

Collusion Attacks. A collusion group with the initiator has access to all information in the previous scenarios, and it allows the attackers to isolate the density estimates from peers who are not in the collusion group. As in

DPD-TS, it is impossible to reconstruct its original time series even with a given peer's density estimates. Therefore, under a collusion attack, DPD-HE provides a similar level of privacy to that reached in DPD-TS in an equivalent scenario.

Outsider Attack

DPD-HE guarantees that an outsider will not learn anything from the data owned by the group members from the data exchanged during the protocol. The overall security is a consequence of the security of the Paillier encryption scheme, which was shown to be semantically secure elsewhere [156]. Semantic security ensures that no function from the plain text can be learned from the ciphertext. As a result, all the communication seems to be random data from the eavesdropper's point of view. Therefore, an eavesdropper cannot reconstruct the global density estimate or even a partial estimate sent among the parties. Consequently, an eavesdropper cannot reconstruct the original data nor estimate confidence intervals. Then, we have:

$$\mathbf{PR}^{TBK}_{DPD-HE[0]}(T) = \infty \tag{5.14}$$

Summary of privacy analysis

Theorem 5.8 *Let Σ be an alphabet of symbols, n the subsequence size, and w the pattern size. Let $\{\beta_j \in \mathbb{R}\}_{j=1}^{|\Sigma|+1}$ be a set of breakpoints which divides the normal curve in $\mid \Sigma \mid$ equiprobable regions. Let T be a time series and $T'' \in \Sigma^w$ be its transform according to the discretization step. DPD-HE has privacy level given by:*

$$\mathbf{PR}^{TBK}_{DPD-HE[c]}(T) = \min\{(\beta_{j+1} - \beta_j)\frac{n}{w} \ : \forall j = 1, \ldots, |\Sigma| + 1\} \tag{5.15}$$

even with collusion attacks, i.e. $c > 1$.

Proof. By similar argumentation as in Theorem 5.4, Eq. 3.13 can be used to write:

$$\mathbf{PR}^{TBK}_{DPD-HE[1]}(T) = \min\{\mathbf{PR}^{TBK}_{DPD-HE[1]}(x_t) \mid t = 0, 1, 2, 3, \ldots, |T|\}$$

Lemma 5.2 stated that $\mathbf{PR}^{TBK}_{DPD-HE[1]}(x_t) = (\beta_{j+1} - \beta_j)n/w$. With more $c > 1$ colluders, this privacy level does not change. □

As expected, encryption protocols do not affect the privacy against insider attacks. However, it provides an extra layer of security against attacks from outsiders.

5.3.4 Experimental Evaluation

The main difference between DPD-TS and DPD-HE is the homomorphic encryption protocol, which allows the peers to securely compute the global density estimates without the risk of an outsider eavesdropping in on the conversation. The following experiments investigate how much overhead is added to the running time due to the cryptographic operations in DPD-HE.

Running time as a function of the Security Parameter. DPD-HE works with the Paillier cryptosystem to run a secure sum protocol. One of the most critical parameters of the Paillier cryptosystem is the length (in bits) of the numbers used to generate the keys, also called modulo length in the original paper [156]. The larger the modulo length, the more secure the encryption, but more time is also needed to compute encryption and decryption. Paillier encryption is quadratic in the modulo length. Therefore, the encryption overhead in DPD-HE is quadratic in modulo length.

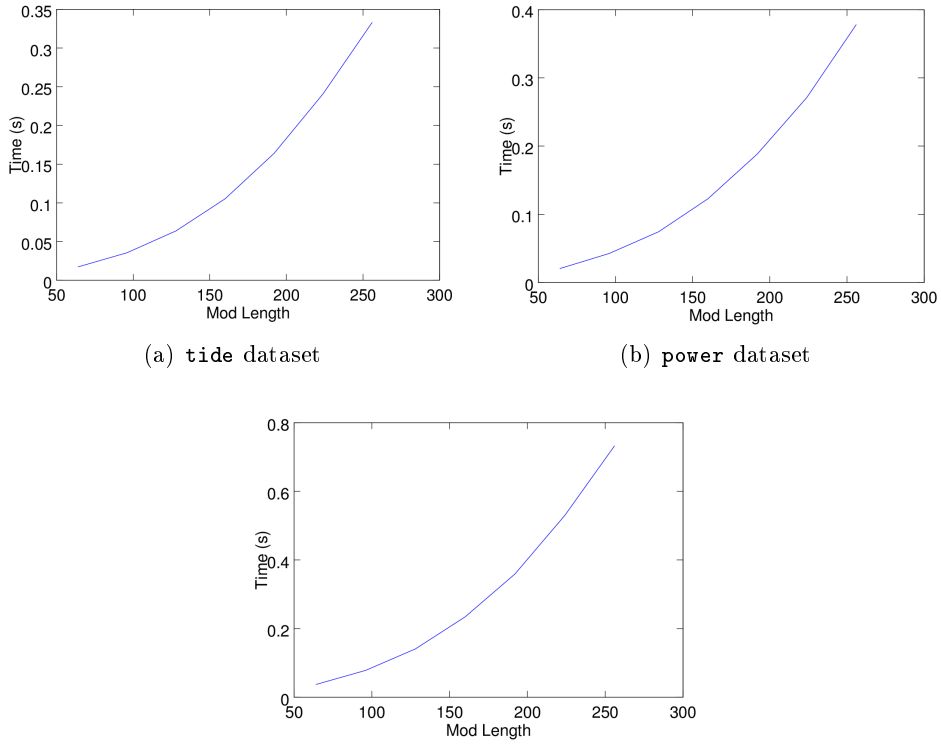
In this experiment, DPD-HE was tested with `power`, `sunspot` and `tide` datasets. For each dataset DPD-HE was run with *mod length* starting from 64 bits up to 256 bits. In all experiments $alphabet = \{abcd\}$, neighborhood radius $r = 2$ and kernel bandwidth $h = 1$ were employed.

For `tide dataset` $n = 480$, $w = 8$ were used. For `power` dataset $n = 672$, $w = 7$ were used. For `sunspot` $n = 100$, $w = 10$ were used. These are the basic settings also used to run DPD-TS experiments.

Figure 5.16 shows that the overhead due to cryptography is quadratic on the mod length (in bits) for all datasets[7].

Running time as a function of dataset size. DPD-HE was tested with three different fixed modulo lengths, for different dataset size. For this experiment, the following settings were applied: $alphabet = \{abc\}$, $n = 10.000$, $w = 100$, $r = 1$, $h = 1$. DPD-HE was run with synthetic datasets

[7] The fitted curve (for tide dataset) of the running time as a function of the mod length is $y = 0.0085x^2 - 0.0091x + 0.0237$, where x is the length in bits and y is the predicted running time.



(a) `tide` dataset

(b) `power` dataset

(c) `sunspot` dataset

Figure 5.16: *Running time as a function of security parameter mod length.*

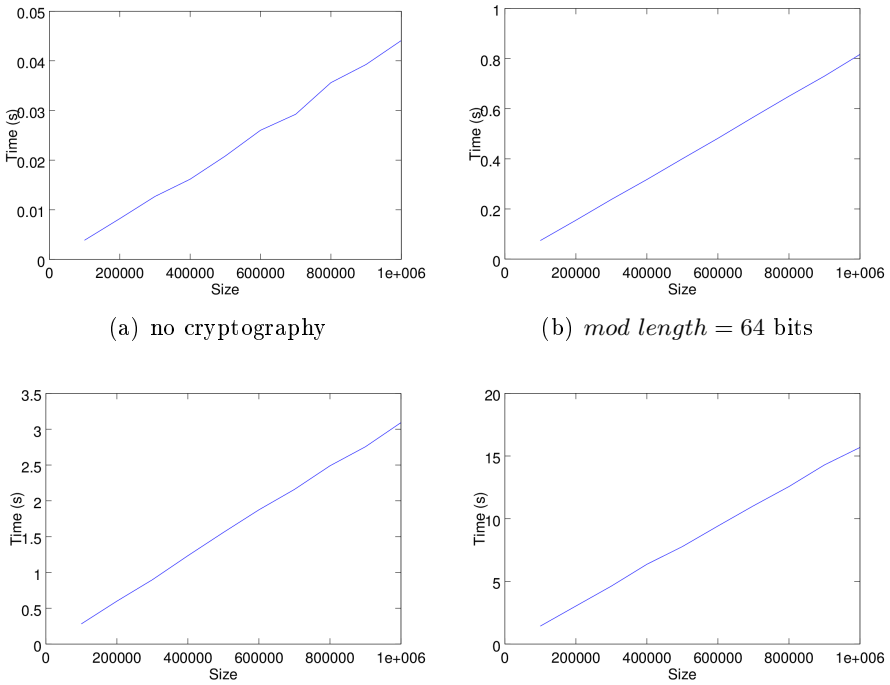
with sizes ranging from 200.000 data points to 1.000.000 data points. This are the same basic setting used to run DPD-TS experiments.

DPD-HE was run with three different values for *mod length*, namely 64, 128, and 256 bits. These are the most common choices for the length of cryptographic keys.

As shown in Figure 5.17, DPD-HE runs in linear time for different dataset sizes. Figure 5.17(a) shows the running time without cryptography, which is used as a comparison benchmark in the analysis of the other experiments. The figure shows a running time overhead due to cryptography, but it does not interfere with the overall linear algorithm's behavior.

5.4 DPD-FS Algorithm

In Section 5.2, we showed that DPD-TS algorithm is able to find frequent patterns in time series. However, DPD-TS is sensitive to the starting time



(a) no cryptography

(b) *mod length* = 64 bits

(c) *mod length* = 128 bits

(d) *mod length* = 258 bits

Figure 5.17: *Time as a function of dataset size for fixed mod length*

point because it generates non-overlapping windows from a given initial position. Consequently, a frequent pattern can be missed if its occurrences are not always aligned with the jumping window. Moreover, DPD-TS needs an explicit alphabet, which is only needed to discretize the original subsequences.

In this Section, we propose the DPD-FS algorithm. DPD-FS uses a sliding window approach to avoid missing out frequent patterns that are not aligned with a given starting position. To prune the search space, DPD-FS first computes the frequency of single discrete values and uses this frequency as the most promising location during the density estimation. Additionally, DPD-FS does not need an explicit alphabet; it works with an implicit alphabet of integers and projects subsequences of the original time series as points in a $\mathbb{Z}^w$ space.

In the following sections, we use the terms discrete subsequence and discretized sequence as defined below:

Algorithm 5.5 DPD-FS: Initiator

Input: number of patterns k, local series T_1, subsequence size n, pattern size w, discretization τ, radius r, peer group $\mathcal{L}$
Output: $\mathcal{P}_1$, a set of local occurrences of global frequent patterns

At the initiator party L_1 do:

1: negotiate($\mathcal{L}$,k,n,w,τ,r); // *as in algorithm DPDTS*
2: $\hat{\varphi}_1 \leftarrow 0$
3: $\mathcal{S}_1 \leftarrow \emptyset$
4: $hotSpots_1 \leftarrow$ identifyHotSpots(T_1, k); // *location of frequent symbols*
5: **for** ($t = 1$; $t \leq |T_1| - n$; $t \leftarrow t + 1$) **do**
6: $S \leftarrow T_1[t : t + n]$
7: $S' \leftarrow$ reduceDimension(S,n,w); // *as in algorithm DPDTS*
8: $S'' \leftarrow$ discretize(S', τ); // *Using eq. (5.16)*
9: $\mathcal{S}_1 \leftarrow \mathcal{S}_1 \cup \{S''\}$
10: $hotSpots_1 \leftarrow$ updateHotSpots(S'');
11: **end for**
12: **for all** $S'' \in \mathcal{S}_1$ **do**
13: $\hat{\varphi}_1(S'') \leftarrow$ estimateDensity($S'', hotSpots_1, w, r$); // *with hotspots*
14: **end for**
15: **send** $\hat{\varphi}_1$ **to** L_2;
16: **receive** $\hat{\varphi}_P$ **from** L_P;
17: $\hat{\varphi} \leftarrow \hat{\varphi}_P$
18: $\mathcal{S} \leftarrow$ getCenters($\hat{\varphi}$, k, r,) // *as in algorithm DPDTS*
19: **send** $\mathcal{S}$ **to** all agent $L_i \in \mathcal{L}$;
20: $\mathcal{P}_1 \leftarrow$ findLocalOccurrences(T_1, $\mathcal{S}$); // *as in algorithm DPDTS*

Definition 5.13 (Discrete subsequence) *Given the integer set $\mathbb{Z}$, a discrete subsequence is a finite list of integers. A subsequence S of size w is an element of $\mathbb{Z}^w$.* □

Definition 5.14 (Discretized sequence) *Given the integer set, a* discretized sequence *T is a list of integers from $\mathbb{Z}^*$.* □

The pseudocode for DPD-FS is given in Algorithms 5.5 and 5.6.

5.4.1 Algorithm Overview

Functions negotiate(), reduceDimension(), getCenters(), and findLocalOccurrences(), work just as they do in DPD-TS algorithm. The points where DPD-FS differs from DPD-TS are discussed in the following.

Discretization with implicit alphabet. Given a reduced subsequence S', the function discretize(S', τ) produces a discretized subsequence S'' using a discretization amount τ, such that every point in the reduced time series

Algorithm 5.6 DPD-FS: Arbitrary Party

Input: k, T_i, n, w, r,$\mathcal{L}$;
Output: $\mathcal{P}_i$, set of local occurrences of global patterns;
At an arbitrary party L_i do:
negotiate($\mathcal{L}$,k,n,w,r); // *as in algorithm DPDTS*
$\hat{\varphi}_i \leftarrow 0$
$\mathcal{S}_i \leftarrow \emptyset$
for $(t = 1;\ t \leq |T_i| - n;\ t \leftarrow t + 1)$ **do**
    $S \leftarrow T_i[t : t + n]$
    $S' \leftarrow$ reduceDimension(S,n,w); // *as in algorithm DPDTS*
    $S'' \leftarrow$ discretize(S', τ); // *Using eq. (5.16)*
    $\mathcal{S}_i \leftarrow \mathcal{S}_i \cup \{S''\}$
    $hotSpots_i \leftarrow$ updateHotSpots(S''); // *location of frequent symbols*
end for
for all $S'' \in \mathcal{S}_i$ **do**
    $\hat{\varphi}_i(S'') \leftarrow$ estimateDensity($S'', hotSpots_i, w, r$); // *with hotspots*
end for
receive $\hat{\varphi}_{i-1}$ **from** L_{i-1};
$\hat{\varphi}_i \leftarrow \hat{\varphi}_i + \hat{\varphi}_{i-1}$; // *Updating with local density*
send $\hat{\varphi}_i$ **to** $L_{(i \mod P)+1}$; // *Send to next; the last one sends to initiator*
receive $\mathcal{S}$ **from** L_1;
$\mathcal{P}_i \leftarrow$ findLocalOccurrences(T_i, $\mathcal{S}$); // *as in algorithm DPDTS*

is converted to an integer:

$$discr(x'_j, \tau) = \lfloor \frac{x'_j}{\tau} \rfloor \tag{5.16}$$

where each x'_j is a point in reduced subsequence S'. This transformation is one-way, and information loss is dependent only on τ.

As a result, a discretized version of the reduced S' is produced without using an explicit discretization alphabet. Of course, there is an implicit alphabet — the integers. Every symbol σ_j in the discretized S'' is an integer, given by $\sigma_j = disc(x'_j, \tau)$.

The total number of different symbols is denoted by α. In this implicit alphabet, the value of α is given by the range covered by the discretized subsequence S'' series, i.e. $\alpha = |\min\{S''\} - \max\{S''\}|$.

Example 5.12 *As an example, consider* $T'' = \langle 2, 2, 3, 4, 2, 2, 4, 4, 4, 4 \rangle$. *Suppose we are looking for patterns of size 2, i.e.* $w = 2$, *with a radius* $r = 2$. *Therefore, we will have the density estimate as shown in Figure 5.18. The two most dense subsequences are "22" and "44", which are indeed the most*

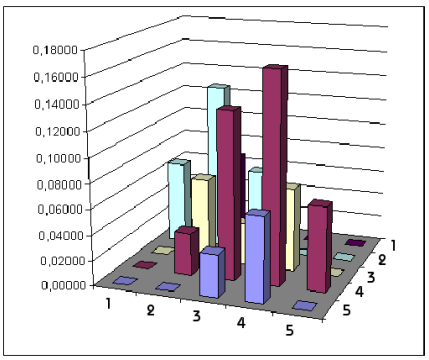


Figure 5.18: *Density of patterns of size 2 in the time series* $\langle 2, 2, 3, 4, 2, 2, 4, 4, 4, 4 \rangle$.

frequent discrete subsequences in the given discretized sequence.

Density Estimation of Subsequences Starting with Frequent Symbols. A common approach to pattern discovery is to use a sliding window of size w over the dataset. The sliding window approach produces $|T''|-w+1$ subsequences, most of which are not good candidates for frequent patterns. This study proposes focusing only on subsequences that are likely to be frequent (hot spots, cf. Alg. 5.5). The goal is to avoid using all subsequences produced by a sliding window approach and, at the same time, to minimize false dismissals.

In a first pass, DPD-FS identifies the positions where the k-most frequent symbols occur, which are called *hot spots*. In a second pass, the density is estimated considering only windows starting at hot spots, i.e., positions where a frequent symbol occurs. This heuristic is based on the observation that a frequent pattern necessarily includes a frequent symbol. Moreover, after marking a given subsequence, DPD-TS skips n points (the subsequence length) to avoid overlapping subsequences. Thus no trivial matches from subsequences close to each other in the time dimension contribute to the density estimation process.

Example 5.13 *Let us continue with the discretized sequence given in the example 5.12. Consider Fig. 5.18 again. In this example, the most frequent symbols are "2" and "4". The density is computed only to sequences "2*" "4*", where the star indicates any other symbol in the current alphabet. After the density is estimated, the most frequent patterns are "44", "22'. Observe that*

"34" has a high density too because pattern "44" contributed to the density of its neighborhood, which includes "34". On the other hand, since the pattern "34" also occurs in the string, it contributes to the density of "44". Since patterns must have a minimum distance from each other, we have that "44" eliminates "34" in this example because they are neighbors within radius $r = 1$. *Under these circumstances, the set of patterns becomes* $\mathcal{P} = \{\text{“22”}, \text{“44”}\}$.

5.4.2 Complexity Analysis

Time. The first step requires a pass through the dataset, which has size $|T|$, to produce the reduced subsequences. All other steps only require one pass through the entire dataset.

Theorem 5.9 *Algorithm DPD-FS takes* $O(|T_i|)$ *steps at each party, where* $|T_i|$ *means the size of the original local time series.*

Proof. The reduction step takes $\lfloor \frac{|T_i|}{n} \rfloor$ subsequences in T_i and compute the average for each subsequence. Discretization is a straightforward application of the transformation at each point in the reduced series. In particular, the density estimation step requires $O(N|T_i|)$ calls of the distance function, where N is the average number of neighbors and $|T_i| - n + 1$ is the maximum number of subsequences. The discovery step is constant, assuming that the density table is ordered by density value. Therefore, the time complexity is $O(|T_i| + N|T_i|)$, which can be simplified to $O(|T_i|)$ assuming $N \ll |T_i|$. □

Space. The density of each pattern is stored in a lookup table, which is a very sparse structure requiring one entry for each pattern that is found in the time series T_i.

Theorem 5.10 *Algorithm DPD-FS requires* $O(|T_i|)$ *space at each party.*

Proof. Each subsequence $S \sqsubseteq T_i$ has fixed length n. Consequently, there are $|T_i| - n + 1$ possible patterns in T_i. Therefore, in the worst-case scenario, all subsequences S are not similar to any other subsequence in T_i and all subsequences of T_i have to be stored in the lookup table. Therefore, DPD-FS requires $O(|T_i| - n + 1)$ space, which can be simplified to $O(|T_i|)$. □

In practice, however, only a few variations of all possible patterns appear, unless the time series is a *random walk*, i.e., a completely random time series.

Communication. Similiar to the two other algorithms, DPD-FS needs is linear on its message size.

Theorem 5.11 *DPD-FS generates messages with size $O(|T_i|)$ at each party, with $|T_i|$ meaning the size of the local time series.*

Proof. There are two rounds of messages; the first round of messages computes the global sum, while the second shares the frequent patterns found globally. Every message has the size of the density estimate, which is a lookup table. This table is always less than (or equal) to $|T_i|$. The equality holds only if every subsequence in the sliding windows is equally frequent. Therefore, the message size is $O(|T_i|)$. □

DPD-FS requires only two rounds of messages, one of which informs the mining results.

5.4.3 Inference and Collusion Attacks Analysis

Insider attacks

Malicious Initiator Attack. Malicious initiators know the parameters' values and local information such as local time series, local hotspots, global density estimates, and the global set of patterns. Additionally, the initiator receives the partial density estimates from the last peer in the group. The density estimates are computed for discretized subsequences, where each point is obtained as the result of a transformation. The original time series data has its privacy protected by the ambiguity provided by the data transformation (reduction and discretization) and can be located only in a region of arbitrary size, controlled by the parameters n and τ. The following lemma gives the details.

Lemma 5.3 *For a given subsequence size n and discretization amount τ and a given mining group with single attackers, i.e. c=1, for each point x_t in the original time series DPD-FS provides the following privacy level:*

$$\mathbf{PR}^{TBK}_{DPD-FS[1]}(x_t) = \tau n \tag{5.17}$$

Proof. Let T be a time series. In the reduction step, each subsequence

$S_i = \langle x_{i+1}, \cdots, x_{i+n-1} \rangle \sqsubseteq T$ is replaced by r_i, the subsequence average:

$$r_i = \frac{\sum_{j=i}^{n} x_j}{n}$$

(Amplitude) In the discretization step, each point r_i is replaced by an integer $a_i = \lfloor \frac{r_i}{\tau} \rfloor$. As a consequence of the floor operation in the discretization process, from the point of view of an attacker the amplitude of r_i can be modeled as a random variable X uniformly distributed in the interval $(a_i\tau, (a_i+1)\tau]$, i.e. $X \sim U(a_i\tau, (a_i+1)\tau)$. Now, using the bounded knowledge privacy Equation (3.9):

$$\begin{aligned}
\mathbf{PR}^{BK}_{DPD-FS[1]}(X) &= 2^{h(X)} = 2^{\int_{a_i\tau}^{(a_i+1)\tau} p(x) log_2 p(x) dx} \\
&= 2^{log_2((a_i+1)\tau - a_i\tau)} \\
&= \tau
\end{aligned}$$

(Time) Each point a_i in the discretized sequence is associated with a point r_i in the reduced time series T'. Furthermore, each point in the reduced time series T' is the average value of n points in $S_i \sqsubseteq T$. Similar to the argumentation for the amplitude, from an an attacker point of view, the time stamp i of a given $x_i \in S_i$ is a random variable V uniformly distributed in the reduction interval, i.e. $V \sim U(i, i+n)$. Applying bounded knowledge privacy Equation (3.9) to V:

$$\begin{aligned}
\mathbf{PR}^{BK}_{DPD-FS[1]}(V) &= 2^{h(V)} = 2^{\int_{i}^{i+n} p(v) log_2 p(v) dv} \\
&= 2^{log_2((i+n)-i)} \\
&= n
\end{aligned}$$

(Combining) Using Eq. (3.12) from Definition 3.5:

$$\begin{aligned}
\mathbf{PR}_{DPD-FS[1]}(x_t) &= \mathbf{PR}^{BK}_{DPD-FS[1]}(X) \mathbf{PR}^{BK}_{DPD-FS[1]}(V) \\
&= \tau n
\end{aligned}$$

□

Without further information, the interval size where the amplitude of r_i lies cannot be reduced. To infer the original data in a tighter interval, the attacker needs to know the average value r_i and its variance σ_i. However,

DPD-FS does not disclose the variance.

Single Arbitrary Party Attack. An arbitrary party has slightly less information than the initiator, and for example, it has no access to the global density estimates. Therefore, this attack does not improve on the previous attack scenario.

Collusion attacks. DPD-FS preserves the original values since it only exchanges a discretized version of the actual averages. As shown in the lemma 5.3, the size of the interval where a given average may be localized is given by the discretization amount and subsequence length, i.e., τn. A collusion attack involving the initiator and arbitrary parties could together have access to parameter values, local datasets, densities estimates and patterns of malicious parties, and global patterns resulting from the mining session. With this information, the malicious collusion could isolate the local patterns from a specific peer of a group of peers. Again, since the density and patterns do not use original data but discretized versions, the additional information does not help decrease the interval's size for a given value of the original time series.

Outsider attacks

An eavesdropper might listen in on the conversation between the members of the mining group. Assuming that the eavesdropper does not have a partner inside the group, messages do not convey enough information to allow a point reconstruction.

Summary of privacy analysis

Theorem 5.12 *For a given time series T, subsequence size n, discretization amount τ and number of colluders $c \geq 1$ in the mining group, DPD-FS provides privacy level of at least τ:*

$$\mathbf{PR}^{TBK}_{DPD-FS[c]}(T) = \tau n, \text{ for } c \geq 1 \tag{5.18}$$

Proof. With Eq. (3.13) it is possible to write:

$$\mathbf{PR}^{TBK}_{DPD-FS[1]}(T) = \min\{\mathbf{PR}^{TBK}_{DPD-FS[1]}(x_t) \mid t = 0, 1, 2, 3, \ldots, |T|\}$$

By lemma 5.3, $\mathbf{PR}_{DPD-FS[1]}(x_t) = \tau n$. By construction, all discrete points are obtained with the same value τ. Thus, the size of the interval is the same for all original points in T, i.e.

$$\min\{\mathbf{PR}^{TBK}_{DPD-FS[1]}(x_t)|\ t = 0, 1, 2, 3, \ldots, |T|\} = \tau n$$

Thus,

$$\mathbf{PR}^{TBK}_{DPD-FS[1]}(T) = \tau n$$

When $c \geq 1$, the size of the interval remains the same. Therefore,

$$\mathbf{PR}^{TBK}_{DPD-FS[c]}(T) = \tau n$$

□

5.4.4 Experimental Evaluation

Efficacy Tests. The first batch of experiments intended to test the DPD-FS' ability to find the most frequent patterns in the datasets. We use a value of τ close to the standard deviation of the dataset since it proved to produce the best results. The subsequence size n and window size w were chosen according to the values used in DPD-TS experiments so that it produces similar results[8]. The radius $r = 1$ for all experiments.

For the `power` dataset, DPD-FS was configured with $n = 96$, $w = 7$ and discretization amount $\tau = 100$. Figure 5.19(a) shows a *normal week*, the most frequent subsequence showing high consumption on workdays and low consumption at weekends. In Figure 5.19(b) an excerpt of `power` dataset with several matches of the same pattern "12 12 12 12 12 9 9" is clearly exhibited, showing the matches at positions 6528 and 9216.

For `tide` DPD-FS was configured with $n = 60$, $w = 7$ and discretization amount $\tau = 13$. Figure 5.19(c) shows an occurrence of a frequent pattern with increasing tide level for 7 months. Figure 5.19(d) shows `tide` dataset with several matches of the same pattern "-1 -1 -1 -1 0 0 0", here showing the matches at positions 4560, 6780 and 7440.

[8] In DPD-TS the value of n is divided by w to generate the reduced time series. On the other hand, DPD-FS uses only n and assumes $w = 1$ in the reduction process. Therefore, we set n in DPD-FS with smaller values to get the same amount of work as in DPD-TS.

Dataset	Size (pts)	Avg time (s)	Stddev (s)
random100k	100 000	0.0025352	± 0.000237
random200k	200 000	0.2046958	± 0.008428
random300k	300 000	0.3677157	± 0.010010
random400k	400 000	0.5026984	± 0.014731
random500k	500 000	0.7461030	± 0.012815
random600k	600 000	0.9458456	± 0.008886
random700k	700 000	1.0976415	± 0.007945
random800k	800 000	1.2588806	± 0.006263
random900k	900 000	1.4727574	± 0.009267
random1000k	1 000 000	1.6535516	± 0.015850

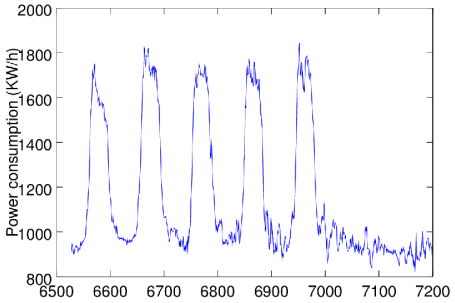
Table 5.3: *Running time data for different time series sizes.*

For `sunspot` dataset, DPD-FS was configured with $n = 10$, $w = 10$ and $\tau = 50$. Figure 5.19(e) shows an occurrence of a frequent pattern, displaying increasing average number of sun spots. Figure 5.19(f) shows `sunspot` dataset with several matches of the same pattern "0 0 0 0 0 1 1 1 1 1", here showing the matches at positions 70, 890, 1260, 1540, 1660, 1820 and 2070

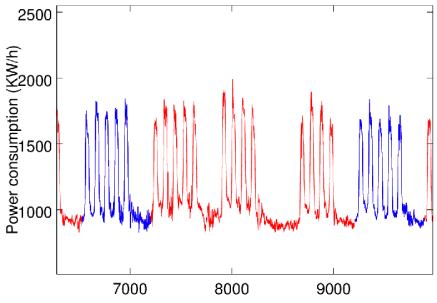
Patterns found by DPD-FS seem to be more natural than those found by DPDTS. Moreover, DPD-FS is not affected by the starting point since it checks every potential place (sliding window with hotspots). In `power` dataset, it proved able to find the pattern corresponding to a typical week in the correct alignment (5 workdays followed by two days on the weekend). In general, DPD-FS was able to find more occurrences of frequent patterns than DPD-TS.

Time as a function of time series length. The running time of DPD-FS was evaluated using the same set of synthetic datasets used to evaluate DPD-TS. We set $n = 1\,000$, $w = 10$ and $\tau = 1$. The result is shown in Figure 5.20 and Table 5.3. Time is indicated in seconds. Results support our claim that DPD-FS is linear in the size of time series. Compared to DPD-TS it is much slower due to the sliding window approach used in DPD-FS. However, DPD-FS misses less frequent patterns, trading off performance for efficacy.

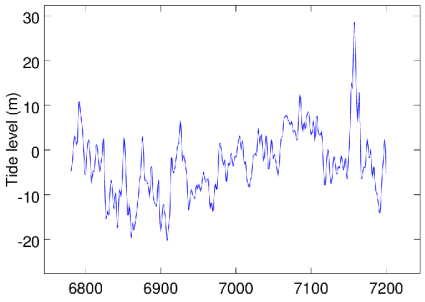
Time as a function of parameter n. In this experiment, the influence of the value of parameter n on the running time of DPD-FS is evaluated.



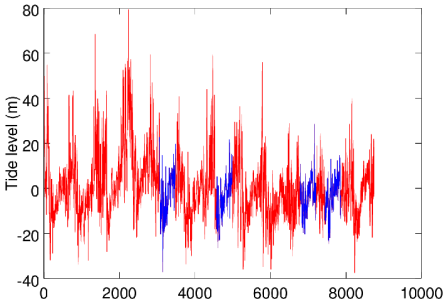
(a) A occurrence of a frequent pattern in `power` dataset



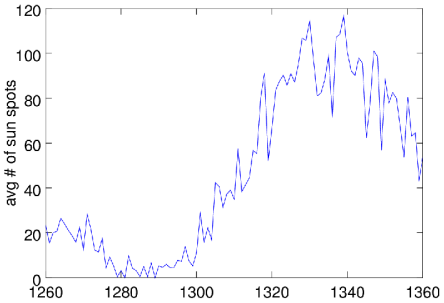
(b) Matches (blue) of frequent pattern in `power` dataset



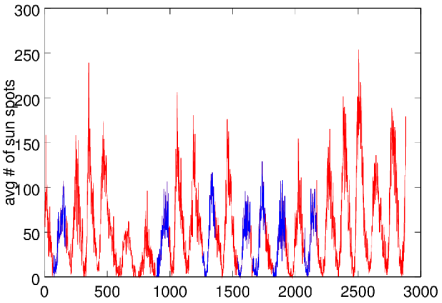
(c) A occurrence of frequent pattern in `tide` dataset



(d) Matches (blue) of frequent pattern in `tide` dataset



(e) A occurrence of frequent pattern in `sunspot` dataset



(f) Matches (blue) of frequent pattern in `sunspot` dataset

Figure 5.19: *Frequent patterns found by DPD-FS in different datasets.*

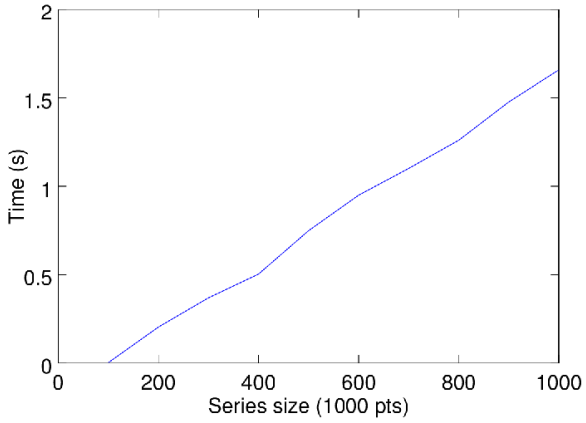


Figure 5.20: *Running time (in seconds) as a function of time series size ($n = 1000$, $w = 10$).*

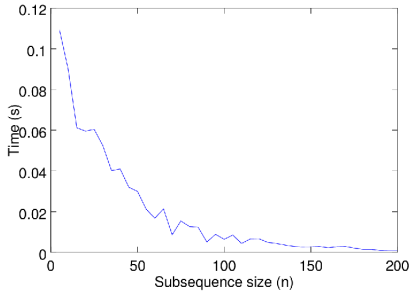
Figure 5.21 shows the results for four datasets (`sunspot`, `tide`, `powerdata`, and `random1M`). Recall that higher values of n produce smaller reduced series and fewer points to process. Hence, less running time.

Time as function of pattern size w. In this set of experiments, the influence of pattern size w on the running time of DPD-FS is evaluated. Figure 5.22 shows the results for four datasets. The sliding window approach followed by DPD-FS can be observed in these results, as it causes a quadratic overhead with the size of the pattern. Therefore, if we need longer patterns, we must choose larger n.

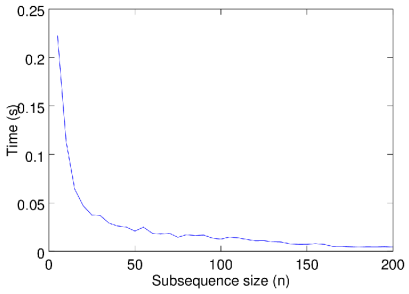
Privacy. For DPD-FS, tests on privacy were not performed, since privacy level is computed as τn, given discretization parameter τ and subsequence size n.

Discussion. The experiments show that utilizing density estimate is a meaningful approach to pattern discovery in time series, provided that trivial matches are avoided. How to avoid trivial matches optimally is an open question.

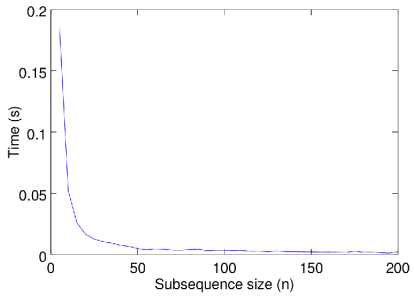
In particular, the choice of the window size w has a significant influence on the results. The best window width is chosen, guided by the information loss metric. However, a systematic way to find the window width is still needed. An even better option would involve working with a variable window.



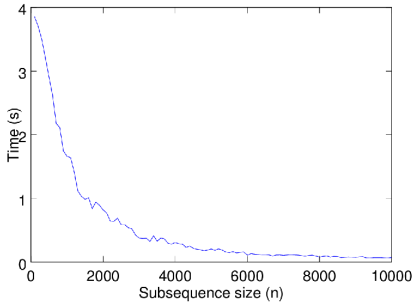
(a) `sunspot` dataset, size=2 880, $w = 10$



(b) `tide` dataset, size=8 700, $w = 7$

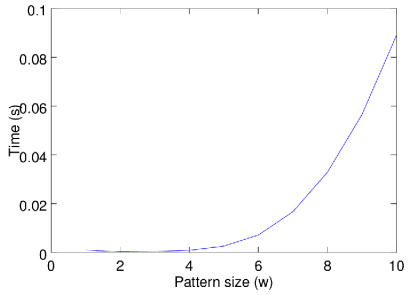


(c) `power` dataset, size = 35 000, $w = 7$

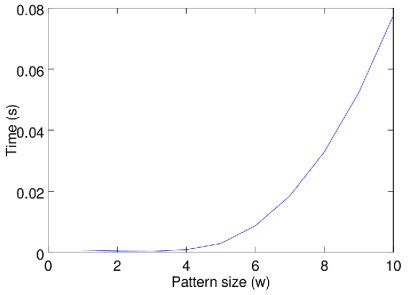


(d) `random1M` dataset, size = 1 000 000, $w = 10$

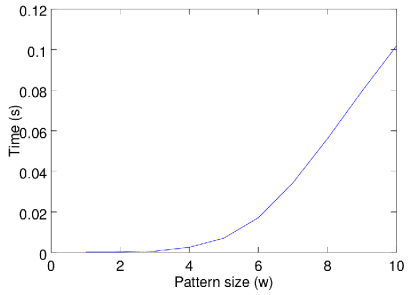
Figure 5.21: *Running time (in seconds) as a function of parameter subsequence size n.*



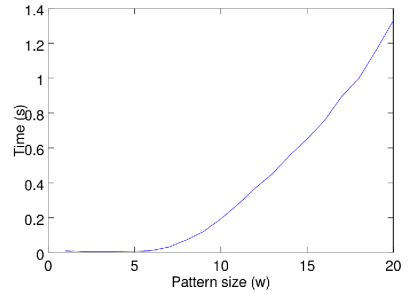
(a) `sunspot` dataset, size=2 880, $n = 10$, $\tau = 50$



(b) `tide` dataset, size=8 700, $n = 60$, $\tau = 13$



(c) `power` dataset, size = 35 000, $n = 96$, $\tau = 100$



(d) `random1M` dataset, size = 1 000 000, $n = 5000$, $\tau = 1$

Figure 5.22: *Running time (in seconds) as a function of pattern size w (DPD-FS).*

5.5 Application to Genomic Data

We applied the main idea of pattern discovery with density estimates, discussed in this chapter, to cluster genomic data [54]. Biologists want to cluster gene expression to find related genes, hinting at new pathways and regulatory functions. In this setting, the data points are time series from gene expression experiments and are typically no longer than 15 time points. Further, gene expression experiments are costly, constraining a given research team to produce only a small number of them. Consequently, typical clustering algorithms, such as k-means or SOM, do not perform well on genomic data due to data sparsity. Several short time series clustering algorithms were proposed, e.g., STEM, IBCC, SiMM, FBPA, and FCV-TS. Current approaches, however, have time complexity quadratic on the number of genes, the number of data points, or both. Moreover, no approach was designed to work on a distributed data set, and none addressed privacy issues. In this context, we developed DTSCluster [54], a scalable clustering algorithm to short time series from genomic experiments. Additionally, as a result of being based on density estimates, it can work in a distributed data scenario. Additionally, DTSCluster's data discretization mechanism provides the possibility of privacy preservation to genomic data, a topic attracting more attention in the last few years.

5.6 Related Work and Discussion

We proposed the first distributed algorithms for pattern discovery in time series with privacy preservation to the best of our knowledge. Therefore, we cannot compare our algorithm directly with competitors. In the following, we discuss works that are partially related to our algorithms. First, we discuss pattern discovery approaches. Then we review a few examples of privacy-preserving approaches for distributed time series forecasting, aggregate statistics, data publishing, and similarity queries.

Pattern Discovery in Time Series. Pattern discovery aims to find frequent patterns in sequences of symbols. It has been extensively studied in bioinformatics, e.g., RNA-seq time series data analysis [191].

A prominent approach to time series related problem is the Matrix Pro-

file Index (MPI) [235]. MPI is a data structure that holds all distances between subsequences of a given size in a time series. It has been shown that MPI effectively finds the closest pair of subsequences, the most distance pair, and other features. MPI serves as a primitive operation to solve several pattern-related problems such as motifs, discords, top-k motifs. Several optimizations and extensions on the original idea have been proposed [5, 105, 246]. However, no distributed MPI or privacy-preserving version has been investigated to the present date.

Gao and Lin developed HIME, a variable-length motif detector [77]. HIME first produces a discretized (symbolic) version of the time series based on the SAX approach. The authors apply several optimization steps to accelerate the enumeration of all motifs of different lengths. The algorithm generates a hierarchical motif enumeration structure describing all repeated subsequences found in the time series. The results show that HIME is linear in the size of the time series. However, HIME is developed for the centralized case and does not address privacy.

Gao et al. [78] addressed the problem of anomaly detection (surprise, or discord) in time series using an ensemble of models generated with different parameter values. The worst models are discarded, and the final model is generated by combining base models with a median. The ensemble approach was linear, with similar results to the state-of-the-art approaches. However, the approach is centralized, and no privacy preservation technique is applied.

We refer the interested reader to a survey [205] on pattern discovery in time series.

Privacy-preserving Distributed Time Series Approaches. Here we discuss distributed and privacy-preserving approaches but not related to pattern discovery, including forecasting, computing aggregate statistics, and time series clustering.

Rajagopalan et al. [171] investigated how to forecast load demand and optimize pricing structure from smart meter data. They proposed an information-theoretical approach to model privacy as information leakage, defined as the mutual information between original and distorted data. The privacy mechanism that distorts the time-series data is designed and applied offline once over the whole sequence before the release. Addressing a similar problem, Gonçalves et al. [87] adopt a vector autoregressive-based approach to

improve the forecast accuracy. Each party builds a local model, and only coefficients are exchanged. Privacy is preserved via multiplicative perturbation of model coefficients, and the algorithm runs until a convergence criterion is met. Imtiaz and colleagues [106] proposed a forecasting algorithm for distributed health data collected from wearable devices. Time series data from users are first clustered using k-means algorithm to identify groups of similar users. For each group, an LSTM (long-short term memory) network is locally trained, and a central server coordinates the upgrade of a global LSTM model. Local data and LSTM parameters are perturbed with noise before being sent to the server. The privacy model adopted is ϵ-differential privacy.

Shi et al. [186] consider how an untrusted data aggregator can compute statistics over multiple participants' time series data, preserving each individual's privacy. They propose a Diffie-Hellman-based encryption scheme that does not require establishing (and storing) pairwise keys between nodes. Each participant perturbs its data with noise from a given probability distribution and encrypts it using an individual key. The aggregate function is computed via homomorphic encryption by the aggregator decrypting only the output. This scheme satisfies a relaxed version of differential privacy definition and can handle collusion when a known fraction of the participants are compromised. Benhamouda et al. [19] define a generic framework for time series statistics aggregation with tighter differential privacy bounds than [186]. The approach is based on Cramer-Shoup's paradigm of smooth projective hashing.

Allard et al. [7] proposes Chiaroscuro to address the problem of securely clustering distributed personal data. The setting assumes a peer-to-peer network of mobile equipment generating personal data, e.g., health information or smart meters in households. The algorithm relies on distributed differential privacy and additively homomorphic encryption scheme to ensure data privacy. The main algorithm is based on k-means, computing centroids for each cluster iteratively until a fixed number of iterations is reached.

Liu et al. [139] addressed the traffic flow prediction problem. Each peer collects time series data but cannot exchange datasets. The proposed approach trains local models at each party, and a parameter aggregation algorithm is run collaboratively to compute a global model. Homomorphic encryption ensures that local parameters are exchanged securely among the

parties.

Privacy-preserving Time Series Publishing. Here we discuss privacy-preserving approaches for data publishing. It is an important setting where sensitive time series generated at different locations must be transferred to a central location for analysis. Data can be generated in smart meters, healthcare individual wearable devices, smartphones, or other IoT devices that do not possess enough computing power to process the acquired data.

Erdogdu et al. [67] investigates privacy in smart meter data. They assume an online setting where peers frequently want to release time series data to a central data miner. For example, data is generated from a smart meter in a smart grid application. Privacy is measured as the amount of mutual information between the original and distorted time series. They assume that peers holding data will not attempt inference attacks against each other. Unfortunately, it is unclear which distortion method was used to generate the distorted data. Moreover, the privacy analysis does not include aspects related to time. Wang et al. [218] devised a modified Laplace noise mechanism to add noise with the same autocorrelation of the original data. Consequently, noise added to sensitive time series can not be filtered out. Arzamasov et al. [13] surveyed privacy metrics applied to smart meters data. They argue that further research on privacy measures for time series is necessary to help users select a suited metric for a specific application.

Privacy-preserving Similarity Queries. Similarity query is a primitive operation in time series. It is part of many other tasks like clustering, pattern discovery, and anomaly detection.

Wang [220] proposed a scheme for k-nearest-neighbor and k-farthest-neighbor queries for multiple time series in a distributed environment. The idea is to transform the query into wavelet coefficients to save bandwidth and provide similarity bounds on the query sets stored at each machine. Liu et al. [138] also designed a system for similarity search over distributed time series. At each local hospital, time series are clustered, and cluster centers are communicated to a trusted server via secret sharing and additive homomorphic encryption. The trusted server also is responsible for pruning the search space and answering queries securely. Wang et al. [224] introduced PatternLDP to address the privacy-preserving time series publishing prob-

lem. They assume a distributed setting with several data providers and a non-trusted collector. Therefore, PatternLPD helps each data provider to ensure privacy is not compromised. The main idea is to sample and add noise only to relevant points that are sufficient to keep the main statistical information of the data. Privacy is measured with w-event privacy, extending differential privacy for w-sized subsequences of a time series. The goal is to allow for distributed similarity search without revealing original points.

5.7 Summary

This chapter investigated privacy-preserving pattern discovery in a distributed time series by transforming the time series into a set of n-dimensional data points. This approach was chosen because it allows nonparametric density estimation techniques to compute the most frequent patterns. Therefore, there is no need to assume any particular probabilistic model a priori. Current works on pattern discovery do not address the distributed case and, more specifically, do not address privacy issues. This study's primary contributions to this topic are summarized below.

We demonstrated that density estimates can be helpful in efficiently finding patterns in distributed time series. Our approach exploits the fact that density estimates are additive, and therefore, it is possible to aggregate local density estimates without exchanging raw data to obtain a global density estimate.

Three algorithms for pattern discovery in distributed time series were proposed: DPD-TS, DPD-FS, and DPD-HE. Since density estimates take up much less space than the original dataset, the overall approach is space and time efficient. It was also shown that the proposed algorithms are linear in the size of the time series. Table 5.4 depicts a quick comparison between the proposed algorithms.

Regarding privacy, the proposed algorithms allow users to define the desired amount of privacy for local datasets. Indeed, the amount of privacy at individual points in each local time series is controlled by choice of simple parameters values of alphabet size $\mid \Sigma \mid$, subsequence size n and discretization amount τ. The level of privacy is guaranteed even in the presence of inference attacks by collusion groups in the mining session.

	DPD-TS	**DPD-FS**	**DPD-HE**
Reference	Sec. 5.2	Sec. 5.4	Sec. 5.3
Privacy with outsider $\mathbf{PR}_{[0]}^{TBK}$	$\min\{(\beta_{j+1}-\beta_j)\frac{n}{w}\}$ [a]	τn [b]	∞
Privacy with single insider $\mathbf{PR}_{[1]}^{TBK}$	$\min\{(\beta_{j+1}-\beta_j)\frac{n}{w}\}$	τn	$\min\{(\beta_{j+1}-\beta_j)\frac{n}{w}\}$
Privacy with collusion $\mathbf{PR}_{[c>1]}^{TBK}$	$\min\{(\beta_{j+1}-\beta_j)\frac{n}{w}\}$	τn	$\min\{(\beta_{j+1}-\beta_j)\frac{n}{w}\}$
Num. of trusted parties required	0	0	0
Time Complexity [c]	$O(\lvert T_i\rvert)$	$O(\lvert T_i\rvert)$	$O(\lvert T_i\rvert E)$
Space Complexity	$O(\lvert T_i''\rvert)$	$O(\lvert T_i''\rvert)$	$O(\lvert T_i''\rvert)$
Number of rounds	2	2	2
Message size	$O(T_i'')$	$O(T_i'')$	$O(T_i'')$

Table 5.4: *Summary of proposed Distributed Pattern Discovery algorithms.*

[a] $\{\beta_j \in \mathbb{R}\}$, $j = 1, \ldots, |\Sigma|$ is the set of breakpoints used to discretize the time series, n is the subsequence size and w is the pattern size.
[b] τ is the discretization amount used in the discretization step and n is the subsequence size.
[c] $|T_i|$ is the size of time series dataset at peer L_i, T'' denotes a discretized version of a given time series T, r denotes the number of rounds and E is the overhead of the encryption and decryption operations.

Chapter 6

pp-Expert: Evaluation Tool

Previous chapters presented new privacy-preserving distributed data mining algorithms. However, the discussion focused on the algorithmic level and its properties without giving details on implementation. In this chapter, we discuss the implementation details of all proposed algorithms. Moreover, this chapter presents *privacy preserving experiments tool* (pp-Expert), an evaluation environment with which the experiments in this study were run. In the following, an overview of the main features is provided, examining the underlying architecture and giving details on using pp-Expert. A description of all datasets used in this study is also provided.

6.1 Overview of pp-Expert

6.1.1 Installation

pp-Expert is implemented in Java 1.8+ and is distributed as a single `.jar` file, which can be executed on any operating system with a Java virtual machine (JVM). To run pp-Expert, it is necessary to download the `.jar` file to any directory visible to the JVM (Java *classpath*), and it is ready to run. The main requirements are the following:

1. Java Virtual Machine (JVM) 1.8 or greater;
2. 4Gb RAM (recommended) or 2Gb RAM (minimum);
3. Core i3 processor or newer (recommended).

6.1.2 Features

The goal of pp-Expert is to allow for experiments that show how different parameter configurations affect execution time and data mining quality for the algorithms presented in this thesis. It is organized as a set of experiments presented in a single graphical user interface (cf. Fig. 6.1). The evaluation scenarios focus on the mining quality in function of privacy level, running time in function of privacy level, and running time in function of dataset size. See previous Chapters to recall the details of each experiment specific to each algorithm and Section 6.3 for details on a more detailed explanation of each parameter needed to run an experiment.

Currently, pp-Expert includes the following set of experiments:

- KDEC and KDEC-S: quality vs. privacy level, shows the influence of sampling parameter τ on the clustering quality given by the clustering error metric; time vs. privacy level, shows the running time as a function of sampling parameter τ; time vs. dataset size, investigates the running time with different datasets sizes; cf. Sections 4.2 and 4.3 for an in-depth discussion on each algorithm;

- DPD-TS and DPD-FS: efficacy test, investigates the ability to find patterns; time vs. time series size, shows the running time as a function of the length of a time series; and time vs. pattern size, investigates the influence of parameter w on the running time; cf. Sections 5.2 and 5.4 for details on each algorithm;

- DPD-HE only: time vs. cryptography parameter (a power of 2, typically 128), shows the running time with different cryptographic key lengths; cf. Section 5.3 to recall the specifics of the said algorithm.

Other features of the pp-Expert are:

- Property files: All parameters used for a given experiment session can be stored for later use in a java property file. In the interface, the buttons labeled *load properties* and *save properties* provide this feature.

- Statistics: All statistics generated in an experiment are stored in one individual file, and this data is stored in the path the user indicates. Moreover, all results are stored in simple text format and can be read by any other mathematical tool, e.g., Matematica, Matlab, or Octave.

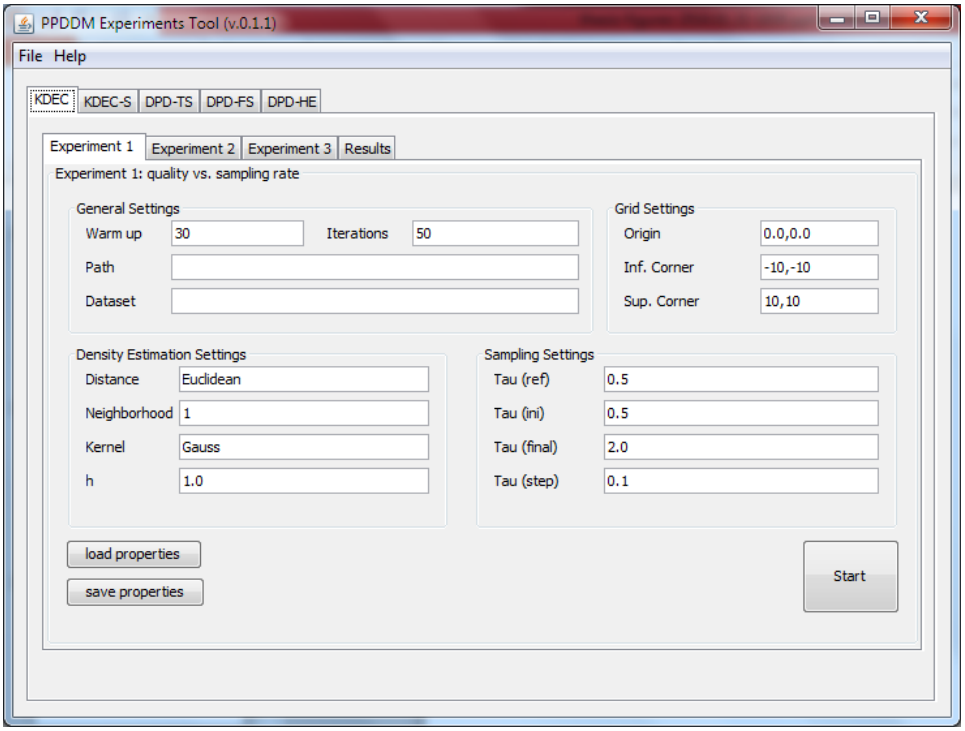


Figure 6.1: *pp-Expert's graphical user interface.*

- Results chart: after each experiment is completed, a chart summarizing the experiment's statistics is generated and displayed in the tab named *Results*. The chart is also saved as a `.jpeg` image in the specified output directory. Figure 6.4 shows an example of results from KDEC-S, experiment 1, which evaluates the impact of sampling rate *tau* on the cluster quality (cluster error).

In the following section, pp-Expert's architecture is discussed. Information on how to run experiments is presented in Section 6.3.

6.2 Architecture

6.2.1 pp-Expert Layers

pp-Expert is organized in the following layers: `view`, `controller`, `model`, `persistence` and `utilities`. Figure 6.2 gives an overview of the pp-Expert's architecture. Next sections discuss each layer[1] in details.

[1] Each layer is implemented in java as a package with the same name

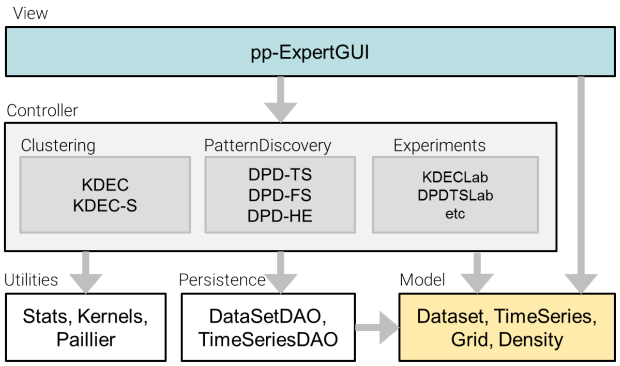


Figure 6.2: *Overview of layers and packages. Arrows indicate dependency.*

View Layer. The view layer provides the tool's graphical user interface. Its main class is `ExperToolGui`, which is the entry point to the tool and is composed of several panels, organized by algorithm and experiment (cf. Fig. 6.3). It allows the user to inform all parameters values and perform experiments on each algorithm individually. After each experiment, a line chart summarizing the experiment's statistics is generated and is displayed as a separate panel named *Results*. For instance, experiments measuring quality vs. sampling rate will display a chart where the x-axis shows the sampling range and the y-axis displays the error metric. Similarly, experiments measuring time vs. some parameter will display a chart where the x-axis shows the parameter range and the y-axis displays the time spent on a given run (cf. Fig. 6.4).

All elements in the interface are from the Java Swing API. The summarizing graph is generated with JFreeChart library.

Controller Layer. The controller layer is the core package of pp-Expert, where all algorithms are implemented. This layer groups together all classes that are responsible for coordination and control.

Classes `KDEC` and `KDEC-S` implement KDEC and KDEC-S algorithms, respectively. They coordinate a clustering session, ensure that all preconditions are met before invoking a cluster algorithm and takes care of all post-conditions following the algorithm completion. Classes `Estimator` and `SecureEstimator` implement density-estimation strategy through the method `estimate()` and returns a `DensityEstimate` object. Classes `DPDTS`, `DPDHE` and `DPDFS` implement pattern discovery algorithms.

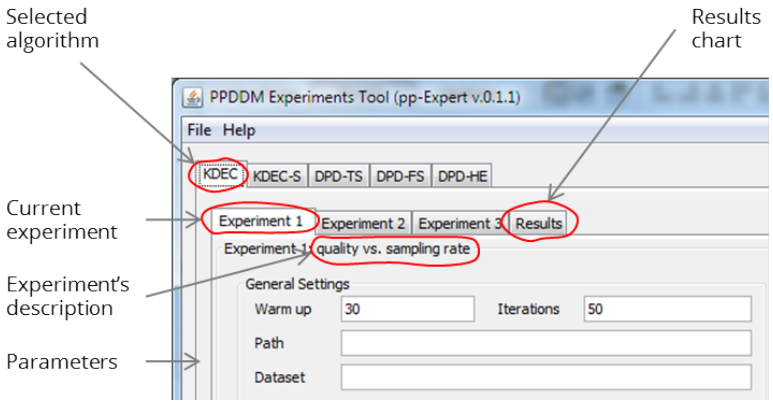


Figure 6.3: *Elements of pp-Expert GUI: algorithms tab, experiments tab, input parameters and resulting statistics graph.*

A group of special controllers for coordinating experiments are also part of this layer, e.g. `KDECLab`, `DPDTSLab`, etc. These controllers run a given algorithm several times with different parameter values and control the generation of statistical data for further analysis.

Model Layer. The model layer is composed of a set of classes that represent data used by other layers.

`DataSet` is a concrete class providing navigation behavior through data points. `Grid` class provides a virtual grid in the data space. It is used mainly as an index structure, allowing for navigation through a dataset.

`DensityEstimate` represents a density estimate of a given data set. It is implemented as a hash table, where each key refers to a list of points around a given grid point. `SecureDensityEstimate` stores a density estimate for any given data set and privacy parameters. In particular, this class provides the method `getClusterGuides()`, which returns a set of cluster guides, as defined by the KDEC-S algorithm.

`TimeSeries` is a class used to manipulate time series data, offering methods like average, min, max, size reduction, discretization. In the model layer, there is also `StringDensityEstimate`, a modified version of `DensityEstimate` that models density estimates of a discretized time series.

Persistence Layer. The persistence layer provides an abstraction for different storage technology. Its main class is `DataSetDAO`, an abstract class that defines the basic behavior required for storing and retrieving collections

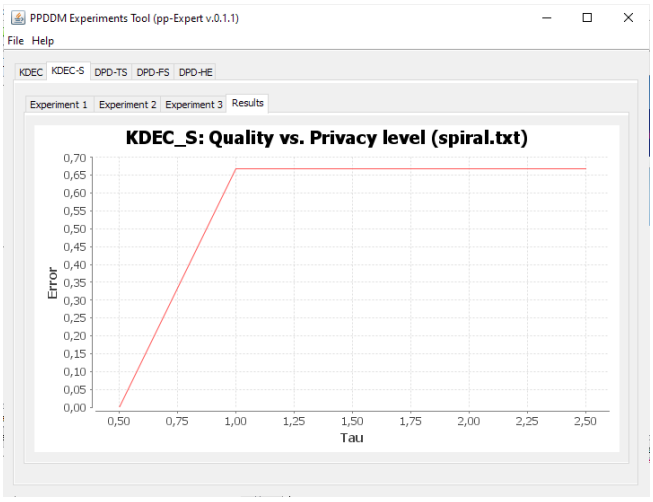


Figure 6.4: *The results tab in the pp-Expert GUI: after an experiment is running, the results tab shows a chart with the selected statistics.*

of data points. `DataSetDAOPlainFile` is an implementation of `DataSetDAO` that handle numeric data points from a text files. Similarly, `TimeSeriesDAO` is an abstract class for time series data storage and `TimeSeriesDAOPlainFile` is a concrete implementation that reads data stored in a simple text file.

Utility Layer. The utility layer contains all classes that provide general functionality but do not fit anywhere else.

`Distance` is a Java interface that defines an abstract distance function. There are currently several implementations of this interface, including `EuclideanDistance`, `ManhattanDistance`, `Levenshtein`, etc.

`Kernel` is a Java interface that defines an abstract kernel function. Concrete classes that implements kernel are `Gauss` and `Triangle`.

`Paillier` is a GNU-licensed implementation of Paillier cryptosystem developed by Omar Hasan[2]. It is distributed under the GPLv3 license as a single java file, which we included in our `util` package.

`Stats` is an implementation of inverse normal cumulative distribution, ported to Java by Sherali Karimov[3], from an original Perl implementation by Petr J. Acklam.

[2] `http://liris.cnrs.fr/~ohasan/pprs/paillierdemo/`
[3] `http://home.online.no/~pjacklam/notes/invnorm/`

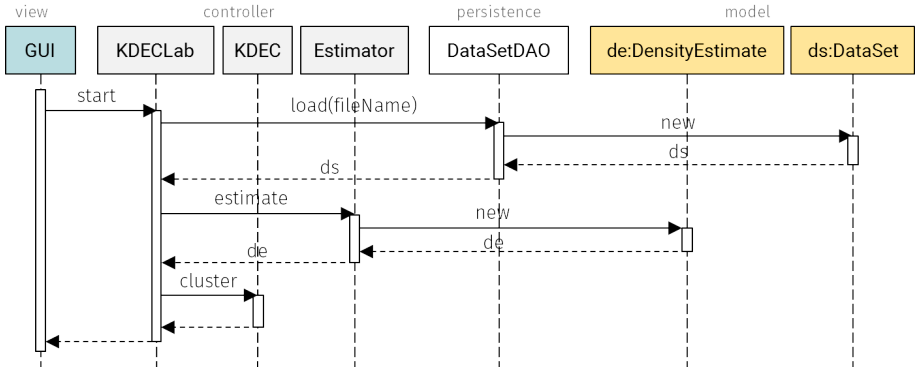


Figure 6.5: *Sequence diagram of KDEC algorithm with classes from all layers.*

Third Party Libraries. We use PtPlot 5.7 for data plots in the primary GUI. PtPlot is a 2D data plotter and histogram tool developed by Edward A. Lee and Christopher Brooks at UC Berkeley EECS Dept. as part of the Ptolemy project[4]. PtPlot is implemented in Java, which can be used as a standalone applet or application. It can also be embedded in your applet or application. The main class used in this study's implementation was `Plot`.

We use JFreeChart 1.0.14 for plotting charts in the testing tool. JFreeChart[5] is a java library distributed under the GLP license. It supports various charts, including scatter plots and time series charts. JFreeChart is maintained by David Gilbert.

6.2.2 Class Interactions

A typical sequence of interactions among the classes pp-Expert is given in Figure 6.5. This is a sequence diagram for an experiment with KDEC. The classes involved in this example are: `KDECLab`, `KDEC` and `Estimator` in control layer; `DensityEstimate` and `DataSet` in model layer; and `DataSetDAO` from the persistence layer.

The sequence starts when any relevant event is detected at the GUI. `KDECLAb` then carries on several preparatory steps before it invokes the specific clustering algorithm, e.g., load data from a file into a `DataSet` object in memory, get the complete experiment specification via parameters. It also starts the estimation process, which produces a `DensityEstimate` object. Finally, with the dataset and density objects already created, the `KDEC`

[4] `http://ptolemy.eecs.berkeley.edu/java/ptplot/`
[5] `http://www.jfree.org/`

algorithm class is invoked to produce a cluster map.

`KDECLAb` repeats the process, as indicated by the number of iterations parameter. After running all iterations, `KDECLab` generates and saves statistics on the specified location. After that, `KDECLAb` returns control to the GUI, and a chart with a summary of the statistics is generated on the tab labeled *Results*.

The same overall idea is used in all other experiments.

6.3 Running an Experiment on pp-Expert

Under Windows, pp-Expert can be started by double-clicking on the `.jar` file. To start it from the command line, type[6]:

```
java -jar PPDDMExperTool.jar
```

After the GUI is displayed, proceed as follows.

1. Select an algorithm from the algorithms tabs, e.g., KDEC-S.
2. Select an experiment. For example, experiment 1, which computes the quality of data mining as a function of the sampling rate (τ).
3. In the *general settings* section, inform:
   - **warmup**: the number of iterations to be run before starting the clock. This is required as JVM tends to have a large variance in performance early on the first iterations.
   - **iterations**: the number of iterations after warm-up; typically 50 (minimum 30 for statistically significant results).
   - **path**: location to the output's directory.
   - **dataset**: name of the dataset to be used in this experiment, e.g. "polar.dat"
4. In the *grid settings* section, inform:
   - **origin**: a tuple indicating where the grid is centered; typically a tuple with zero in all dimensions, e.g. 0,0,0 for a 3-dimensional dataset.

[6] Assuming that this command is issued in the same directory where the PPDDMExperTool.jar file is located

- **inferior and superior corners**: tuples indicating the extreme points (minimal and maximal) in the hyperrectangle defined by the grid representing the range of values covered by the data;

5. In the *density estimation setting* section, inform:

   - **distance function**: a string with the name of a distance function, e.g. Euclidean, Manhattan, etc.
   - **neighborhood**: radius of the neighborhood used to compute the density of a given point; a typical value is 1.0.
   - **kernel**: a string with the name of kernel function to compute the density estimates, e.g. Gaussian, triangle, Epanechnickov, etc.
   - **h**: the kernel bandwidth; typically is same as the neighborhood radius.

6. In the *sampling settings* section, inform:

   - **reference** τ (tau): is a real number indicating the sampling rate of each dimension; in experiment 1, a reference cluster map is generated with this information and subsequent cluster maps are compared with it when computing its quality;
   - **initial** τ: the starting value of τ used in experiments that investigates the effect of differents sampling rates on the quality of the mining result;
   - **final** τ: the final value of τ in the experiment, used during experiments that varies this parameter;
   - **step**: the size of each increment, from the first until the final value of τ, , used during experiments that varies this parameter;

7. To save all parameters in this session, use the button *save properties*.

8. Click the *start* button.

When the experiment is over, an output file is created. The file name indicates the algorithm, experiment number, short description of the experiment, dataset name, and main parameters. For example, assuming KDEC algorithm was selected, and experiment 1 (quality vs. privacy) was run with

dataset named "polar.dat" with neighborhood $n = 1$ and kernel bandwidth $h = 1.0$, the resulting file name[7] is:

`KDEC_exper1_quali_priv_polar_n1_h1.0.out`

A chart with the experiments results showing how the investigated parameter affected a given aspect is also generated with the same name and extension `.jpg`. For example, Figure 6.4 shows how cluster error is affected by different values of sampling parameter τ (tau) when running KDEC-S.

6.4 Datasets

This section gives an overview of each dataset utilized in this study.

Datasets for Clustering

We created several synthetic datasets for clustering. Well-known datasets were also used to evaluate density-based clustering.

Synthetic datasets. Sample datasets were created, consisting of points generated from a mixture model with four Gaussians, each with variance σ^2 =1 in all dimensions. The idea is to simulate a dataset with four clusters (cf. Fig. 6.6). A dataset with 500 points was also generated, built to perform basic tests. Datasets with 5K, 10K, 15K, 20K, 25K, and 30K points were also created to analyze time as a function of the size of the dataset.

Polar dataset. A dataset with 400 points was also generated, 200 points of which were generated from a Gaussian with $\mu = 0$ and $\sigma^2 = 5$ and 200 points generated around the center with radius $R \sim N(20, 1)$ and angle uniformly distributed from $\theta \sim U(0, 2\pi)$. This is a more interesting dataset because clusters are not easily defined (cf. Fig. 6.7).

Synthetic dataset S1 The $S1$ is a synthetic dataset[8] from [74] was utilized, containing 5000 points distributed in 15 predefined clusters (cf. Fig. 6.8).

[7] All parameter values are indicated in a string of comments in the first line of the file; the comment line starts with the character '#'.

[8] `http://cs.joensuu.fi/sipu/datasets/`

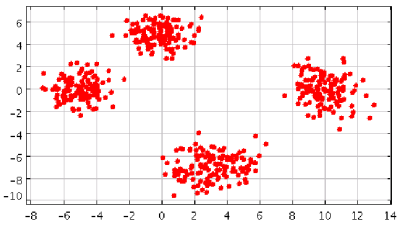


Figure 6.6: *`Gaussian` dataset: data points generated from a mixture of 4 Gaussians.*

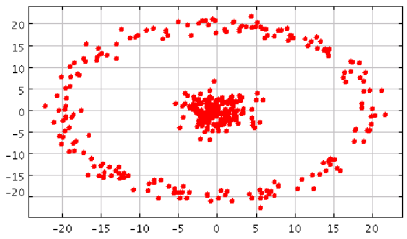


Figure 6.7: *`Polar` dataset: Points are distributed along an arbitrary shape, with two clusters.*

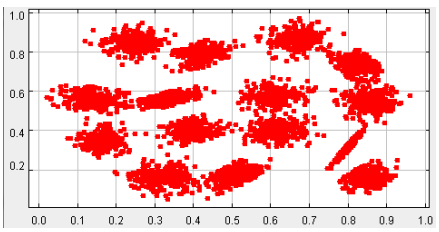


Figure 6.8: *`S1` dataset: data points generated from a mixture of 15 Gaussians.*

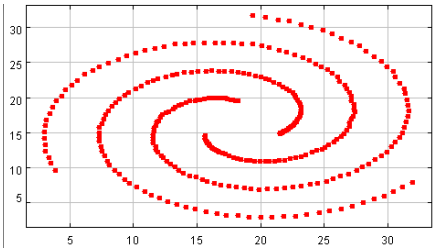


Figure 6.9: *`Spiral` dataset: data points form three different spirals.*

The Spiral dataset. The spiral dataset[9] is a synthetic dataset from [36] was also utilized, which contains 312 points and 3 clusters (cf. Fig. 6.9).

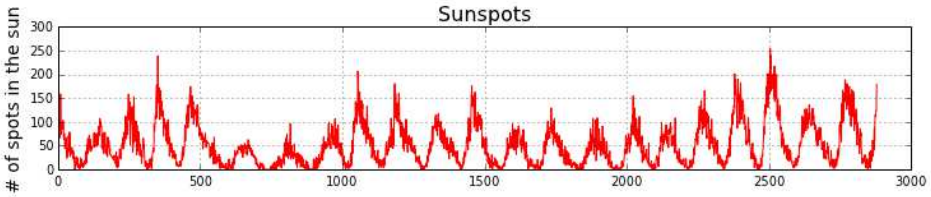
Datasets for Pattern Discovery

The following time series datasets were utilized in this study: `power`, `sunspot`, and `tide` datasets. All datasets were downloaded from UCR Time Series Data Homepage[10] [56], except the synthetic ones.

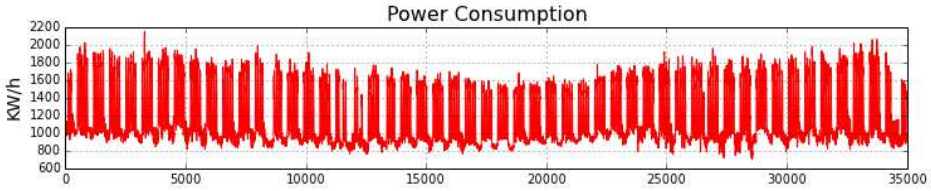
Sunspot dataset. This dataset records the monthly average number of sunspots from January 1749 until 1993. There are 2 880 data points in this dataset[11] (cf. Fig. 6.10(a)).

Power dataset. This dataset presents the electricity consumption from Netherlands Energy Research Foundation (ECN) for one year, recorded

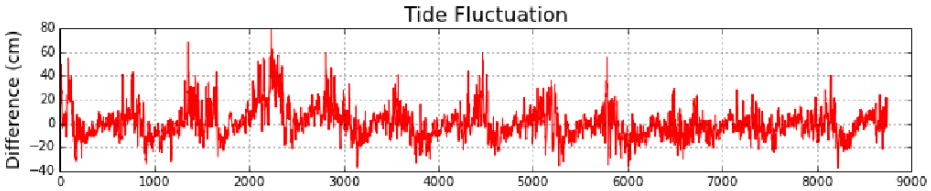
[9] `http://cs.joensuu.fi/sipu/datasets/`
[10] `https://www.cs.ucr.edu/~eamonn/time_series_data_2018/`
[11] For up to date sunspot data visit `http://sidc.oma.be/`



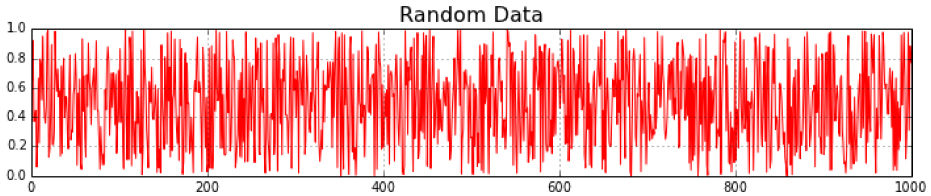
(a) `sunspot` dataset, size=2 880



(b) `power` dataset, size = 35 000



(c) `tide` dataset, size=8 700



(d) Excerpt of `random` dataset

Figure 6.10: *Time series datasets.*

every 15 minutes. There are 35 040 data points corresponding to the year 1997. This dataset has a pattern structure that can be observed visually (cf. Fig. 6.10(b)).

Tide dataset. This dataset is a collection of 12 years of tide fluctuations at Crescent City, Northern California, recorded by the National Ocean Service (NOA) from January 1980 to December 1991[12]. There are two records per day: 8 746 data points in this series. Fluctuation is measured in centimeters in relation to mean low water level (cf. Fig. 6.10(c)).

Synthetic time series dataset collection. We also generated a collection of datasets based on random data, which was used in experiments where running time was evaluated in function of dataset size. The collection consists of 10 dataset with 100 000 points, 200 000 points, up to 1 000 000 points. A small excerpt of a random time series is shown in Fig. 6.10(d).

6.5 Availability

Source code of pp-Expert and datasets used in this study are available upon request. All algorithms implemented in this study are integrated into pp-Expert, distributed as an executable `.jar` file. Alternatively, there is a different distribution with algorithms only, distributed as an API in a non-executable `.jar` file.

[12] `https://tidesandcurrents.noaa.gov/sltrends/sltrends_station.shtml?id=9419750`

Chapter 7

Conclusion and Outlook

We live in the information era. Using web-based applications like video streams and social networks is so natural that we do not even think about the massive amount of information collected about our preferences and behaviors.

With all the benefits of global networking, individual privacy has become a hot topic of discussion. On the one hand, we want all the information available and, in turn, the knowledge that can be derived from it. On the other hand, we do not want to sacrifice our privacy in the process. Disclosing strategic information in a business-to-business scenario can even bring more harm than good, and from an individual point of view, there is an increasing awareness that personal data is a valuable asset. In this context, privacy-preserving data mining opens up many possibilities, enabling scenarios where data privacy plays a key role. In intelligent business applications spanning several organizations, e.g., in a B2B system, datasets are likely to represent trade secrets. Similarly, medical data records cannot be freely distributed since rigid privacy regulations protect them. Only with a guarantee of privacy preservation enterprises or clinics will want to participate in a distributed mining endeavor.

This thesis investigated the general question of how to provide *good* mining results while preserving data *privacy*. This question involves defining what is meant by privacy and good mining result. In the following, the main contributions of this thesis toward answering this general question are presented.

7.1 Summary of Contributions

Before discussing the lessons learned throughout this study, it is best to revisit the primary research question posed at the beginning of this investigation.

The general question concerned how to provide both privacy and, at the same time, good mining results in a distributed data environment. The whole field of privacy-preserving data mining is trying to answer this question. This thesis contributes to the big picture of building a privacy-preserving data mining system as summarized in the following.

1. **Privacy Measures for DDC and DTS Mining**

   How to define and formalize the concept of privacy and corresponding privacy measure in a distributed environment taking into account inference attacks and collusion of malicious insiders? Many privacy definitions exist in privacy-preserving data mining literature, reflecting different assumptions and focuses of interest. In Chapter 2, the different existing privacy formalizations for distributed data mining, their assumptions, and limitations are reviewed. It is shown that existing privacy measures do not satisfy all properties required in distributed environments with inference attacks and collusion. In Chapter 3, new privacy measures are proposed to address inference attack scenarios while providing an intuitive interpretation of privacy. These new privacy measures are tied to the more specific mining tasks – namely, distributed data clustering and pattern mining in time series.

   What kind of attacks to sensitive data owned by each participant in a distributed data mining setting can take place? This question is addressed in Chapters 4, analyzing the KDEC scheme, and Chapter 5, analyzing this study's algorithms for pattern discovery (DPD-TS, DPD-HE and DPD-FS). The main threats identified are inference attacks, allowing for the reconstruction of sensitive data, and the collusion problem, when a subset of the members of a mining group colludes to improve the results of their attacks. These threats are part of what is known as insider attacks, named as such since it comes from the members of a mining group. Another type of attack is also possible, coming from eavesdroppers – agents that are not part of the mining

group. These threats are also analyzed, although they have less chance of happening when current security technology is applied.

2. **Privacy-Preserving Distributed Mining Algorithms**

   How to give a guarantee that DDC and DTS preserve a certain privacy level? Formalizing privacy is important because it allows for the finding of a privacy metric to assess the privacy level of a given dataset or a data mining algorithm. Different privacy metrics for clustering and pattern discovery are proposed in this study (in Ch. 3), following the overall idea that privacy level is the reciprocal of certainty. In this context, all proposed algorithms allow the user to specify the required level of reconstructed data using privacy parameters. Therefore, in a general sense, users can control the size of the interval where the attacker can be sure a given reconstructed point occurs, trading off quality for privacy.

   Distributed Data Clustering. The KDEC is analyzed concerning privacy, and, subsequently, an inference attack is developed against it, exploiting the fact that density estimates are too precise, even when sampled. This study proposed an improved algorithm KDEC-S for distributed data clustering. KDEC and KDEC-S are based on a distributed density estimation process. The privacy in KDEC-S comes from the fact that it uses an imprecise estimation process, where the imprecision leads to a given level of uncertainty via a sampling parameter τ. KDEC-S provides more privacy than KDEC keeping the same mining quality. Moreover, it is linear in the number of data points in the dataset. Compared to the generative models approach, KDEC-S does not require the number of clusters a priori can find arbitrary shape clusters while providing the same level of privacy in less time. Compared to secure multi-party approaches, KDEC-S trades off privacy level for communication and time efficiency, while SMC protocols are designed to protect input data from outsiders they do not avoid inference attacks by malicious insiders. KDEC-S guarantees its privacy levels even in the presence of collusion of malicious insiders.

   Distributed Time Series. This thesis also proposed algorithms DPD-TS, DPD-HE, and DPD-FS for frequent pattern discovery in time series data. In particular, this study's pattern discovery algorithms are

particularly innovative in that they are among the first to address privacy in the context of distributed time series. The main assumption is that we can represent subsequences of a time series as a data point in a n-dimensional space. Moreover, this study tested the hypothesis that finding frequent patterns could be reduced to finding local maxima in a n-dimensional density space. The hypothesis turns out to be true, and all three algorithms have good scalability on the number of data points. DPD-TS directly applies these ideas, where privacy is controlled by a combination of two parameters: alphabet and subsequence size. DPD-HE goes a step further, using secure sum and homomorphic encryption to provide an additional security layer, trading off performance for security. DPD-FS presents an alternative approach to improve the pattern discovery process – giving up some degree of performance. DPD-FS makes fewer assumptions, needs fewer parameters and has a precise privacy level control mechanism given by a combination of discretization amount and subsequence size. All three algorithms are linear on the size of time series, and all of them provide privacy guarantees under different inference attack scenarios even with the collusion of malicious insiders.

7.2 Further Directions

Privacy-preserving data mining remains an ongoing topic of research. Many works have been proposed in recent years, and multiple research efforts are being conducted right now. Many new directions of research can be pursued.

An interesting facet of data mining to pursue is how to provide data mining algorithms with the anytimeness property. *Anytimeness* is the property of having a partial answer whenever the user may choose to request it [235]. This property is very useful in data mining because it allows the user to get partial results. Thus, users may decide whether the chosen parameter values are correct and need not wait a couple of hours to discover that a given parameter value is wrong.

It would also be worth investigating how to apply *differential privacy* to the algorithms presented in this thesis and try to link our privacy measures with the choice of privacy budget ϵ. This extension would provide an extra layer of privacy protection to our approaches and would be a hybrid approach

[229].

Federated Learning presents new challenges with a massive number of devices with limited resources [122]. Thus, extending our proposed algorithm to the *federated learning* setting seems a valuable line of research. This extension could open up many application possibilities, including large-scale mobile-based applications.

Bibliography

[1] A. Acquisti, L. Brandimarte and G. Loewenstein (2015): Privacy and human behavior in the age of information. *Science*, vol. 347(6221), pp. 509–514.

[2] M. Adams (2017): Big Data and Individual Privacy in the Age of the Internet of Things. *Technology Innovation Management Review*, vol. 7, pp. 12–24.

[3] C. C. Aggarwal (2018): An introduction to cluster analysis. In *Data Clustering*, pp. 1–28. Chapman and Hall/CRC.

[4] D. Agrawal and C. C. Aggarwal (2001): On the Design and Quantification of Privacy Preserving Data Mining Algorithms. In *Proc. of the 20th Symp. on Principles of Database Systems (PODS)*, pp. 247–255. ACM.

[5] S. Alaee, K. Kamgar and E. Keogh (2020): Matrix Profile XXII: Exact Discovery of Time Series Motifs Under DTW. In *Intl. Conf. on Data Mining (ICDM)*, pp. 900–905. IEEE.

[6] K. A. Albashiri, F. Coenen and P. Leng (2009): EMADS: An Extendible Multi-agent Data Miner. *Knowledge-Based Systems*, vol. 22(7), pp. 523–528.

[7] T. Allard, G. Hébrail, F. Masseglia and E. Pacitti (2015): Chiaroscuro: Transparency and Privacy for Massive Personal Time-Series Clustering. In *Proc. of the Intl. Conf. on Management of Data (SIGMOD)*, pp. 779–794. ACM.

[8] N. Almutairi, F. Coenen and K. Dures (2018): Secure Third Party Data Clustering Using Φ-Data: Multi-User Order Preserving Encryp-

tion and Super Secure Chain Distance Matrices. In *Proc. of Intl. Conf. on Innovative Techniques and Applications of Artificial Intelligence (SGAI)*, pp. 3–17. Springer.

[9] R. Altilio, P. Di Lorenzo and M. Panella (2019): Distributed data clustering over networks. *Pattern Recognition*, vol. 93, pp. 603 – 620.

[10] R. Alvarez and M. Nojoumian (2020): Comprehensive survey on privacy-preserving protocols for sealed-bid auctions. *Computers & Security*, vol. 88, p. 101502.

[11] F.-A. Ana, M.-S. Loreto, L.-M. M. José, S. M. Pablo, M. J. María Pilar and S.-L. A. Myriam (2020): Mobile applications in oncology: A systematic review of health science databases. *International Journal of Medical Informatics*, vol. 133, p. 104001.

[12] C. Anda and S. A. Ordonez Medina (2019): Privacy-by-design generative models of urban mobility. *Arbeitsberichte Verkehrs-und Raumplanung*, vol. 1454.

[13] V. Arzamasov, R. Schwerdt, S. Karrari, K. Böhm and T. B. Nguyen (2020): Privacy Measures and Storage Technologies for Battery-Based Load Hiding - an Overview and Experimental Study. In *Proc. of 11th Intl. Conf. on Future Energy Systems*, e-Energy, p. 178–195. ACM.

[14] G. Ateniese, L. V. Mancini, A. Spognardi, A. Villani, D. Vitali and G. Felici (2015): Hacking Smart Machines with Smarter Ones: How to Extract Meaningful Data from Machine Learning Classifiers. *Int. Journal of Security and Networks*, vol. 10(3), p. 137–150.

[15] S. Augenstein, H. B. McMahan, D. Ramage, S. Ramaswamy et al. (2020): Generative Models for Effective ML on Private, Decentralized Datasets. In *8th Intl. Conf. on Learning Representations, (ICLR)*. OpenReview.net.

[16] R. Aydogan, D. Festen, K. Hindriks and C. Jonker (2017): Alternating Offers Protocols for Multilateral Negotiation. In *Modern Approaches to Agent-based Complex Automated Negotiation*, *Studies in Computational Intelligence*, vol. 674, pp. 153–167. Springer.

[17] A. Bagnall, J. Lines, A. Bostrom, J. Large and E. Keogh (2017): The great time series classification bake off: a review and experimental evaluation of recent algorithmic advances. *Data mining and knowledge discovery*, vol. 31(3), pp. 606–660.

[18] F. J. Baldán and J. M. Benítez (2019): Distributed Fastshapelet Transform: a big data time series classification algorithm. *Information Sciences*, vol. 496, pp. 451–463.

[19] F. Benhamouda, M. Joye and B. Libert (2016): A new framework for privacy-preserving aggregation of time-series data. *ACM Trans. on Information and System Security*, vol. 18(3), pp. 1–21.

[20] E. Bertino, D. Lin and W. Jiang (2008): A Survey of Quantification of Privacy Preserving Data Mining Algorithms. In *Privacy-Preserving Data Mining: Models and Algorithms*, *Advances in Database Systems*, vol. 34, pp. 183–205. Springer.

[21] M. Bezzi (2010): An information theoretic approach for privacy metrics. *Trans. Data Priv.*, vol. 3(3), pp. 199–215.

[22] A. Bhowmik, J. Ghosh and O. Koyejo (2019): Aggregation for Sensitive Data. In *Proc. of 13th Intl. Conf. on Sampling Theory and Applications (SampTA)*, pp. 1–5.

[23] J. Biskup (2016): Selected Results and Related Issues of Confidentiality-Preserving Controlled Interaction Execution. In *FoIKS*, pp. 211–234. Springer.

[24] A. Bittau, U. Erlingsson, P. Maniatis, I. Mironov et al. (2017): PROCHLO: Strong Privacy for Analytics in the Crowd. In *Proc.of the 26th Symp. on Operating Systems Principles*, p. 441–459. ACM.

[25] A. Blázquez-García, A. Conde, U. Mori and J. A. Lozano (2021): A Review on Outlier/Anomaly Detection in Time Series Data. *ACM Computing Surveys (CSUR)*, vol. 54(3), pp. 1–33.

[26] D. Bogdanov (2013): *Sharemind: Programmable Secure Computations with Practical Applications*. Ph.D. thesis, University of Tartu.

[27] B. Bozdemir, S. Canard, O. Ermis, H. Möllering, M. Önen and T. Schneider (2021): Privacy-preserving density-based clustering. In *Proc. of the Asia Conf. on Computer and Communications Security*, pp. 658–671.

[28] M. Brandizi, O. Melnichuk, R. Bild, F. Kohlmayer et al. (2017): Orchestrating differential data access for translational research: a pilot implementation. *BMC Medical Informatics and Decision Making*, vol. 17(1), p. 30.

[29] L. Breiman (1996): Bagging Predictors. *Machine Learning*, vol. 24(2), pp. 123–140.

[30] A. Brodsky, C. Farkas, D. Wijesekera and X. Wang (2000): Constraints, Inference Channels and Secure Databases. In *Proc. of Intl. Conf. on Principles and Practice of Constraint Programming (CP)*, pp. 98–113. Springer.

[31] Z. Cai, Z. He, X. Guan and Y. Li (2016): Collective Data-Sanitization for Preventing Sensitive Information Inference Attacks in Social Networks. *Trans. on Dependable and Secure Computing*, vol. 15(4), pp. 577–590.

[32] D. Calvaresi, M. Schumacher and J.-P. Calbimonte (2020): Personal Data Privacy Semantics in Multi-Agent Systems Interactions. In *Advances in Practical Applications of Agents, Multi-Agent Systems, and Trustworthiness(PAAMS)*, pp. 55–67. Springer.

[33] S. Carpov, T. H. Nguyen, R. Sirdey, G. Constantino and F. Martinelli (2016): Practical Privacy-Preserving Medical Diagnosis Using Homomorphic Encryption. In *Proc. of the 9th Intl. Conf. on Cloud Computing (CLOUD)*, pp. 593–599.

[34] M. Chamikara, P. Bertok, D. Liu, S. Camtepe and I. Khalil (2018): Efficient data perturbation for privacy preserving and accurate data stream mining. *Pervasive and Mobile Computing*, vol. 48, pp. 1–19.

[35] M. Chamikara, P. Bertok, D. Liu, S. Camtepe and I. Khalil (2020): Efficient privacy preservation of big data for accurate data mining. *Information Sciences*, vol. 527, pp. 420–443.

[36] H. Chang and D.-Y. Yeung (2008): Robust Path-based Spectral Clustering. *Pattern Recognition*, vol. 41(1), pp. 191–203.

[37] S. Chatel, A. Pyrgelis, J. R. Troncoso-Pastoriza and J.-P. Hubaux (2021): SoK: Privacy-Preserving Collaborative Tree-based Model Learning. *Proceedings on Privacy Enhancing Technologies*, vol. 3, pp. 182–203.

[38] J. Chattratichat, J. Darlington, Y. Guo, S. Hedvall, M. Köhler and J. Syed (1999): An Architecture for Distributed Entrerprise Data Mining. In *Proc. of the 7th Intl. Conf. on High-Performance Computing and Networking (HPCN Europe)*. Springer.

[39] Z. Chen, C. Fiandrino and B. Kantarci (2021): On blockchain integration into mobile crowdsensing via smart embedded devices: A comprehensive survey. *Journal of Systems Architecture*, vol. 115, p. 102011.

[40] A. Cheu, A. Smith, J. Ullman, D. Zeber and M. Zhilyaev (2019): Distributed Differential Privacy via Shuffling. In *Proc. of Advances in Cryptology (EUROCRYPT)*, pp. 375–403. Springer.

[41] A. Cheu and J. Ullman (2021): The Limits of Pan Privacy and Shuffle Privacy for Learning and Estimation. In *Proc. of 53rd ACM Symp. on Theory of Computing (STOC)*, STOC 2021, p. 1081–1094. ACM.

[42] B. Chiu, E. Keogh and S. Lonardi (2003): Probabilistic Discovery of Time Series Motifs. In *Proc. of 9th Intl. Conf. on Knowledge Discovery and Data Mining*, KDD, pp. 493–498. ACM.

[43] C. Clifton, M. Kantarcioglu, J. Vaidya, X. Lin and M. Zhu (2002): Tools for Privacy Preserving Data Mining. *ACM SIGKDD Explorations Newsletter*, vol. 4(2), pp. 28–34.

[44] R. Corizzo, G. Pio, M. Ceci and D. Malerba (2019): DENCAST: Distributed density-based clustering for multi-target regression. *Journal of Big Data*, vol. 6(1), pp. 1–27.

[45] J. C. da Silva, O. A. C. Cortes, G. H. B. Oliveira and M. Klusch (2012): Density-based Pattern Discovery in Distributed Time Series.

In *Proc. of the 21st Brazilian Symp. on Artificial Intelligence (SBIA)*, pp. 62–71. Springer.

[46] J. C. da Silva, C. Giannella, R. Bhargava, H. Kargupta and M. Klusch (2005): Distributed Data Mining and Agents. *Engineering Applications of Artifical Intelligence*, vol. 4(18), pp. 791–807.

[47] J. C. da Silva and M. Klusch (2005): Inference on Distributed Data Clustering. In *Proc. of the 4th Machine Learning and Data Mining in Pattern Recognition (MLDM)*, pp. 610–619. Springer.

[48] J. C. da Silva and M. Klusch (2006): Inference in Distributed Data Clustering. *Engineering Applications of Artificial Intelligence*, vol. 19(4), pp. 363–369.

[49] J. C. da Silva and M. Klusch (2007): Privacy-preserving Discovery of Frequent Patterns in Time Series. In *Proc. of the 7th Industrial Conf. on Advances in Data Mining (ICDM)*, pp. 318–328. Springer.

[50] J. C. da Silva and M. Klusch (2007): Privacy Preserving Pattern Discovery in Distributed Time Series. In *Proc. of the 3rd Intl. Work. on Privacy Data Management (PDM)*, pp. 207–214. IEEE.

[51] J. C. da Silva, M. Klusch and S. Lodi (2016): Privacy-Awareness of Distributed Data Clustering Algorithms Revisited. In *Proc. of the 15th Intl. Conf. on Intelligence Data Analysis (IDA)*. Springer.

[52] J. C. da Silva, M. Klusch, S. Lodi and G. Moro (2004): Inference Attacks in Peer-to-Peer Homogeneous Distributed Data Mining. In *Proc. of the 16th European Conf. on Artificial Intelligence (ECAI)*, pp. 450–454. IOS Press.

[53] J. C. da Silva, M. Klusch, S. Lodi and G. Moro (2006): Privacy-preserving Agent-Based Distributed Data Clustering. *Journal of Web Intelligence and Agent Systems*, vol. 4(2), pp. 221–238.

[54] J. C. da Silva, G. H. Oliveira, S. Lodi and M. Klusch (2017): Clustering Distributed Short Time Series with Dense Patterns. In *Proc. of the 16th Intl. Conf. on Machine Learning and Applications (ICMLA)*. IEEE.

[55] M. Daniels and C. Farkas (2018): Health data privacy: A case of undesired inferences. In *Intl. Conf. on Biomedical & Health Informatics (BHI)*, pp. 291–294. IEEE, IEEE.

[56] H. A. Dau, E. Keogh, K. Kamgar, C.-C. M. Yeh et al. (2018). The UCR Time Series Classification Archive. Available at www.cs.ucr.edu/∼eamonn/time_series_data_2018/.

[57] J. Dean and S. Ghemawat (2008): MapReduce: Simplified Data Processing on Large Clusters. *Communications of the ACM*, vol. 51(1), pp. 107–113.

[58] D. Denning and J. Schlorer (1983): Inference Controls for Statistical Databases. *Computer*, vol. 16(7), pp. 69 –82.

[59] D. Desfontaines and B. Pejó (2020): Sok: Differential privacies. *Proceedings on Privacy Enhancing Technologies*, vol. 2020(2), pp. 288–313.

[60] M. C. Doganay, T. B. Pedersen, Y. Saygin, E. Savaş and A. Levi (2008): Distributed Privacy Preserving K-means Clustering with Additive Secret Sharing. In *Proc. of Intl. Workshop on Privacy and Anonymity in Information Society (PAIS)*, pp. 3–11. ACM.

[61] X. Dong, Z. Yu, W. Cao, Y. Shi and Q. Ma (2020): A survey on ensemble learning. *Frontiers of Computer Science*, vol. 14(2), pp. 241–258.

[62] T. Dugan and X. Zou (2016): A Survey of Secure Multiparty Computation Protocols for Privacy Preserving Genetic Tests. In *1st Intl. Conf. on Connected Health: Applications, Systems and Engineering Technologies*, CHASE, pp. 173–182. IEEE.

[63] C. Dwork, F. McSherry, K. Nissim and A. Smith (2006): Calibrating Noise to Sensitivity in Private Data Analysis. In *Theory of Cryptography*, pp. 265–284. Springer.

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

[65] D. Eckhoff and I. Wagner (2017): Privacy in the Smart City — applications, technologies, challenges, and solutions. *IEEE Communications Surveys & Tutorials*, vol. 20(1), pp. 489–516.

[66] M. Elliot and J. Domingo-Ferrer (2018): The Future of Statistical Disclosure Control. *arXiv preprint arXiv:1812.09204*.

[67] M. Erdogdu, N. Fawaz and A. Montanari (2015): Privacy-Utility Trade-Off for Time-Series with Application to Smart-Meter Data. In *Proc. of Workshops on Computer Sustainability at the 29th Conf. on Artificial Intelligence*, pp. 32–36. AAAI.

[68] D. Evans, V. Kolesnikov and M. Rosulek (2017): A pragmatic introduction to secure multi-party computation. *Foundations and Trends in Privacy and Security*, vol. 2(2-3).

[69] A. Fakhrazari and H. Vakilzadian (2017): A Survey on Time Series Data Mining. In *Proc. of Intl. Conf. on Electro Information Technology (EIT)*, pp. 476–481. IEEE.

[70] C. Farkas and S. Jajodia (2002): The Inference Problem: A Survey. *ACM SIGKDD Explorations Newsletter*, vol. 4(2), pp. 6–11.

[71] K. Fawagreh, M. M. Gaber and E. Elyan (2014): Random forests: from early developments to recent advancements. *Systems Science & Control Engineering*, vol. 2(1), pp. 602–609.

[72] U. Fayyad, G. Piatetsky-Shapiro and P. Smyth (1996): From Data Mining to Knowledge Discovery in Databases. *AI Magazine*, vol. 17(3), pp. 37–54.

[73] P. Fournier-Viger, J. C.-W. Lin, B. Vo, T. T. Chi, J. Zhang and H. B. Le (2017): A survey of itemset mining. *WIREs Data Mining and Knowledge Discovery*, vol. 7(4), p. e1207.

[74] P. Fränti and O. Virmajoki (2006): Iterative Shrinking Method for Clustering Problems. *Pattern Recognition*, vol. 39(5), pp. 761–775.

[75] A. Galicia, R. Talavera-Llames, A. Troncoso, I. Koprinska and F. Martínez-Álvarez (2019): Multi-step forecasting for big data time series based on ensemble learning. *Knowledge-Based Systems*, vol. 163, pp. 830–841.

[76] W. Gan, J. C.-W. Lin, H.-C. Chao and J. Zhan (2017): Data mining in distributed environment: a survey. *WIREs Data Mining and Knowledge Discovery*, vol. 7(6), p. e1216.

[77] Y. Gao and J. Lin (2019): HIME: discovering variable-length motifs in large-scale time series. *Knowl. Inf. Syst.*, vol. 61(1), pp. 513–542.

[78] Y. Gao, J. Lin and C. Brif (2020): Ensemble Grammar Induction For Detecting Anomalies in Time Series. In *Proc. of 23rd Intl. Conf. on Extending Database Technology, (EDBT)*, pp. 85–96. OpenProceedings.org.

[79] C. Gentry (2009): *A Fully Homomorphic Encryption Scheme*. Ph.D. thesis, Stanford University.

[80] Z. Gheid and Y. Challal (2016): Efficient and Privacy-Preserving k-Means Clustering for Big Data Mining. In *Trustcom/BigDataSE/ISPA*, pp. 791–798. IEEE.

[81] O. Goldreich (2002). Secure Multi-Party Computation. Final Draft, version 1.4, Available at http://www.wisdom.weizmann.ac.il/~oded/pp.html.

[82] O. Goldreich (2004): *Foundations of Cryptography: Volume 2 – Basic Applications*. Cambridge University Press.

[83] C. Gong, Z.-g. Su, P.-h. Wang and Y. You (2021): Distributed evidential clustering toward time series with big data issue. *Expert Systems with Applications*, p. 116279.

[84] M. Gong, Y. Xie, K. Pan, K. Feng and A. Qin (2020): A Survey on Differentially Private Machine Learning. *IEEE Computational Intelligence Magazine*, vol. 15(2), pp. 49–64.

[85] Y. Gong, Y. Fang and Y. Guo (2016): Private Data Analytics on Biomedical Sensing Data via Distributed Computation. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, vol. 13(3), pp. 431–444.

[86] C. Gonçalves, R. J. Bessa and P. Pinson (2021): A critical overview of privacy-preserving approaches for collaborative forecasting. *International Journal of Forecasting*, vol. 37(1), pp. 322–342.

[87] C. Gonçalves, R. J. Bessa and P. Pinson (2021): Privacy-Preserving Distributed Learning for Renewable Energy Forecasting. *IEEE Transactions on Sustainable Energy*, vol. 12(3), pp. 1777–1787.

[88] V. Goyal, H. Li, R. Ostrovsky, A. Polychroniadou and Y. Song (2021): ATLAS: efficient and scalable MPC in the honest majority setting. In *Intl. Cryptology Conference*, pp. 244–274. Springer.

[89] R. L. Grossman, S. M. Bailey, H. Sivakumar and A. L. Turinsky (1999): Papyrus: A System for Data Mining over Local and Wide-area Clusters and Super-clusters. In *Proc. of the Conf. on Supercomputing (SC)*, p. 63. ACM.

[90] I. Hagestedt, M. Humbert, P. Berrang, I. Lehmann, R. Eils, M. Backes and Y. Zhang (2020): Membership Inference Against DNA Methylation Databases. In *European Symp. on Security and Privacy (EuroS&P)*, pp. 509–520.

[91] J. Han (2019): *Scalable Approximate Inference and Some Applications*. Ph.D. thesis, Dartmouth College.

[92] J. Han and Q. Liu (2016): Bootstrap Model Aggregation for Distributed Statistical Learning. In *Advances in Neural Information Processing Systems 29*, pp. 1795–1803. Curran Associates, Inc.

[93] J. Han, J. Pei and M. Kamber (2011): *Data mining: concepts and techniques*. Elsevier.

[94] D. Hand, H. Mannila and P. Smyth (2001): *Principles of Data Mining*. The MIT Press.

[95] M. U. Hassan, M. H. Rehmani and J. Chen (2020): Differential Privacy Techniques for Cyber Physical Systems: A Survey. *IEEE Comm. Surveys & Tutorials*, vol. 22(1), pp. 746–789.

[96] T. Hastie, R. Tibshirani and J. Friedman (2001): *The Elements of Statistical Learning*. Springer Series in Statistics. Springer.

[97] M. Hastings, B. Hemenway, D. Noble and S. Zdancewic (2019): Sok: General Purpose Compilers for Secure Multi-Party Computation. In *Symp. on Security and Privacy (SP)*, pp. 1220–1237. IEEE.

[98] D. Hasuda and J. Bezerra (2021): Exploring Differential Privacy in Practice. In *Proc. of the 23rd Intl. Conf. on Enterprise Information Systems (ICEIS)*, pp. 877–884. SciTePress.

[99] Z. He, J. Yu, J. Li, Q. Han, G. Luo and Y. Li (2020): Inference Attacks and Controls on Genotypes and Phenotypes for Individual Genomic Data. *Trans. on Computational Biology and Bioinformatics*, vol. 17(3), pp. 930–937.

[100] S. Heidari, M. Alborzi, R. Radfar, M. A. Afsharkazemi and A. R. Ghatari (2019): Big data clustering with varied density based on MapReduce. *Journal of Big Data*, vol. 6(1), pp. 1–16.

[101] J. R. Higgins (1996): *Sampling Theory in Fourier and Signal Analysis*. Clarendon Press.

[102] A. Hinneburg and H.-H. Gabriel (2007): DENCLUE 2.0: fast clustering based on kernel density estimation. In *Proc. of the 7th Intl. Conf. on Intelligent Data Analysis (IDA)*, pp. 70–80. Springer.

[103] C. Huang, P. Kairouz, X. Chen, L. Sankar and R. Rajagopal (2017): Context-Aware Generative Adversarial Privacy. *Entropy*, vol. 19(12), p. 656.

[104] X. Huo and S. Cao (2019): Aggregated inference. *Wiley Interdisciplinary Reviews: Computational Statistics*, vol. 11(1), p. e1451.

[105] M. Imamura, T. Nakamura and E. Keogh (2020): Matrix Profile XXI: A Geometric Approach to Time Series Chains Improves Robustness. In *Proc. of 26th Intl. Conf.on Knowledge Discovery & Data Mining*, pp. 1114–1122.

[106] S. Imtiaz, S.-F. Horchidan, Z. Abbas, M. Arsalan, H. N. Chaudhry and V. Vlassov (2020): Privacy Preserving Time-Series Forecasting of User Health Data Streams. In *Intl. Conf on Big Data*, pp. 3428–3437.

[107] J. Jia and W. Qiu (2020): Research on an Ensemble Classification Algorithm Based on Differential Privacy. *IEEE Access*, vol. 8, pp. 93499–93513.

[108] M. Jia, Y. Wang, C. Shen and G. Hug (2021): Privacy-Preserving Distributed Clustering for Electrical Load Profiling. *IEEE Transactions on Smart Grid*, vol. 12(2), pp. 1429–1444.

[109] M. Kantarcioglu (2008): A Survey of Privacy-Preserving Methods Across Horizontally Partitioned Data. In *Privacy-Preserving Data Mining*, *The Kluwer Intl. Series on Advances in Database Systems*, vol. 34, pp. 313–335. Springer.

[110] H. Kargupta and P. Chan (Eds.) (2000): *Advances in Distributed and Parallel Knowledge Discovery*. AAAI Press / MIT Press.

[111] H. Kargupta, S. Datta, Q. Wang and K. Sivakumar (2003): On the Privacy Preserving Properties of Random Data Perturbation Techniques. In *Proc. of the 3rd Intl. Conf. on Data Mining (ICDM)*, pp. 99–106. IEEE.

[112] H. Kargupta, I. Hamzaoglu and B. Stafford (1997): Scalable, Distributed Data Mining: an Agent Architecture. In *Proc. of the 3rd Intl. Conf. on Knowledge Discovery and Data Mining (KDD)*, pp. 211–214. AAAI Press.

[113] H. Kargupta, B.-H. Park, D. Hershberger and E. Johnson (2000): Collective Data Mining: A New Perspective Toward Distributed Data Mining. In *Advances in Distributed and Parallel Knowledge Discovery*, chap. 5, pp. 131–174. AAAI Press / MIT Press.

[114] E. J. Keogh, K. Chakrabarti, M. J. Pazzani and S. Mehrotra (2000): Dimensionality Reduction for Fast Similarity Search in Large Time Series Databases. *Knowledge and Information Systems*, vol. 3(3), pp. 263–286.

[115] L. U. Khan, I. Yaqoob, N. H. Tran, S. M. A. Kazmi, T. N. Dang and C. S. Hong (2020): Edge-Computing-Enabled Smart Cities: A Comprehensive Survey. *Internet of Things Journal*, vol. 7(10), pp. 10200–10232.

[116] D. Kifer and A. Machanavajjhala (2012): A Rigorous and Customizable Framework for Privacy. In *Proc. of the 31st Symp. on Principles of Database Systems (PODS)*, pp. 77–88. ACM.

[117] D. Kifer and A. Machanavajjhala (2014): Pufferfish: A Framework for Mathematical Privacy Definitions. *ACM Transactions on Database Systems*, vol. 39(1), pp. 3:1–3:36.

[118] M. Klusch, S. Lodi and G. Moro (2003): Agent-based Distributed Data Mining: the KDEC Scheme. In *Intelligent Information Agents: the AgentLink perspective*, *LNCS*, vol. 2586. Springer.

[119] S. B. Kotsiantis (2013): Decision trees: a recent overview. *Artificial Intelligence Review*, vol. 39(4), pp. 261–283.

[120] N. Labroche, N. Monmarché and G. Venturini (2002): A New Clustering Algorithm Based on the Chemical Recognition System of Ants. In *Proc. of the 15th European Conf. on Artificial Intelligence (ECAI)*, pp. 345–349. IOS Press.

[121] K. Li, G. Luo, Y. Ye, W. Li, S. Ji and Z. Cai (2021): Adversarial Privacy-Preserving Graph Embedding Against Inference Attack. *IEEE Internet of Things Journal*, vol. 8(8), pp. 6904–6915.

[122] T. Li, A. K. Sahu, A. Talwalkar and V. Smith (2020): Federated Learning: Challenges, Methods, and Future Directions. *IEEE Signal Processing Magazine*, vol. 37(3), pp. 50–60.

[123] Y. Li, C. Bai and C. K. Reddy (2016): A Distributed Ensemble Approach for Mining Healthcare Data under Privacy Constraints. *Information Science*, vol. 330, pp. 245–259.

[124] Y. Li, Z. Jiang, Y. Lin, X. Wang, S. Yiu and Z. Huang (2019): Outsourced privacy-preserving C4.5 decision tree algorithm over horizontally and vertically partitioned dataset among multiple parties. *Cluster Computing*, vol. 22, pp. 1581–1593.

[125] Z. Li, L. Yang and Z. Li (2020): Mixture-Model-Based Graph for Privacy-Preserving Semi-Supervised Learning. *IEEE Access*, vol. 8, pp. 789–801.

[126] M. Liang, Q. Li, Y. Geng, J. Wang and Z. Wei (2017): REMOLD: An Efficient Model-Based Clustering Algorithm for Large Datasets with Spark. In *Proc. of the 23rd Intl. Conf. on Parallel and Distributed Systems (ICPADS)*, pp. 376–383. IEEE.

[127] X. Limón, A. Guerra-Hernández, N. Cruz-Ramírez and F. Grimaldo (2019): Modeling and implementing distributed data mining strategies in JaCa-DDM. *Knowledge and Information Systems*, vol. 60(1), pp. 99–143.

[128] J. Lin, E. Keogh, S. Lonardi and P. Patel (2002): Finding Motifs in Time Series. In *Proc. of the 2nd Workshop on Temporal Data Mining at the 8th Intl. Conf. on Knowledge Discovery and Data Mining (KDD)*. ACM.

[129] X. Lin, C. Clifton and M. Zhu (2005): Privacy-preserving clustering with Distributed EM Mixture Modeling. *Knowledge and Information Systems*, vol. 8(1), pp. 68–81.

[130] Y. Lindell and B. Pinkas (2000): Privacy Preserving Data Mining. In *Proc. of the 20th Intl. Conf. on Advances in Cryptology (CRYPTO)*, pp. 36–54. Springer.

[131] Y. Lindell and B. Pinkas (2002): Privacy Preserving Data Mining. *Journal of Cryptology*, vol. 15(3), pp. 177–206.

[132] Y. Lindell and B. Pinkas (2009): Secure Multiparty Computation for Privacy-Preserving Data Mining. *Journal of Privacy and Confidentiality*, vol. 1(1). Article 5.

[133] Y. Lindell, B. Pinkas, N. P. Smart and A. Yanai (2019): Efficient Constant-Round Multi-party Computation Combining BMR and SPDZ. *Journal of cryptology*, vol. 32(3), pp. 1026–1069.

[134] J. Liu, J. Z. Huang, J. Luo and L. Xiong (2012): Privacy Preserving Distributed DBSCAN Clustering. In *Proc. of the Joint EDBT/ICDT Workshops*, pp. 177–185. ACM.

[135] J. Liu, L. Xiong, J. Luo and J. Z. Huang (2013): Privacy Preserving Distributed DBSCAN Clustering. *Transactions on Data Privacy*, vol. 6(1), pp. 69–85.

[136] Q. Liu and A. T. Ihler (2014): Distributed estimation, information loss and exponential families. In *Advances in Neural Information Processing Systems*, pp. 1098–1106.

[137] X. Liu and H. Ma (2019): Privacy Preserving Finite-time Consensus in Networks With Time-varying Topology. In *Proc. of 34rd Conf. of Chinese Association of Automation (YAC)*, pp. 312–316. IEEE.

[138] X. Liu, Y. Zheng, X. Yi and S. Nepal (2020): Privacy-Preserving Collaborative Analytics on Medical Time Series Data. *IEEE Transactions on Dependable and Secure Computing*, pp. 1–16.

[139] Y. Liu, J. James, J. Kang, D. Niyato and S. Zhang (2020): Privacy-preserving Traffic Flow Prediction: A Federated Learning Approach. *IEEE Internet of Things Journal*, vol. 7(8), pp. 7751–7763.

[140] S. Lodi, G. Moro and C. Sartori (2010): Distributed data clustering in multi-dimensional peer-to-peer networks. In *Proc. of the 21st Australasian Conf. on Database Technologies*, vol. 104, pp. 171–178. Australian Computer Society, Inc.

[141] J. M. Luna, P. Fournier-Viger and S. Ventura (2019): Frequent itemset mining: A 25 years review. *WIREs Data Mining and Knowledge Discovery*, vol. 9(6), p. e1329.

[142] J. Luo, M. Wu, D. Gopukumar and Y. Zhao (2016): Big Data Application in Biomedical Research and Health Care: A Literature Review. *Biomedical Informatics Insights*, vol. 8, pp. 1–10.

[143] L. Lyu, Y. W. Law, S. M. Erfani, C. Leckie and M. Palaniswami (2016): An Improved Acheme for Privacy-preserving Collaborative Anomaly Detection. In *Proc. of Intl. Conf. on Pervasive Computing and Communication Workshops*, (PerCom), pp. 1–6. IEEE.

[144] S. Mehnaz, N. Li and E. Bertino (2020): Black-box model inversion attribute inference attacks on classification models. *arXiv preprint arXiv:2012.03404*.

[145] L. Mehner, S. N. v. Voigt and F. Tschorsch (2021): Towards Explaining Epsilon: A Worst-Case Study of Differential Privacy Risks. In *Eur. Symp. on Security and Privacy Works. (EuroS PW)*, pp. 328–331.

[146] D. Melnyk (2020): *Byzantine Agreement on Representative Input Values Over Public Channels*. Ph.D. thesis, ETH Zurich.

[147] R. Mendes and J. P. Vilela (2017): Privacy-Preserving Data Mining: Methods, Metrics, and Applications. *IEEE Access*, vol. 5, pp. 10562–10582.

[148] A. J. Menezes, P. C. Van Oorschot and S. A. Vanstone (2018): *Handbook of Applied Cryptography*. CRC Press.

[149] S. Merugu and J. Ghosh (2003): Privacy-preserving Distributed Clustering using Generative Models. In *Proc. of the 3rd Intl. Conf. on Data Mining (ICDM)*, pp. 211–218. IEEE.

[150] S. Merugu and J. Ghosh (2005): A privacy-sensitive approach to distributed clustering. *Pattern Recognition Letters*, vol. 26, pp. 399–410.

[151] D. Minnen, C. Isbell, I. Essa and T. Starner (2007): Discovering multivariate motifs using subsequence density estimation and greedy mixture learning. In *Proc. of the 22nd Conf. on Artificial Intelligence*, p. 615–620. AAAI.

[152] M. Morgenstern (1987): Security and Inference in Multilevel Database and Knowledge-base Systems. In *Proc. of the Int. Conf. on Management of Data (SIGMOD)*, pp. 357–373. ACM.

[153] A. Mueen, E. J. Keogh, Q. Zhu, S. Cash and M. B. Westover (2009): Exact Discovery of Time Series Motifs. In *Proc. of the Intl. Conf. on Data Mining, SDM*, pp. 473–484. SIAM.

[154] M. Naldi and G. D'Acquisto (2015): Differential privacy: An estimation theory-based method for choosing epsilon. *arXiv preprint arXiv:1510.00917*.

[155] N. N. Neto, S. Madnick, A. M. G. D. Paula and N. M. Borges (2021): Developing a Global Data Breach Database and the Challenges Encountered. *Data and Information Quality*, vol. 13(1).

[156] P. Paillier (1999): Public-Key Cryptosystems Based on Composite Degree Residuosity Classes. In *Proc. of the Intl. Conf. on Theory and Application of Cryptographic Techniques (EUROCRYPT)*, pp. 223–238. Springer.

[157] Y. Pan, A. Efrat, M. Li, B. Wang, H. Quan, J. Mitchell, J. Gao and E. Arkin (2020): Data Inference from Encrypted Databases: a multi-dimensional order-preserving matching approach. In *Proc. of 21st Intl. Symp. on Theory, Algorithmic Foundations, and Protocol Design for Mobile Networks and Mobile Computing*, pp. 151–160.

[158] J. Paparrizos and L. Gravano (2017): Fast and Accurate Time-Series Clustering. *ACM Trans. Database Syst.*, vol. 42(2).

[159] E. Parzen (1962): On Estimation of a Probability Density Function and Mode. *Ann. Math. Statist.*, vol. 33(3), pp. 1065–1076.

[160] S. Patel, S. Garasia and D. Jinwala (2012): An Efficient Approach for Privacy Preserving Distributed K-Means Clustering Based on Shamir's Secret Sharing Scheme. In *Proc. of 6th Intl. Conf. on Trust Management (TM)*, pp. 129–141. Springer.

[161] S. J. Patel, D. Punjani and D. C. Jinwala (2015): An Efficient Approach for Privacy Preserving Distributed Clustering in Semi-honest Model Using Elliptic Curve Cryptography. *Intl. Journal of Network Security*, vol. 17(3), pp. 328–339.

[162] Y. Perez-Riverol, J. Bai, C. Bandla, D. García-Seisdedos, S. Hewapathirana, S. Kamatchinathan, D. Kundu, A. Prakash, A. Frericks-Zipper, M. Eisenacher, M. Walzer, S. Wang, A. Brazma and J. Vizcaíno (2021): The PRIDE database resources in 2022: a hub for mass spectrometry-based proteomics evidences. *Nucleic Acids Research*.

[163] E. Pernot-Leplay (2020): China's Approach on Data Privacy Law: A Third Way Between the US and the EU? *Penn State Journal of Law & International Affairs*, vol. 8(1).

[164] F. Petropoulos, D. Apiletti, V. Assimakopoulos, M. Z. Babai, D. K. Barrow, S. B. Taieb, C. Bergmeir, R. J. Bessa, J. Bijak, J. E. Boylan et al. (2020): Forecasting: theory and practice. *arXiv preprint arXiv:2012.03854*.

[165] B. Pinkas (2002): Cryptographic Techniques for Privacy-preserving Data Mining. *ACM SIGKDD Explorations Newsletter*, vol. 4(2), pp. 12–19.

[166] G. N. Pradhan and B. Prabhakaran (2017): Association rule mining in multiple, multidimensional time series medical data. *Journal of Healthcare Informatics Research*, vol. 1(1), pp. 92–118.

[167] V. Primault, A. Boutet, S. B. Mokhtar and L. Brunie (2019): The Long Road to Computational Location Privacy: A Survey. *IEEE Communications Surveys & Tutorials*, vol. 21, pp. 2772–2793.

[168] A. L. Prodromidis and P. K. Chan (2000): Meta-Learning in Distributed Data Mining Systems: issues and approaches. In *Advances in Distributed and Parallel Knowledge Discovery*. AAAI Press / MIT Press.

[169] C. Qiao and K. N. Brown (2019): Asynchronous Distributed Clustering Algorithm for Wireless Sensor Networks. In *Proc. of the 4th Intl. Conf. on Machine Learning Technologies*, ICMLT, p. 76–82. ACM.

[170] J. Qin, Y. Zhu and W. Fu (2020): Distributed Clustering Algorithm in Sensor Networks via Normalized Information Measures. *IEEE Transactions on Signal Processing*, vol. 68, pp. 3266–3279.

[171] S. Rajagopalan, L. Sankar, S. Mohajer and H. Poor (2011): Smart Meter Privacy: A utility-privacy framework. In *Proc. of Intl. Conf. on Smart Grid Communications (SmartGridComm)*, pp. 190–195. IEEE.

[172] V. Rastogi and S. Nath (2010): Differentially Private Aggregation of Distributed Time-Series with Transformation and Encryption. In *Proc. of the Intl. Conf. on Management of Data (SIGMOD)*, pp. 735–746. ACM.

[173] V. H. A. Ribeiro and G. Reynoso-Meza (2020): Ensemble learning by means of a multi-objective optimization design approach for dealing with imbalanced data sets. *Expert Systems with Applications*, vol. 147, p. 113232.

[174] R. J. Roiger (2017): *Data mining: a tutorial-based primer*. Chapman and Hall/CRC.

[175] A. Rosato, R. Altilio and M. Panella (2021): A decentralized algorithm for distributed ensemble clustering. *Inf. Sci.*, vol. 578, pp. 417–434.

[176] J. D. Rosenblatt and B. Nadler (2016): On the optimality of averaging in distributed statistical learning. *Information and Inference: A Journal of the IMA*, vol. 5(4), pp. 379–404.

[177] J. N. S. Rubí and P. R. d. L. Gondim (2020): Interoperable Internet of Medical Things platform for e-Health applications. *Intl. Journal of Distributed Sensor Networks*, vol. 16(1), p. 1550147719889591.

[178] S. J. Russell and P. Norvig (1995): *Artificial Intelligence: A Modern Approach*. Prentice-Hall.

[179] O. Sagi and L. Rokach (2018): Ensemble Learning: A survey. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, vol. 8(4), p. e1249.

[180] G. L. Santos, K. H. d. C. Monteiro and P. T. Endo (2020): Living at the Edge? Optimizing availability in IoT. In *The Cloud-to-Thing Continuum*, pp. 79–94. Palgrave Macmillan, Cham.

[181] S. Scardapane, R. Altilio, M. Panella and A. Uncini (2016): Distributed spectral clustering based on Euclidean distance matrix completion. In *Intl. Joint Conf. on Neural Networks (IJCNN)*, pp. 3093–3100.

[182] F. Schaub (2018): Context-adaptive privacy mechanisms. In *Handbook of Mobile Data Privacy*, pp. 337–372. Springer.

[183] P. Senin, J. Lin, X. Wang, T. Oates, S. Gandhi, A. Boedihardjo, C. Chen, S. Frankenstein and M. Lerner (2015): Time series anomaly discovery with grammar-based compression. In *Proc. of the Intl. Conf. on Extending Database Technology (EDBT)*, pp. 481–492.

[184] P. Shen and C. Li (2014): Distributed Information Theoretic Clustering. *IEEE Transactions on Signal Processing*, vol. 62(13), pp. 3442–3453.

[185] A. Shewale, B. N. Keshavamurthy and C. N. Modi (2019): An Efficient Approach for Privacy Preserving Distributed K-Means Clustering in Unsecured Environment. In *Recent Findings in Intelligent Computing Techniques*, pp. 425–431. Springer.

[186] E. Shi, T.-H. H. Chan, E. G. Rieffel, R. Chow and D. Song (2011): Privacy-Preserving Aggregation of Time-Series Data. In *Proc. of 18th Network and Distributed System Security Symposium (NDSS)*. The Internet Society.

[187] R. Shokri, M. Stronati, C. Song and V. Shmatikov (2017): Membership Inference Attacks Against Machine Learning Models. In *Symp. on Security and Privacy (SP)*, pp. 3–18. IEEE.

[188] B. W. Silverman (1986): *Density Estimation for Statistics and Data Analysis*. Chapman and Hall.

[189] L. Singh and M. Sayal (2007): Privacy Preserving Burst Detection of Distributed Time Series Data Using Linear Transforms. In *Proc. of the Symp. on Computational Intelligence and Data Mining (CIDM)*, pp. 646–653. IEEE.

[190] P. Smyth and D. Wolpert (1999): Linearly combining density estimators via stacking. *Machine Learning*, vol. 36(1-2), pp. 59–83.

[191] D. Spies, P. F. Renz, T. A. Beyer and C. Ciaudo (2017): Comparative analysis of differential gene expression tools for RNA sequencing time course data. *Briefings in Bioinformatics*, vol. 20(1), pp. 288–298.

[192] S. J. Stolfo, A. L. Prodromidis, S. Tselepis, W. Lee, D. W. Fan and P. K. Chan (1997): JAM: Java Agents for Meta-Learning over Distributed Databases. In *Proc. of the 3rd Conf. on Knowledge Discovery and Data Mining (KDD)*, pp. 74–81. AAAI Press.

[193] L. Sun, S. Bao, S. Ci, X. Zheng, L. Guo and Y. Luo (2019): Differential Privacy-Preserving Density Peaks Clustering Based on Shared Near Neighbors Similarity. *IEEE Access*, vol. 7, pp. 89427–89440.

[194] L. Sweeney (2002): k-Anonymity: a model for protecting Privacy Protection Using Generalization and Suppression. *Journal on Uncertainty, Fuzziness and Knowledge-based Systems*, vol. 10(5), pp. 557–570.

[195] R. Talavera-Llames, R. Pérez-Chacón, A. Troncoso and F. Martínez-Álvarez (2018): Big data time series forecasting based on nearest neighbours distributed computing with Spark. *Knowledge-Based Systems*, vol. 161, pp. 12–25.

[196] J. Tang, A. Korolova, X. Bai, X. Wang and X. Wang (2017): Privacy Loss in Apple's Implementation of Differential Privacy on MacOS 10.12. *arXiv: CoRR*, vol. abs/1709.02753.

[197] S. G. Teo, J. Cao and V. C. S. Lee (2015): DAG: A Model for Privacy Preserving Computation. In *Intl. Conf. on Web Services*, pp. 289–296.

[198] S. G. Teo, J. Cao and V. C. S. Lee (2020): DAG: A General Model for Privacy-Preserving Data Mining. *IEEE Transactions on Knowledge and Data Engineering*, vol. 32(1), pp. 40–53.

[199] The United Nations (1948): *Universal Declaration of Human Rights*. UN General Assembly.

[200] B. Thuraisingham (2002): Data Mining, National Security, Privacy and Civil Liberties. *ACM SIGKDD Explorations Newsletter*, vol. 4(2), pp. 1–5.

[201] B. Thuraisingham (2020): Multi-Generational Database Inference Controllers. In *6th Intl. Conf. on Big Data Security on Cloud (BigDataSecurity)*, pp. 8–12.

[202] T. S. Toland, C. Farkas and C. M. Eastman (2010): The Inference Problem: Maintaining Maximal Availability in the Presence of Database Updates. *Computers & Security*, vol. 29(1), pp. 88–103.

[203] Q. Tong, X. Li and B. Yuan (2018): Efficient distributed clustering using boundary information. *Neurocomputing*, vol. 275, pp. 2355–2366.

[204] M. Y. Topaloglu, E. M. Morrell, S. Rajendran et al. (2021): In the Pursuit of Privacy: The Promises and Predicaments of Federated Learning in Healthcare. *Frontiers in Artificial Intelligence*, vol. 4, p. 147.

[205] S. Torkamani and V. Lohweg (2017): Survey on Time series Motif Discovery. *WIREs Data Mining and Knowledge Discovery*, vol. 7(2), p. e1199.

[206] H.-Y. Tran and J. Hu (2019): Privacy-preserving big data analytics a comprehensive survey. *Journal of Parallel and Distributed Computing*, vol. 134, pp. 207 – 218.

[207] A. Triastcyn and B. Faltings (2020): Federated Generative Privacy. *IEEE Intelligent Systems*, vol. 35(4), pp. 50–57.

[208] S. Truex, N. Baracaldo, A. Anwar, T. Steinke, H. Ludwig, R. Zhang and Y. Zhou (2019): A Hybrid Approach to Privacy-Preserving Federated Learning. In *Proc. of 12th Works. on Artificial Intelligence and Security*, AISec'19, p. 1–11. ACM, New York, NY, USA.

[209] C. W. Tsai, C. F. Lai, M. C. Chiang and L. T. Yang (2014): Data Mining for Internet of Things: A Survey. *IEEE Communications Surveys Tutorials*, vol. 16(1), pp. 77–97.

[210] N. Tsapanos, A. Tefas, N. Nikolaidis and I. Pitas (2015): A Distributed Framework for Trimmed Kernel k-Means Clustering. *Pattern Recognition*, vol. 48(8), pp. 2685–2698.

[211] Y.-T. Tsou, H.-L. Chen and J.-Y. Chen (2021): RoD: Evaluating the Risk of Data Disclosure Using Noise Estimation for Differential Privacy. *IEEE Transactions on Big Data*, vol. 7(1), pp. 214–226.

[212] A. Tuladhar, S. Gill, Z. Ismail and N. D. Forkert (2020): Building Machine Learning Models Without Sharing Patient Data: A simulation-based analysis of distributed learning by ensembling. *Journal of Biomedical Informatics*, vol. 106, p. 103424.

[213] J. Vaidya and C. Clifton (2003): Privacy-Preserving k-Means Clustering Over Vertically Partitioned Data. In *Proc. of the 9th Intl. Conf. on Knowledge Discovery and Data Mining (KDD)*, pp. 206–215. ACM.

[214] J. Verbraeken, M. Wolting, J. Katzy, J. Kloppenburg, T. Verbelen and J. S. Rellermeyer (2020): A survey on distributed machine learning. *Computing Surveys (CSUR)*, vol. 53(2), pp. 1–33.

[215] V. Verykios, A. Elmagarmid, E. Bertino, Y. Saygin and E. Dasseni (2004): Association rule hiding. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, vol. 16(4), pp. 434 – 447.

[216] I. Wagner and D. Eckhoff (2018): Technical privacy metrics: a systematic survey. *ACM Computing Surveys (CSUR)*, vol. 51(3), pp. 1–38.

[217] H. Wang, A. Li, B. Shen, Y. Sun and H. Wang (2020): Federated Multi-View Spectral Clustering. *IEEE Access*, vol. 8, pp. 202249–202259.

[218] H. Wang and Z. Xu (2017): CTS-DP: publishing correlated time-series data via differential privacy. *Knowledge-Based Systems*, vol. 122, pp. 167–179.

[219] H. Wang, Z. Xu, S. Jia, Y. Xia and X. Zhang (2021): Why current differential privacy schemes are inapplicable for correlated data publishing? *World Wide Web*, vol. 24, pp. 1–23.

[220] J.-P. Wang, Y.-C. Lu, M.-Y. Yeh, S.-D. Lin and P. Gibbons (2013): Communication-Efficient Distributed Multiple Reference Pattern Matching for M2M Systems. In *Proc. of the 13th Intl. Conf. on Data Mining (ICDM)*, pp. 787–796. IEEE.

[221] T. Wang, X. Zhang, J. Feng and X. Yang (2020): A Comprehensive Survey on Local Differential Privacy toward Data Statistics and Analysis. *Sensors*, vol. 20(24).

[222] Y. Wang, Q. Chen, T. Hong and C. Kang (2018): Review of smart meter data analytics: Applications, methodologies, and challenges. *IEEE Transactions on Smart Grid*, vol. 10(3), pp. 3125–3148.

[223] Y. Wang, Q. Chen, T. Hong and C. Kang (2019): Review of Smart Meter Data Analytics: Applications, Methodologies, and Challenges. *IEEE Transactions on Smart Grid*, vol. 10(3), pp. 3125–3148.

[224] Z. Wang, W. Liu, X. Pang, J. Ren, Z. Liu and Y. Chen (2020): Towards Pattern-aware Privacy-preserving Real-time Data Collection. In *Conf. on Computer Communications*, pp. 109–118.

[225] J. Waring, C. Lindvall and R. Umeton (2020): Automated machine learning: Review of the state-of-the-art and opportunities for healthcare. *Artificial Intelligence in Medicine*, vol. 104, p. 101822.

[226] C. Woodward (2011): Data-Mining Case Tests Boundaries of Medical Privacy. *Canadian Medical Association Journal*, vol. 183(9).

[227] C. Xia, J. Hua, W. Tong and S. Zhong (2020): Distributed K-Means Clustering Guaranteeing Local Differential Privacy. *Computers & Security*, vol. 90, p. 101699.

[228] T. Xiang, Y. Li, X. Li, S. Zhong and S. Yu (2018): Collaborative ensemble learning under differential privacy. *Web Intelligence*, vol. 16(1), pp. 73–87.

[229] R. Xu, N. Baracaldo, Y. Zhou, A. Anwar and H. Ludwig (2019): HybridAlpha: An Efficient Approach for Privacy-Preserving Federated Learning. In *Proc. of the 12th Work. on Artificial Intelligence and Security (AISec)*, p. 13–23. ACM.

[230] Z. Xu and X. Yi (2011): Classification of Privacy-preserving Distributed Data Mining protocols. In *Proc. of the 6th Intl. Conf. on Digital Information Management (ICDIM)*, pp. 337–342. IEEE.

[231] D. Yagoubi, R. Akbarinia, F. Masseglia and T. Palpanas (2020): Massively Distributed Time Series Indexing and Querying. *IEEE Transactions on Knowledge and Data Engineering*, vol. 32(1), pp. 108–120.

[232] Q. Yang, Y. Liu, T. Chen and Y. Tong (2019): Federated Machine Learning: Concept and Applications. *ACM Transactions on Intelligent Systems Technology*, vol. 10(2).

[233] A. C.-C. Yao (1982): Protocols for Secure Computations (Extended Abstract). In *Proc. of the 23rd Annual Symp. on Foundations of Computer Science (FOCS)*, pp. 160–164. IEEE.

[234] A. C.-C. Yao (1986): How to Generate and Exchange Secrets. In *Proc. of the 27th Annual Symp. on Foundations of Computer Science (FOCS)*, pp. 162 –167. IEEE.

[235] C.-C. M. Yeh, Y. Zhu, L. Ulanova, N. Begum, Y. Ding, H. A. Dau, Z. Zimmerman, D. F. Silva, A. Mueen and E. Keogh (2018): Time series joins, motifs, discords and shapelets: a unifying view that exploits the matrix profile. *Data Mining and Knowledge Discovery*, vol. 32(1), pp. 83–123.

[236] S. Yu (2016): Big Privacy: Challenges and Opportunities of Privacy Study in the Age of Big Data. *IEEE Access*, vol. 4, pp. 2751–2763.

[237] Y. Yu, J. Zhao, X. Wang, Q. Wang and Y. Zhang (2015): Cludoop: An Efficient Distributed Density-Based Clustering for Big Data Using

Hadoop. *Hindawi Intl. Journal of Distributed Sensor Networks*, vol. 501, p. 579391.

[238] M. J. Zaki and Y. Pan (2002): Introduction: Recent Developments in Parallel and Distributed Data Mining. *Distributed and Parallel Databases*, vol. 11(2), pp. 123–127.

[239] X. Zhang, W. K. Cheung and C. Li (2011): Learning latent variable models from distributed and abstracted data. *Information Sciences*, vol. 181(14), pp. 2964 – 2988.

[240] X. Zhang, W. K. Cheung and Y. Ye (2016): Mining from distributed and abstracted data. *Data Mining and Knowledge Discovery*, vol. 6(5), pp. 167–176.

[241] Y. Zhang, J. C. Duchi and M. J. Wainwright (2013): Communication-efficient algorithms for statistical optimization. *The Journal of Machine Learning Research*, vol. 14(1), pp. 3321–3363.

[242] Z. Zhang, C. Zhang and S. Zhang (2003): An Agent-Based Hybrid Framework for Database Mining. *Applied Artificial Intelligence*, vol. 17(5-6), pp. 383–398.

[243] L. Zhao, L. Ni, S. Hu, Y. Chen, P. Zhou, F. Xiao and L. Wu (2018): InPrivate Digging: Enabling Tree-based Distributed Data Mining with Differential Privacy. In *Conf. on Computer Communications (INFOCOM)*, pp. 2087–2095.

[244] Y. Zhao and I. Wagner (2018): On the strength of privacy metrics for vehicular communication. *IEEE Transactions on Mobile Computing*, vol. 18(2), pp. 390–403.

[245] Y. Zhu, Y. Fu and H. Fu (2008): On Privacy in Time Series Data Mining. In *Proc. of the 12th Pacific-Asia Conf. on Advances in Knowledge Discovery and Data Mining (PAKDD)*, pp. 479–493. Springer.

[246] Y. Zhu, C.-C. M. Yeh, Z. Zimmerman, K. Kamgar and E. Keogh (2018): Matrix profile XI: SCRIMP++: time series motif discovery at interactive speeds. In *2018 IEEE International Conference on Data Mining (ICDM)*, pp. 837–846. IEEE.