UNIVERSIDADE DE LISBOA
INSTITUTO SUPERIOR TÉCNICO

Big Data Privacy by Design Computation Platform

Rui Nuno Lopes Claro

Supervisor: Doctor Miguel Filipe Leitão Pardal
Co-Supervisor: Doctor José Miguel Ladeira Portêlo

Thesis to obtain the Masters Degree in
Information Systems and Computer Engineering

2018

Resumo

Abstract

We live in the age of Big Data. Personal user data, in particular, is necessary for the operation and improvement of everyday Internet services like Google, Facebook, WhatsApp, Spotify, etc. Many times, the capture and use of personal data is not made explicit to the users, but it is central to the business model of companies. However, each person's right to privacy has to be respected. How can these two conflicting needs be reconciled, *i.e.*, how can we build useful Big Data systems that are respectful of user privacy? The goal of this work is to design and implement a proof-of-concept of a platform for performing privacy-preserving computations, providing an easy-to-use method to implement privacy-preserving techniques. This system could be used, for example, to monitor the vital signs of patients (without exposing them to other people), to produce real time recommendations based on location (without disclosing location to others), sports/fitness applications, etc. This proof-of-concept will implement privacy-preserving versions of Machine Learning algorithms and compare them against a baseline reference, allowing a better understanding of the trade-offs of using this technology.

so the trade-offs can be quantified and better understood.

Palavras-Chave
Keywords

Palavras-Chave

Preservação de privacidade em Computações
Aprendizagem automática
Extracção de Informação
Big Data
Processamento de Dados
Computação Multi-Entidade Segura

Keywords

Privacy-preserving Computations
Machine Learning
Data Mining
Big Data
Data Processing
Secure Multi-Party Computation

Acknowledgements

Contents

List of Figures

List of Tables

x

List of Acronyms

BARD Big dAta pRivacy by Design platform

CMU Carnegie Mellon University

DP Differential Privacy

DT Decision Trees

EMR Eletronic Medical Records

ENISA European Union Agency for Network and Information Security

EU European Union

FE Functional Encryption

FHE Fully Homomorphic Encryption

GC Garbled Circuits

GDPR General Data Protection Regulation

HE Homomorphic Encryption

k-M k-Means

k-NN k-Nearest Neighbors

LR Logistic Regression

ML Machine Learning

OT Oblivious Transfer

PHE Partial Homomorphic Encryption

PII Personal Identifiable Information

PPDM Privacy-Preserving Data Mining

PPML Privacy-Preserving Machine Learning

RBF Radial Basis Function

SMPC Secure Multi-Party Computation

SPI Sensitive Personal Information

STPC Secure Two-Party Computation

SVM Support Vector Machines

1
Introduction

With the so called "Big Data revolution", vast amounts of data are now being analyzed and processed by companies that take advantage of the enormous quantities of information that is generated every day[1]. Big Data and Business Analytics market reflects this growth rate, expecting to hit the $210 billion mark in the year 2020[2]. Through this data processing, meaningful information can be obtained to improve existing systems or to discover new approaches in business models. An example of this is the deployment of *Data Mining* algorithms to better understand their customers, and to devise better recommendation systems, in order to surpass their competitors in customer satisfaction. Another example lies in the field of healthcare, where it can be beneficial to match patient records from different hospitals in order to identify inefficiencies and develop best practices [30].

Most times data contains private information about individuals, such as health records or daily routines. This kind of data cannot be freely processed because that leads to breaches of private information, such as the AOL Search Leak[3] or the Microsoft Hotmail privacy breach[4]. Due to these breaches, and despite the value that Data Mining adds to businesses and medical systems, consumers show an increasing concern in the privacy threats posed by it [7]. The privacy of an individual may be violated due to, for example, unauthorized access to personal data, or the use of personal data for purposes other than the one for which data was collected.

To deal with the privacy issues in Data Mining, a sub-field known as Privacy-Preserving Data Mining (PPDM) has been gaining influence over the last years [10]. The objective of PPDM

---

[1]http://www.vcloudnews.com/every-day-big-data-statistics-2-5-quintillion-bytes-of-data-created-daily/
[2]https://www.idc.com/getdoc.jsp?containerId=prUS42371417
[3]https://www.networkworld.com/article/2185187/security/15-worst-internet-privacy-scandals-of-all-time.html
[4]https://www.networkworld.com/article/2185187/security/15-worst-internet-privacy-scandals-of-all-time.html

is to guarantee the privacy of sensitive information, while at the same time preserve the utility of the data for Data Mining purposes [1]. This can be achieved by using one or more privacy-preserving techniques, such as Differential Privacy (DP) [11] or Secure Multi-Party Computation (SMPC) [10].

Machine Learning (ML) algorithms in the context of Big Data processing are also producing significant results, so that it is possible to do knowledge learning from datasets in order to predict future labels (*i.e.* classes of data) or clusters (groups of related data) for new data. An example of an application of ML algorithms in Data Mining is *Classification* [28], in which a training set is processed in order to create a classifier for data, and then that classifier is used to predict class labels for new data. These applications show a greater impact in the field of medicine as mentioned above, with Google's DeepMind building ML algorithms to process admissions in hospitals faster[5], and with IBM's Watson supporting medical personnel consider treatment options for their patients[6]. Some examples of ML algorithms include Decision Trees (DT), k-Nearest Neighbors (k-NN), and Support Vector Machines (SVM) [28].

By combining ML algorithms and privacy-preserving techniques, it is possible to create Data Mining processes that, not only allow for knowledge learning on large datasets but also to maintain a level of privacy that is desirable by individuals and that complies with the laws in force [10].

1.1   Contributions

The main contribution of this thesis is the design and creation of a proof-of-concept platform for privacy-preserving distributed ML computations without resorting to third parties. With it, we aim to give users a ML platform without them having to build their own, meaning less maintenance costs derived of maintaining code developed or dedicated servers. Since the platform has its foundations on privacy-preserving techniques, it addresses satisfactorily the

---

[5]https://deepmind.com/applied/deepmind-health/
[6]https://www.mskcc.org/about/innovative-collaborations/watson-oncology

privacy demands that those users want for their data.

We show in

> ref to use case

possible usages of this platform, detailing the context of the data to process, the trade-offs that are required, etc.

> Expand
> Contri-
> butions

1.2 Structure of this Document

The remainder of this thesis is structured as follows.

Related Work 2

This chapter provides and overview on the Privacy-Preserving Machine Learning (PPML) paradigm. We start by defining Data Security and Data Privacy, in sections 2.1 and 2.2 respectively. We detail each concept so that it gives an understanding of the differences between them. Section 2.3 presents the concepts of data processing and Data Mining, gives an overview of the CRISP-DM model, and defines the attack models that can be assumed when developing a Privacy-Preserving Data Mining (PPDM) algorithm. For developing a PPDM algorithm, one can implement one or more of the privacy-preserving techniques briefly presented in section 2.4. We present Machine Learning (ML) applied in Data Mining in section 2.5. Finally, we discuss known use cases that show what can be achieved in the field in section 2.6.

2.1 Data Security

Data Security refers to protective digital measures that are applied to prevent unauthorized access to computers, databases, and websites, as well as to prevent data destruction or alteration.

2.1.1 Data Security Principles

We define here the core security principles widely accepted in the literature, often known as the CIA triad (*Confidentiality*, *Integrity*, and *Availability*) [25].

- *Confidentiality* is defined as the property that data, and services that process such data, cannot be accessed by unauthorized entities.

- *Integrity* is defined as the property that data, and services that process such data, cannot be modified in an unauthorized or undetected manner.

- *Availability* is defined as the property that access to data, and services that process such data, is always possible when needed by the authorized parties and in a timely manner.

For applying Data Security measures, various technologies can be implemented, such as:

- *Data backups* ensure that data that has been lost can be recovered. This technique is standard procedure for most companies since the permanent loss of crucial data can seriously cripple a company's business.

- *Data erasure*, in contrast to backups, is a technique to permanently delete data from a hard drive or other digital media, to ensure that no sensitive data is leaked when a company wants to permanently remove an asset from usage or when required by court order.

- *Data encryption*, or disk encryption, refers to techniques that allow a user to encrypt data in a disk or part of it, such that it remains protected and cannot be decrypted easily by an unauthorized party.

- *Identity-based security* is a method to limit the access to data such that only a user that has been authenticated and has permission to access a piece of data can do so.

These techniques offer ways to protect data, but sometimes this is not enough. Usually due to incorrect programming that cause bugs in the system, vulnerabilities occur in the software that allows unauthorized parties to bypass these mechanisms and get access to data that should be confidential.

2.1.2 Examples of Data Security Breaches

Data Security breaches refer to attacks, usually by unauthorized access, to systems that contain private data. These attacks are commonly made by organized hacker groups to gain leverage against companies or to make a profit by selling the data in black markets. Next, we present some examples of recent Data Security breaches.

- Sony Pictures hack[1]. In 2014, a hacker group leaked confidential data from Sony Pictures in an attempt to gain leverage with the company to make it comply to their demands. The hacker group threatened to commit acts of terrorism in theaters if Sony released a movie related to the North Korean leader.

- Yahoo! data breach[2]. In 2016, Yahoo! reported two separate data breaches occurring in 2014 and 2013, of over 1.5 billion user accounts, including Yahoo! email access, which in turn can reveal bank and family details as well as passwords for other services.

- Ashley Madison data breach[3]. In 2015, a group of hackers stole user data from the adultery website Ashley Madison, and threatened to release usernames and personally identifying information if the website was not shut down.

2.2 Data Privacy

Privacy is an important field in information security because it gives a person his/her personal space and defines his/her personal private information, giving the person the right to decide which personal information is for sharing and which should be kept confidential. The right to privacy also limit the access that other entities, being them the government or private companies, have to personal data.

---

[1] https://www.washingtonpost.com/news/the-switch/wp/2014/12/18/the-sony-pictures-hack-explained/
[2] https://www.theguardian.com/technology/2016/dec/14/yahoo-hack-security-of-one-billion-accounts-breached
[3] http://fortune.com/2015/08/26/ashley-madison-hack/

JP:começas a secção 2.1 com a definição de 'Data Security'. Onde está a definição de 'Data Privacy'

2.2.1 Privacy Definitions

Privacy can be defined as the ability or right that an individual has of protecting his/her personal information and extends to the ability or right to prevent invasions on the personal space of said individual [2].

One of the prime examples of Privacy applied to information technology problems and mentioned in the literature is related to Eletronic Medical Records (EMR) [30]. These records must be handled with extra care because they contain a large number of sensitive information about the patients. The Patient Record Systems should be able to disclose information only to selected personnel, but not all the information about the patient, only what is necessary to proceed in helping the patient. This example illustrates the tension between having access to the data, that can be useful, but at the same time keeping it closed to other users.

2.2.2 Privacy Protection Principles

New concepts have arisen in recent years for privacy specific protection principles [11]. Their definitions are as follows:

- *Unlinkability* is defined as the property that ensures privacy-relevant data cannot be linked across domains that are constituted by a common purpose and context. In other words, multiple actions from the same user/entity must be unlinkable.

- *Transparency* is defined as the property that ensures all privacy-relevant data processing can be understood and reconstructed at any time. Transparency has to cover not only the actual processing but also the planned processing and after processing to fully know what has happened and which entities were involved. Transparency is related to the principles concerning openness and it is a prerequisite to accountability. The user/entity must know and understand how his/her private data is being handled.

- *Intervenability* is defined as the property that ensures mediation is possible concerning

privacy-relevant data processing, in particular by those persons whose data is being processed. Intervenability is related to the rights of an individual, in a way that the owner of privacy-relevant data must have the means to rectify or erase said data.

2.2.3 European Union Legislation

It is also important to mention in the context of this work the current legislation in the European Union (EU) regarding data protection.

The Data Protection Directive[4] is the current law regarding privacy in the EU and is in force since 1995. More recently a replacement has been proposed and accepted in the EU, the General Data Protection Regulation (GDPR)[5], that will take effect in May 2018. Both these laws are regulated by European entities, namely the European Union Agency for Network and Information Security (ENISA)[6] and the Article 29 Data Protection Working Party[7].

According to ENISA, EU data protection law applies to any processing of personal data [10]. This personal data is defined as any information related to an identified or identifiable natural person. In the context of Big Data analysis, the focus is more on indirect identification, which translates into three different approaches: *i)* The possibility of isolating some or all records which identify an individual in a dataset; *ii)* The linking of at least two records concerning the same individual in the same database or in different databases; and *iii)* The possibility to infer the value of an attribute in a dataset from the value of other attributes.

Another important cornerstone of EU data protection law are the principles relating to data quality:

- The *fairness principle* requires that personal data should never be processed without the individual being actually aware of it.

---

[4] http://eur-lex.europa.eu/legal-content/EN/TXT/?uri=celex:31995L0046
[5] http://eur-lex.europa.eu/legal-content/EN/TXT/?uri=uriserv:OJ.L_.2016.119.01.0001.01.ENG&toc=OJ:L:2016:119:TOC
[6] https://www.enisa.europa.eu/
[7] http://ec.europa.eu/newsroom/just/item-detail.cfm?item_id=50083

- The *purpose limitation principle* implies that data can only be collected for specified, explicit and legitimate purposes.

- The *data minimization principle* states that data processed should be the one which is strictly necessary for the specific purpose previously determined by the data controller.

These three principles together mandate that data processing must be done with the consent of the subject, informing the subject of what is the purpose of the processing, and do not deviate from this purpose without informing the subject.

Finally, and also important to mention in the context of this work, are the rights of the data subject according to the EU law. There are two important rights that a subject has: the *right of access* and the *right to object*. The *right of access* ensures that any data subject is entitled to obtain from the data controllers communication of the data that is subjected to processing and to know the logic involved in any processing of data concerning him/her. This is particularly relevant in the context of Big Data analysis because it limits technological lock-ins and other competition impediments, and it enhances transparency and trust between users and service providers. The *right to object* ensures that data subjects have a right to revoke any prior consent, and to object to the processing of data relating to them, giving to the subject the power to remove himself completely or partially to any data processing mechanisms using his personal data.

2.2.4   Examples of Data Privacy Breaches

In recent years we can find a number of attacks made to systems that handle personal information. Next are a number of examples found relevant regarding Data Privacy breaches.

- Target Pregnancy Leak[8]. In 2012, Target, an American retail company, started merging data from user searches and demographics data in order to learn when their customers

---

[8]https://www.forbes.com/sites/kashmirhill/2012/02/16/how-target-figured-out-a-teen-girl-was-pregnant-before-her-father-did/#3001668b6668

were pregnant, in order to approach them with a specific advertisement. This is a clear violation of private sensitive information about their customers and their private life.

- Netflix Prize[9]. In 2007, Netflix created a contest to improve their recommendation system. For that, they released a training dataset, with all the personal information regarding customers removed and customer *ids* replaced by randomized *ids*. Later it was shown that it was not enough when a group of researchers linked public information in another movie-rating website (IMDB) with the released dataset and were able to partially de-anonymize the training dataset, compromising the identity of some users.

- AOL search data leak[10]. In 2006, AOL released to the general public a text file containing search keywords for numerous of their users, intended for research purposes. The users were not identified, but personally identifiable information was present in many of the queries. These queries contained a user *id* attributed by AOL, and an individual could be identified and matched to their account and search history by such information.

- Massachusetts GIC medical encounter database[11]. A researcher from Carnegie Mellon University (CMU) linked the anonymized database (which contained birth date, sex and ZIP code) with voter registration and was able to link medical records with individuals.

2.3 Privacy Implications of Personal Data Processing

Data processing is the conversion of raw data to meaningful information through a process. Data is manipulated to produce results that lead to a resolution of a problem or improvement of an existing situation.

Data Mining is the process of discovering interesting patterns and knowledge from large amounts of data [24]. We will describe the Data Mining process according to the widely used

---

[9]https://www.wired.com/2009/12/netflix-privacy-lawsuit
[10]https://techcrunch.com/2006/08/06/aol-proudly-releases-massive-amounts-of-user-search-data
[11]https://techpinions.com/can-you-be-identified-from-anonymous-data-its-not-so-simple/7627

CRISP-DM model [47], in which the process is separated into six major phases, as described next and in Figure 2.1.

- **Business Understanding:** In this initial phase, the project goals and requirements must be understood from a business perspective, and then converted into a Data Mining problem.

- **Data Understanding:** During this phase, an initial data collection is done, followed by a number of activities in order to get familiar with the data, to understand how the data is organized, to identify if the data has quality problems, or even to detect interesting subsets in the data collected.

- **Data Preparation:** This phase covers all the data preparation tasks to construct the final dataset from the initial raw data collected. These tasks include attribute selection, data cleaning, and transformation of data to fit the modeling tools.

- **Modeling:** In this phase, various modeling techniques are selected and applied, and the underlying parameters are calibrated to optimal values. Usually, there are several techniques for the same Data Mining problem type, and some of these techniques require specific data formats.

- **Evaluation:** At this point in the Data Mining process, one or more models have been developed that appear to have high quality, from a data analysis perspective. These models must be evaluated thoroughly so that we can be certain that they properly achieve the business goals.

- **Deployment:** In this last phase, the knowledge gained by the Data Mining process will need to be organized and presented in a way that the customer can use it. This, of course, depends on the requirements presented at the beginning of the process. An example of a common deployment that results from Data Mining is a simple report on the knowledge obtained.
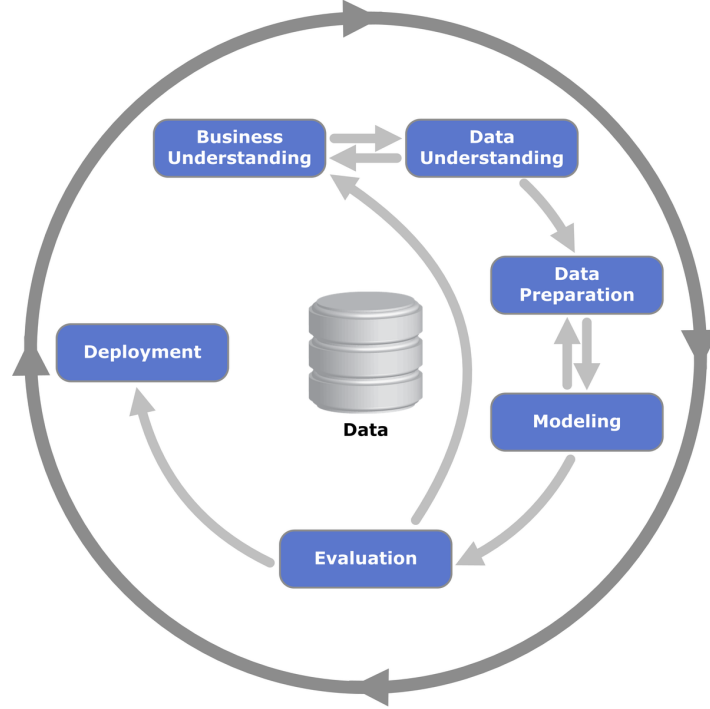
Figure 2.1: Process diagram showing the relationship between the different phases of CRISP-DM [47].

In the Data Mining process, we must be aware that, sometimes, private information about individuals can be used and it may lead to breaches of privacy. We define this private information as Personal Identifiable Information (PII) or Sensitive Personal Information (SPI) [39]. This concept can be defined as information that can be used alone or in conjunction with other information to identify, contact, or locate a single person, or to identify an individual in a context.

2.3.1 Attack Models

When considering the security of a system, one must have into account the concepts of threat model, attack model, and the two types of adversaries, *honest-but-curious* and *malicious*, to understand how to better implement an efficient and trustworthy security layer.

A threat model specifies the potential threats that can be used against a system, by identifying, enumerating and prioritizing them. A well known and defined threat assessment model

is the STRIDE model developed by Microsoft [22].

Cryptographic attacks are used whenever the target system relies on cryptography for protection. An attack model is, in terms of cryptanalysis, a classification of cryptographic attacks specifying the type of access the attacker has to a system when attempting to break an encrypted message. We can summarize the cryptographic attack in the following four categories:

- **Ciphertext-only attack:** In this type of attack, the cryptanalyst has access only to the ciphertext and has no access to the plaintext. This is the most common type of attack, and it is a requirement for modern ciphers to be resistant to it. An example of a ciphertext-only attack is the brute force attack, where the attacker makes a trial-and-error approach to decrypt the ciphertext.

- **Known-plaintext attack:** In this type of attack, the cryptanalyst has access to a number of pairs of plaintext and the corresponding ciphertext.

- **Chosen-plaintext attack:** In this type of attack, the cryptanalyst is able to encrypt arbitrary plaintext and have access to the resulting ciphertext, allowing him to make a statistical analysis on the plaintext state space.

- **Chosen-ciphertext attack:** In this type of attack, the cryptanalyst is able to choose arbitrary ciphertext and obtain the corresponding plaintext.

We can also classify the types of adversaries according to their strategies in two major groups: *honest-but-curious* and *malicious* adversaries. The honest-but-curious adversary follows the protocol in plan, but he will try to extract information from his viewpoint of the protocol to gain some form of advantage or gain access to confidential information. The malicious adversary is one that can deviate from the protocol specification as he desires, and will try to disrupt and/or collect as many information as he can.

2.4   Privacy-Preserving Techniques

In this section, we will describe some of the techniques used in preserving Data Privacy, namely anonymization, Differential Privacy (DP), Secure Multi-Party Computation (SMPC), Garbled Circuits (GC), Oblivious Transfer (OT), Homomorphic Encryption (HE) and Functional Encryption (FE).

2.4.1   Anonymization

Data anonymization is a type of information sanitization technique that has the final intent of privacy protection. This can be achieved by either removing or encrypting PII from datasets so that the individuals whom the data describe remain anonymous [36].

Anonymization techniques use a variety of approaches, for example, suppression, where a piece of information (*e.g.*, name, age) is removed from the dataset; generalization, where data is coarsened into less refined sets; perturbation, where data is modified by adding noise; and permutation, where sensitive associations between entities in the dataset are swapped.

The goals behind anonymization of data are tightly intertwined with the privacy goals we want to achieve for the data being processed. Usually, one or more techniques mentioned above are applied to the data until certain properties are met, for example, *k-anonymity*, *ℓ-diversity* or *t-closeness*.

- *k-anonymity* property states that, for each individual whose data is released in some dataset, he must not be distinguishable from at least k individuals that are also present in the release [44].

- *ℓ-diversity* property is an extension of the *k-anonymity* property, and furthers the anonymization of data by reducing the granularity of the data representation such that for any given record, there exists at least ℓ different sensitive attribute values, in addition to the guarantees made by *k-anonymity* [31].

- *t-closeness* property is a further refinement of ℓ-*diversity*. This property requires that the distribution of a sensitive attribute in any equivalence class is close to the distribution of the attribute in the overall table, effectively limiting the amount of individual-specific information an observer can learn [29].

2.4.2   Differential Privacy

The concept of Differential Privacy (DP) arose due to recent Data Privacy breaches mentioned in section 2.2.4. In [15] it is shown that the security standard for statistical databases, which states that access to a statistical database should not enable one to learn anything about an individual that could not be learned without access, is not achievable because of the existence of auxiliary information, *i.e.*, information available from sources other than the statistical database.

DP is a process of maximizing the accuracy of queries in statistical databases while minimizing the chances of identifying its records. The core of the procedure is based on *randomized response* [46], giving the possibility to infer statistical information from the dataset, while still ensuring high levels of privacy.

Detailed information about DP and algorithms designed to achieve it are described in [16].

2.4.3   Secure Multi-Party Computation

Secure Multi-Party Computation (SMPC) (also known as secure computation, multi-party computation, or privacy-preserving computation) is a protocol for different parties to jointly compute a function over their inputs while keeping those inputs private. The problem of computing functions while also preserving the privacy of the inputs is referred in the literature as a SMPC problem [48]. Generally speaking, a SMPC problem deals with computing any probabilistic function on any input, while also ensuring the correctness of the computation and guaranteeing that no more information is revealed to a participant in the computation that can be inferred from that participant's input and output [20].

A strategy to solve these problems is to trust an external entity (a trusted third party), that can mediate the computation. This approach can be risky because it requires a third party that all participants agree to trust, which can sometimes be difficult to find. Sometimes, the data has such high degree of importance to the participants that even disclosing them to a trusted third party is not viable.

When building a SMPC protocol, the most important properties that must be ensured are *input privacy* and *correctness* [19]:

- The *input privacy* property states that no information about the private data held by the parties can be inferred during the execution of the protocol. The only inferences about private data are those that could be inferred from seeing the output of the computation made by the protocol.

- The *correctness* property relates to the existence of malicious parties that could try to deviate from the normal functioning of the protocol. In these cases, the protocol should prevent honest parties to output incorrect results. The approach to implementing the correctness property comes in two alternatives: either the protocol is robust, *i.e.* it guarantees that the honest parties compute the correct output; or the honest parties abort the computation if they find an error during the execution of the protocol.

The first implementation of secure computation was introduced as Secure Two-Party Computation (STPC) [48]. It is a simplification of the problem of SMPC, and a known protocol for STPC is Yao's GC protocol, which is detailed in subsection 2.4.5. In STPC there are only two participants in the computation, and usually one is responsible for starting and encoding the computation mechanism, while the other is the one responsible for evaluating the computation. In multi-party computation, the parties have no special roles and, instead, the encoding is shared amongst the parties, by secret sharing, and the evaluation is made by a protocol.

Some recent implementations of SMPC protocols are based on *Secret Sharing*. Secret Sharing

MP:add some more text about the protocol.

allows one party to distribute a secret over a number of parties by distributing shares to each party. Three of the types of secret sharing techniques more commonly used are: Shamir Secret Sharing [40], Additive Secret Sharing and Replicated Secret Sharing.

2.4.4   Oblivious Transfer

Oblivious Transfer (OT) [35] is a protocol in which a sender transfers one of the potentially many pieces of information he has to a receiver, but remains oblivious as to what piece has been transferred. Let s^0 and s^1 be two strings held by a sender that wants to transfer one of the strings to a receiver, holding a selection bit b; the protocol allows for only one of the inputs s^b to be transfered; the receiver learns nothing about s^{1-b}, and the sender does not learn b.

An interesting implementation of OT is the one done by Pinkas and Naor [32]. In it, the authors describe an extension to the basic 1-out-of-2 OT protocol, to a 1-out-of-N protocol, and a k-out-of-N protocol.

2.4.5   Garbled Circuits

Yao's Garbled Circuits (GC) [49] are a cryptographic protocol that allows two mutually mistrusting parties to evaluate a function over their private inputs without resorting to a trusted third party. In other words, GC allow parties holding input x and y to evaluate an arbitrary function $f(x, y)$ without leaking any information about their inputs beyond what is inferred from the function output. The idea behind GC is that one party prepares an encrypted version of a circuit that computes $f(x, y)$ and the second party then computes the output of the circuit without learning any intermediate values.

Some optimizations have been proposed for Yao's GC. Kolesnikov and Schneider [27] present a technique that eliminates the need to garble XOR gates. Pinkas *et al.* present a technique that reduces the size of a garbled table from four to three ciphertexts [34].

2.4.6 Homomorphic Encryption

Homomorphic Encryption (HE) [37] is a cryptographic technique that allows computations to be carried with the ciphertext, so that, when decrypted, the resulting plaintext reflects the computation made. In other words, HE allows to make some computation over the ciphertext, for example, addition, without decrypting it, and the result is the same as making that computation on the plaintext. This is of great importance because it allows chaining multiple services that make computations on a ciphertext, without the need to expose the data to those services.

Homomorphic cryptosystems can be classified into two distinct groups: partially homomorphic cryptosystems and fully homomorphic cryptosystems.

- In Partial Homomorphic Encryption (PHE), there is only one operation, like addition, multiplication or XOR, that is allowed by the homomorphic property, but not for all possible operations. Some examples of existing partially homomorphic cryptosystems are: ElGamal cryptosystem [17]; Unpadded RSA [38]; and Pailier cryptosystem [33].

- In Fully Homomorphic Encryption (FHE), it is possible to make two different operations on the ciphertext, namely addition and multiplication. This concept was first introduced in [37], and for many years it remained just as a concept, until recent years when fully homomorphic implementations were developed, for instance, Gentry's cryptosystem [18].

2.4.7 Functional Encryption

In Functional Encryption (FE) systems, a decryption key allows a user to learn a specific function of the encryption data, while also stopping that same user from learning anything more about the encrypted data. In other words, having a secret key only allows for a specific computation of a function over the ciphertext [5]. When comparing FE with HE, the main

difference is that, in HE, the result of a computation in the ciphertext remains encrypted, while in FE, the result of the computation is available in the clear.

JP:se calhar é conveniente expandir um pouco esta frase

2.5   Privacy-Preserving Machine Learning

In the context of Data Mining, ML techniques come as an important addition to the data processing step. An example of that is the *Classification* method. Classification is described by a two-step process, in which a classification algorithm is employed to build a classifier for the data by analyzing a training set made of tuples of data and their associated labels, and then the classifier is used to predict class labels for new data. As such, because the large size of the datasets produced in Big Data operations, classification algorithms have a large quantity of data to learn from, making them less prone to erroneous classification of new data.

The conjunction between ML and privacy-preserving comes from the need to do knowledge learning over large datasets, while also maintaining protection over the privacy of the data, without degrading the quality of data by using anonymization techniques.

In the field of ML, we can identify a number of algorithms that can be used in Data Mining. In Table 2.1, we list some of them, giving a brief description, and identifying a privacy-preserving technique that has been implemented with it.

Table 2.1: PPML algorithms.

Machine Learning Algorithm	Short summary	Privacy-Preserving Technique	Reference
Decision Trees	Protocol for distributed learning of decision-tree classifiers.	SMPC	[8]
Naive Bayes	Differentially private naive Bayes classifier. Centralized access to the dataset.	DP	[45]
SVM	Algorithm for Support Vector Machines classification over vertically partitioned data.	SMPC	[50]
k-NN	Nearest neighbors of records in horizontally distributed data.	SMPC	[41]
k-Means	k-Means clustering based on additive secret sharing.	SMPC	[14]
Logistic Regression (LR)		DP	[9]

2.6 Use Cases

In terms of PPML and its applications, it is important to distinguish the subject of the data that is being processed. Different data can be subject to different constraints regarding laws and privacy. Some sensitive data may be only processable in a local environment, while other data can only be processed in a less individualized way.

We now detail three relevant and different subjects that are subject to different privacy constraints.

- **Health records:** Healthcare systems are one of the examples where vast amounts of data are collected every day, and it is of relevance to do Data Mining on patient records, for a better understanding of patients and to improve the healthcare system. Patient records contain very sensitive information about individuals and cannot be processed without the Data Mining system being in compliance with the legislation on Data Privacy, therefore it is of interest to build privacy-preserving systems for the healthcare system, so that hospitals and other health-related organizations can share and infer knowledge without violating the privacy of their patients.

- **Governance (students and taxes):** Bogdanov *et al.*[4] made a statistical study in 2015 using SMPC to look for correlations between working during university studies and failing to graduate in time. For this study, it was necessary to link the database of individual tax payments and the database of higher education universities. These types of government data are subject to strict legislation and cannot simply be handled without strong privacy guarantees. To solve this problem, a SMPC system was developed and deployed that could assure a level of privacy that would be in compliance with the laws on Data Privacy. The data processing steps were all made using SMPC between three parties, using OT so that each party would not know each other inputs.

  In the end, the study using SMPC was compared with an anonymized study using k-anonymity with $k = 3$. The loss of samples in the latter was 10%-30%, depending on the demographic group, thus suggesting that producing studies on existing databases using SMPC to enforce privacy can give more accurate results than the same study run using k-anonymity measures.

- **Human mobility:** Another subject that provides great challenges in the field of Data Privacy are the mobility traces generated by people when driving, walking, etc. Mobility traces are highly unique so it is possible, even after anonymizing the dataset, to link an individual to his mobility patterns, as shown by Montjoye *et al.* [12]. Since mobility data contains the approximate whereabouts of individuals, it can be used to reconstruct their

JP:até agora tens usado sempre um estilo diferente para as referências. Convém ser consistente no estilo usado.

movements across space and time. Applying privacy-preserving techniques to process this highly sensitive data can result in better geographic-based recommendation systems.

2.7   Summary

The previous sections provided an overview of the state of the art surrounding PPML.

expand

3

BARD

In this chapter we present the project from Altran that this thesis is a part of.

Expand this introduction

3.1 Motivation

The idea for this project came from the rise in the Big Data market. The evolution of Big Data the last years, caused by the increasing number of devices connected to the Internet, provided analysts with the data to develop and improve systems in a varied scope of subjects, such as Healthcare, Industry, etc. But this data has private information about individuals, and, as we have shown before, the processing of it without certain precautions leads to reaches of private information. Big dAta pRivacy by Design platform (BARD) project comes to solve this problem, by raising awareness to it, and provide solutions, in the form of methods and protocols to build privacy-preserving solutions for Big Data systems.

3.2 Objectives

The objectives defined for BARD are described bellow:

- Define methods to support protection of personal data for harvesting, sharing, quering and processing data assets, for supporting all the decisions to be taken while developing the platform.

- Analyze the effects of the existing legislation on the construction of privacy-preserving solutions.

- Conceive a Privacy by Design architecture which balances the needs of the data subject, the demands of the data consumer, and the legal constrains.

- Develop a Privacy by Design platform based on a reference architecture for the entire data flow process, in order to maximize value with both the people and companies.

As so, the expected output of BARD is a platform that provides mechanisms for the protection of personal data, that complies with the current legislation, and that assures Privacy by Design and by Default.

3.3   Architecture

We now present the architectural specifications of BARD. The CRISP-DM model was used as a baseline for representing data life cycle. BARD focuses on Data Quality (*e.g.*: cleaning, annotation), Data Representation (*e.g.*: anonymization, ciphering) and Data Processing (*e.g.*: Computation on ciphertext).

The Data Quality step refers to the transformation of raw input data into a structured, consistent and, whenever possible, complete representation. An important aspect to mention in this step is that the data must be processed in plaintext by a trusted entity, meaning it is done by the owned of the data or an entity in which the owner of the data trusts and has explicit permission to perform the operations. The Data Representation step refers to the protection of Personal Identifiable Information (PII) contained in the data.   This data should be integrated using privacy-preserving data integration techniques, data should be aggregated using anonymization techniques and data should be represented using either hashing techniques or homomorphic cryptosystems. The Data Processing step is done in two different approaches. On one hand, Secure Multi-Party Computation (SMPC) techniques, such as Garbled Circuits (GC) and Homomorphic Encryption (HE), are performed over the data. On the other hand, Machine Learning (ML) algorithms, adapted to work with hashed or encrypted data, are used in performing knowledge learning.

We now describe the internal structure that the project aims to provide. As stated in the objectives, the main goal of BARD is to produce a platform that will provide mechanisms so that companies can perform privacy-preserving computations for ML algorithms, that are respectful of user privacy, and comply with the legislation. A representation of the internal structure of that platform is described by the following components:

- **A dataset** to train the ML algorithm, or the values representing the already trained algorithm.

- **A sample** or a set of samples that represent the input of the "user", to be predicted.

- **A prediction algorithm** that depends on the ML algorithm and the privacy-preserving technique chosen.

- **A set of toolkits** for each of the techniques used.

These components altogether allow the user of the platform to perform privacy-preserving ML over data of his own choosing.

BARD Scheme
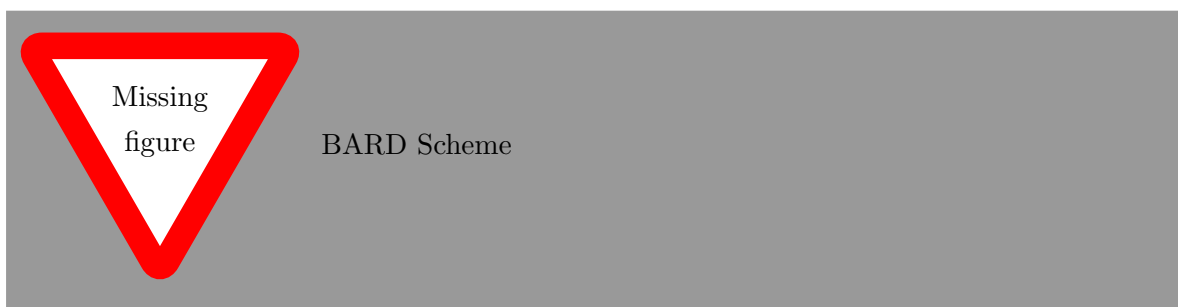
fazer um esquema para o BARD. não só de arquitectura, mas como as pessoas interagem com o BARD.

passos no flow geral de processamento de dados : recolha de dados, pre-processamento, etc

3.4   My Contributions to BARD

It is important to mention that, in the following chapters of this thesis, we will present results that were not done by us. Since this thesis is part of a larger project developed in Altran, the implementation was done by a team of developers. As so, some results are presented for completeness of the solution, but were not obtained by us. Those results are the ones obtained using HE.

Our contributions to the project are then the development of a solution using the toolkit VIPP for privacy-preserving computations using GC. This includes the development of the baseline part of the implementation, the expanded algorithms, the testing of the various toolkits and the actual development of the final solution with GC. Throughout the rest of the document we will highlight the parts that were not developed by us.

3.5   BARD API

3.6   Summary

Summary chapter 4.

Implementation 4

When building a platform in the scope of privacy-preserving Machine Learning (ML), we must consider not only the traditional steps in data processing mentioned in section 2.3 and in Figure 2.1, but also have an increased care when preprocessing data to incorporate the cryptographic techniques that we used. This chapter describes the work that was done in implementing a privacy-preserving ML platform. We start off with a description of the datasets chosen to evaluate our platform 4.1. Then, we explain the preprocessing that was done to those datasets 4.2. Section 4.3 presents the baseline implementation of the chosen ML algorithms, resorting to a widely used ML toolkit for Python. In section 4.4 the implementation of the prediction phase of the algorithms is detailed. Finally, in section 4.5, we detail which cryptographic protocols we used, why, and how we implemented them, mentioning which toolkits were used.

4.1 Datasets Used

For running the experiments, we used datasets that are highly used in the literature, namely the Breast Cancer Wisconsin dataset[1], the Pima Indians Diabetes dataset[2], the Credit Approval dataset[3], and the Adult Income dataset [4]. Table 4.1 presents a brief description of each dataset.

---

[1] https://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+(Diagnostic)
[2] https://archive.ics.uci.edu/ml/datasets/pima+indians+diabetes
[3] http://archive.ics.uci.edu/ml/datasets/credit+approval
[4] https://archive.ics.uci.edu/ml/datasets/adult

JP:replace with short description of datasets.

Table 4.1: Datasets used in Big dAta pRivacy by Design platform (BARD).

Dataset	Subject	Instances	Features
Breast Cancer Wisconsin	HealthCare	569	30
Pima Indians Diabetes	HealthCare	768	8
Credit Approval	Finance	690	15
Adult Income	Governance	48842	14

4.2   Data Preprocessing

Although our data is obtained from publicly available data sources, it is still required to do some preprocessing on the data. For example, the datasets contained missing values on some of the entries that needed to be addressed. Another example is the existence of categorical data that ML algorithms cannot process. We now describe some of the techniques we used in the data preparation phase for the datasets described in Table 4.1.

- **One-hot Encoding**[26] was initially used in digital circuits in order to determine the state of a state machine without using a decoder. The binary code is converted in a group of bits in which only one bit can have the value high (1), and all the others low (0). For example, the binary 110 is converted to 01000000. In ML, we apply this technique to deal with the problems created by using datasets with categorical (or nominal) data. Most ML algorithms require numerical representations in the data. A solution can be to assign an integer number to each different value present in the data, but this leads to the model assuming a natural ordering between categories that may not exist. Using one-hot encoding in categorical data allows us to circumvent this problem, creating additional binary variables for each unique value, *e.g.* if a variable describing pets has the values "dog", "cat" and "fish", after encoding three more variables will be added to the dataset, each representing a possible value. Then, a high (1) will be placed on the binary variable that represents the original value, and low (0) on all the others. Using the pets example, if the original value was "dog", the resulting three binary variables

will be high on the binary variable representing "dog", and low on the other two ("dog" becomes $1, 0, 0$).

- **Feature Scaling** is a technique to normalize the range of data. For some ML algorithms, having broad range of values in one of the features may cause this feature to govern the modeling. There are different ways of achieving this, for example, *rescaling*, where the range of values is scaled to a target range (usually $[0, 1]$ or $[-1, 1]$), or *standardization*, where the features are rescaled so that they have the properties of a standard normal distribution, *i.e.* the mean average at zero and the standard deviation from the mean at one.

In our implementation, we used one-hot encoding in the Adult Income dataset and in the Credit Approval dataset. These two datasets contain categorical features that required the use of encoding to be processed by the ML algorithms.

In the Adult Income dataset, we have multiple examples of categorical features, namely the *workclass*, *education*, *marital-status*, *occupation*, *relationship*, *race*, *sex* and *native-country*. these were all encoded so that only numerical values were left. Of course, this caused a exponential growth on the number of features, going from the initial 14 to 108. In the Credit Approval dataset, we encoded all the non-numerical features, changing the number of features from 15 to 51. Feature scaling was used in all of the datasets, in order to reduce the errors caused by wide ranges. We used rescaling of all values to numbers between 0 and 1.

Besides the techniques mentioned above, it was also necessary to deal with the missing values that existed in the datasets. These values must be dealt with, since many ML algorithms cannot work with them. So, for the numerical features, we replaced those missing values with the median of the other existing values. In the case of the categorical features, the missing values are considered as another valid value for that feature, in order to make the calculation of one-hot encoding to the whole dataset easy.

Finally, it was required for our setup a validation and a testing set, which some of the datasets did not contain originally. So we divided those datasets into training, validation and testing

sets, using a proportion of 70/15/15. When a testing set already existed, the validation set was created from the training set, and forced to have the same size as the testing set.

4.3   Baseline

The baseline approach consists on setting up ground values so that meaningful comparisons can be achieved. For understanding the overhead created by privacy-preserving technologies, we implemented the baseline using the publicly available ML toolkit, scikit-learn for Python[5].

In order to understand and explain what was done and why, we detail next the ML algorithms that were implemented in the baseline and in sections 4.4 and 4.5.

4.3.1   Decision Trees

Decision Trees (DT) are a decision support tool that uses a tree-like graph that represents decisions and possible outcomes of those decisions. It is an algorithm that is composed of conditional control sequences. DT learning uses DT as a predictive model for classification. These DT are composed of nodes and leaves. The nodes represent decisions to take, or more specifically, thresholds that a feature is compared against, in order to decide which branch of the tree to follow. The leaves represent class labels.

To build a DT classifier there are a number of algorithms that can be used, each with different approaches and benefits. For example, ID3, C4.5, C5.0 and CART [43]. These algorithms focus in maximizing one or more metrics, such as Gini impurity or information gain.

The classification process of a sample in DT is an intuitive method. Each node of the tree has the information on which feature in the sample to compare against the threshold. So, classifying a sample in DT is done by traversing the tree from the top, comparing the values selected on each node with it's respective threshold, and depending on the result, choose one child node or the other. When a leaf is reached, a class label is retrieved from the leaf and

---

[5]http://scikit-learn.org

assigned to the sample, ending the classification process.

4.3.2 Support Vector Machines

Support Vector Machines (SVM) are supervised learning models that analyze data used for classification and regression analysis. An SVM model represents the examples as points in space, mapped so that the margin between the two classes for the data is as wide as possible. The vectors that define this margin, or *hyperplane*, are called support vectors. For classifying new samples, one maps them in that space, and predicts which class they belong based on which side of the gap they lie.

For calculating the SVM for linear classification, and in the case of a *hard-margin*, *i.e.*, when the training data is linearly separable, we select two parallel hyperplanes so that the distance between them is as large as possible. These hyperplanes can be described by the equations in 4.1.

$$\vec{w}.\vec{x} - b = 1,$$
$$\vec{w}.\vec{x} - b = -1$$

$$(4.1)$$

where $\vec{w}$ is the normal vector to each hyperplane, $\vec{x}$ is the training set and b is a scalar number. To maximize the distance between hyperplanes, we minimize the value of $||\vec{w}||$ subject to $y_i(\vec{w} \cdot \vec{x}_i - b) \geq 1, for \ i = 1, \ldots, n$, where the y_i are either 1 or -1, depending on the class label, and n is the number of samples in the training set.

In the case where the data is not linearly separable (*soft-margin*), we minimize instead the *hinge* loss function given by the equation in 4.2.

$$f(w, \lambda) = \left[\frac{1}{n} \sum_{i=1}^{n} \max\left(0, 1 - y_i(\vec{w} \cdot \vec{x}_i - b)\right) \right] + \lambda \|\vec{w}\|^2 \qquad (4.2)$$

where the parameter λ determines the tradeoff between increasing the margin-size and ensuring that the $\vec{x}_i$ lie on the correct side of the margin.

For SVM non-linear classification, we employ a *kernel trick*, in which the dot product is replaced by a non-linear kernel function. The most used kernels are mentioned bellow.

- Linear: $k(\vec{x_i}, \vec{x_j}) = (\vec{x_i} \cdot \vec{x_j})$

- Polynomial: $k(\vec{x_i}, \vec{x_j}) = (\vec{x_i} \cdot \vec{x_j})^d$

- Radial Basis Function (RBF): $k(\vec{x_i}, \vec{x_j}) = \exp(-\gamma \|\vec{x_i} - \vec{x_j}\|^2)$

The classification of new samples in SVM is done using the scoring function in 4.3, where each testing sample x is attributed to a prediction label.

$$f(x) = \sum_{i=1}^{m} \alpha_i K(x_{SV}^{(i)}, x) + b \tag{4.3}$$

where α_i is the coefficient associated with the support vector $x_{SV}^{(i)}$, K is the kernel function chosen, and b is a scalar number.

4.3.3   *k*-Nearest Neighbors

k-Nearest Neighbors (*k*-NN) learning algorithm is based in the principles of pattern recognition. This algorithm is among the simplest of all ML algorithms, since it does not require a training step. To classify a new sample, we calculate the Euclidean Distance of it with the training samples, and discover the k samples that are close to it. Then, by majority vote of its k neighbors, the class label is assigned to the new sample.

4.3.4   *k*-Means

k-Means (*k*-M) clustering is a method commonly used to partition a dataset into k groups. It proceeds by selecting k initial cluster centers (centroids) and then iteratively refining them.

this refining is done in two distinct steps. 1) Each instance is assigned to its closest cluster. This is done by calculating the Euclidean Distance between each instance and each cluster center.Then, the lowest distance indicates which cluster the instance must be assigned to. 2) Each cluster center is updated to be the mean of all the instances assigned to it. The algorithm stops when the centroids no longer change position. This means that, depending on the data, it is not guaranteed that the optimal solution is found.

The classification of each testing sample is done similarly as the first step,*i.e.*, by computing the Euclidean Distance of the new sample with each centroid discovering the cluster whose centroid is closest to it. The label of the cluster becomes the predicted label of the sample.

4.3.5 Logistic Regression

Logistic Regression (LR) is a regression model in which the dependent variable is categorical. This variable is usually binary, *i.e.*, it can only take two values, usually 0 or 1 that represents opposite outcomes such as "win/lose" or "healthy/sick". This binary logistic model is used to estimate the probability of a binary response based on one or more variables. To define LR, one must begin with the Logistic function, which in turn is given by equation 4.4.

$$\sigma(t) = \frac{e^t}{e^t + 1} = \frac{1}{1 + e^{-t}} \tag{4.4}$$

where t is the input. If we express t as $t = \beta_0 + \beta_1 x$, we can write the logistic function as in equation 4.5.

$$F(x) = \frac{1}{1 + e^{-(\beta_0 + \beta_1 x)}} \tag{4.5}$$

To classify testing samples we use equation 4.6 to attribute a prediction label to each sample x.

acabar a seccao LR

$$f(x) = \beta_0 + \sum_{i=1}^{m} \beta_i x_i \qquad (4.6)$$

4.4   Expanded Algorithms

Taking into account that the baseline implementation was done using scikit-learn, we could not explicitly compare execution times, due to the fact that the operations in the toolkit were made in a "black box" mode. To solve this problem, we implemented the evaluation prediction processes of the ML algorithms without using the toolkit, *i.e.*, directly from the algorithm equations.

We successfully recreated the prediction processes by retrieving from the toolkit the specifications for each ML model, and we now enumerate them.

- For the DT, it was needed to retrieve all of the model, *i.e.*, the binary tree, so that we could traverse it according to each testing sample, comparing the feature with the threshold of each node, and choosing which branch of the tree to follow, until a label is reached.

- In the case of SVM, we retrieved from the toolkit all the coefficients needed to compute equation 4.3, namely the support vectors, the α coefficients for each support vector, the kernel function that was used(linear or polynomial), the exponent for the polynomial kernel if needed, and the scalar value b.

- For k-M, we just required from the toolkit the centroids for each cluster, as well as the prediction labels associated with each one. The classification of each testing sample was done by discovering which centroid was closest to it.

- In LR, we extracted the $\beta = (\beta_0, \beta_1, ..., \beta_n)$ that appear in equation 4.6. β_0 is the intercept from the linear regression, and β_i are each regression coefficient that are multiplied by each feature of the sample.

In the case of k-NN, since training does not exist, there was no need to use the toolkit, so no information was retrieved from it, and it was not required to do a expanded version of the algorithm.

4.5   Cryptographic Domain

In the final step of our implementation, we made adjustments to the evaluation processes of the ML algorithms discussed above so that we could apply two privacy-preserving techniques, Garbled Circuits (GC) (described in section 2.4.5) and Homomorphic Encryption (HE) (described in section 2.4.6). These two techniques offer different means to obtain privacy-preserving computations, and we must consider them when choosing which ML algorithm to use with each cryptographic algorithm. GC builds ciphered Boolean circuits, which means that most of the computations are possible to implement on them. However, arithmetic computations require a large number of logic gates, creating a overhead that makes GC very slow. So, for some of the ML algorithms, we used a HE system, since it offers arithmetic operations as a core operation. Due to these constrains, we decided to adapt the DT and k-M to be evaluated using GC, and SVM and LR to be evaluated using HE.

4.5.1   Garbled Circuits and Decision Trees

The process of evaluating a DT in a privacy-preserving context is similar to evaluating in the usual manner, as described in 4.3.1. The main differences are the use of ciphered Boolean circuits instead of operations, *i.e.*, basic operations such as additions, comparisons, are replaced with logic gates, and the evaluation of the DT involves evaluating every single node in it, to disclose the least possible information due to observation of the computations.

In figure 4.1 we show how the computations are done inside each node of the DT. Each node contains the *featureID*, the ID of the feature to be selected from the sample to be classified, and the *threshold*, the value that is compared against the selected feature and that decides which branch of the tree to follow next. The first MUX gate selects from the sample the

feature to be compared. The GREATERTHAN gate compares the selected feature with the threshold. Then, the value from the comparison (0 or 1) is used as a selection bit in the second MUX to choose the *next_featureID* and *next_threshold* for the next node in the tree.
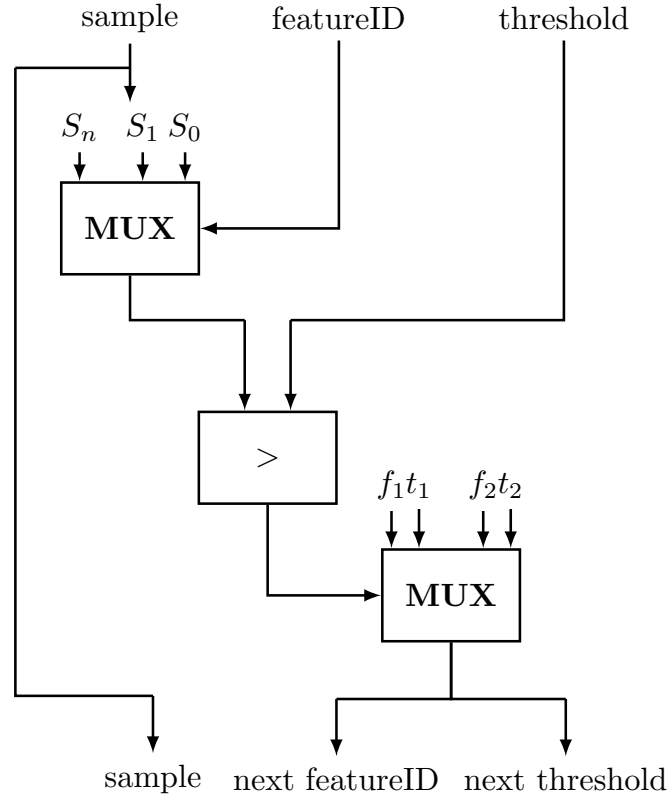
Figure 4.1: Diagram representing the Boolean circuit of each node in a DT.

It is also important to mention that the trees that we used are always complete, *i.e.* the number of nodes n is always the maximum possible, and it can be defined as $n = 2^{h+1} - 1$, where h is the *height* of the tree. We can see in figure 4.5.1 the implications of this expansion of the binary trees.

(a) Original DT.   (b) Complete DT.
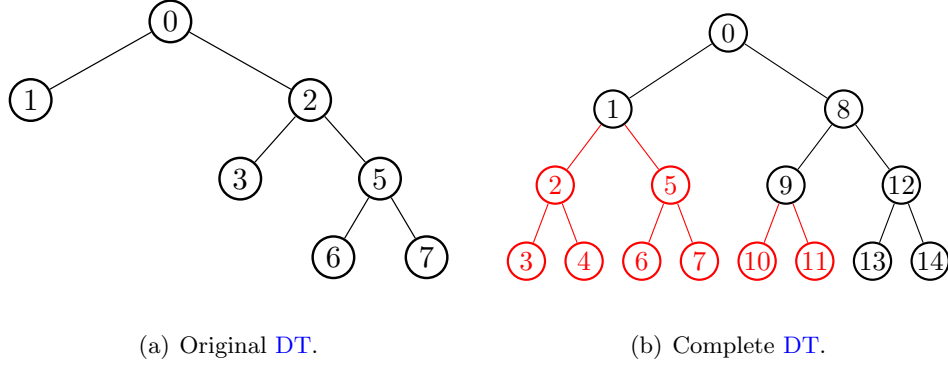
Figure 4.2: Expansion of binary trees.

In figure 4.2(a) we have a binary DT with $height = 3$ and with different path lengths to the leaves, depending on the branch of the tree taken, but on the tree in figure 4.2(b), all paths have the same length. With this, we can effectively hide the sparseness of the tree, which can leak relevant information about the original data. However, this solution increases exponentially the total amount of nodes that need to be evaluated, and that decreases performance significantly, as we will see in chapter 5.

4.5.2   Garbled Circuits and k-Means

As in the previous section, evaluating the k-M algorithm in a privacy-preserving manner is similar to evaluating in the usual manner. We took the operations in the prediction step and transformed them in Boolean circuits, with logic gates such as multiplexers, adders, etc.

In figure 4.3 we show how we designed the circuit to represent the k-M prediction. The ED blocks represent the operations to perform the Euclidean distance between the sample and each centroid provided by the k-M model. The ARGMIN block represent the operations needed to choose, between all the distances calculated, which of it is the smallest.
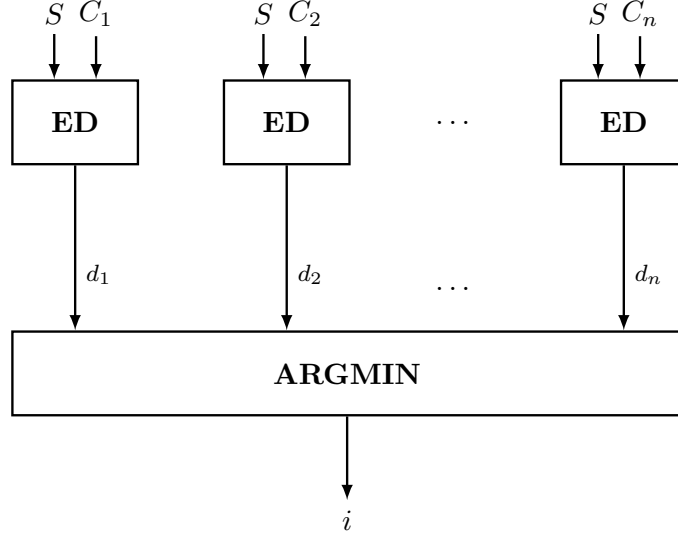
Figure 4.3: Circuit representing the prediction of the k-M algorithm.

As we show in ref(evaluation-k.means), there is an exponential decrease in performance when the number of clusters increases, or the sample has a larger number of features. This is caused by the multiplications that are done calculating the Euclidean distance. Usually this is reason to use HE instead, but the algorithm to implement comparisons is hard to implement [3].

is this
justifi-
cation
enough?

4.5.3   Homomorphic Encryption and Support Vector Machines

For the SVM algorithm, we only considered the linear kernel, as it simplifies the scoring function. The equation 4.3 is then simplified to the following:

$$f(x) = \sum_{i=1}^{m} \alpha_i x_{SV}^{(i)} x + b = \sum_{i=1}^{m} \alpha_i \sum_{j=1}^{n} x_j x_{SV}^{(i,j)} + b \tag{4.7}$$

To compute this function using a Fully Homomorphic Encryption (FHE) system, we must convert it to:

$$f_{FHE}(x) = D_k\left(\sum_{i=1}^{m} E_k(\alpha_i) \cdot \sum_{j=i}^{n} E_k(x_j) \cdot E_k(x_{SV}^{(i,j)}) + E_k(b) \right) \tag{4.8}$$

where E_k represents encryption and D_k represents decryption.

Converting equation 4.8 to be computed using a Partial Homomorphic Encryption (PHE) system is equally straightforward, but this must be done under two assumptions: 1) the data to be evaluated (x) and the model parameters ($\alpha_i, x_{SV}^{(i,j)}, b$) must come from two different parties, and 2) the owner of the model parameters must be the one processing the data. Under these assumptions, the scoring function for a multiplicative PHE system becomes:

$$f_{FHE}(x) = D_k \left(\prod_{i=1}^{m} \left(\prod_{j=1}^{n} E_k(x_i)^{x_{SV}^{(i,j)}} \right)^{\alpha_i} \boldsymbol{.} E_k(b) \right) \qquad (4.9)$$

4.5.4   Homomorphic Encryption and Logistic Regression

For the LR algorithm, in order to use a FHE system, the prediction function described in 4.6 must be converted to:

$$f_{FHE}(x) = D_k \left(E_k(\beta_0) + \sum_{i=0}^{m} E_k(\beta_i) \boldsymbol{.} E_k(x_i) \right) \qquad (4.10)$$

where E_k represents encryption and D_k represents decryption.

Like in the previous subsection, converting it to be computed using a PHE system is equally straightforward, but this can only be done under the same two assumptions: 1) the data to be evaluated (x) and the model parameters ($\beta_0, \beta_1, \ldots, \beta_m$) must come from two different parties, and 2) the owner of the model parameters must be the one processing the data. Under these assumptions, the linear prediction function for a multiplicative PHE system becomes:

$$f_{PHE}(x) = D_k \left(E_k(\beta_0) \boldsymbol{.} \prod_{i=1}^{m} E_k(x_i)^{\beta_i} \right) \qquad (4.11)$$

4.6   Use Case: Healthcare

As mentioned in section 2.6, healthcare systems generate vast amounts of data every day. Processing this data can be beneficial for both the health institution (hospitals, clinics) and

for the patients. However, usage of these Eletronic Medical Records (EMR) cannot be freely done by institutions without consent of the patients and compliance with data protection legislation. As a result, this processing is performed *in-house*, with only a few exceptions[6]. The problem is that developing and/or maintaining a Data Mining infrastructure in an institution amasses costs that can it may not be willing to support.

Our contribution to mitigate this standoff between the gains and costs of Data Mining EMR, is to provide a product that removes the costs of maintenance and development from the institutions, while at the same time provide enough privacy guarantees to comply with existing legislation.

We now describe a typical use case scenario for privacy-preserving processing of EMR.

- **Description:** Design and implementation of a platform to process EMR in order to improve treatments and diagnoses, while maintaining identities private. This is achieved by training models using this data and then predict medical conditions for future patients. All the computations should be done resorting to privacy-preserving techniques.

- **Actors involved:** Healthcare institutions, patients, medical staff.

- **Preconditions:** Access to data and to EMR of patients. Consent from each patient regarding the processing of his/her data.

- **Basic Flow:**

  1. The institution supplies the platform with data to train the models for one or more ML algorithms. This training must be done in an encrypted and/or anonymised domain.

  2. A new patient arrives at the institution and it is asked if he/she consents to the use of the platform to speedup his/her diagnosis, including consent to data collection and data processing. If the patient agrees, the process can continue.

---

[6]https://www.reuters.com/article/us-health-medicalrecords-sharing/few-u-s-hospitals-can-fully-share-electronic-medical-records-idUSKCN1C72UV

3. Data of the patient is collected by a person from the medical staff, including his/her symptoms, medical history, etc.

4. This data is supplied to the platform, and the platform performs one or more predictions, depending on the number of models the platform has, using privacy-preserving techniques to do so.

5. The platform informs the doctor of what are the results of the predictions.

6. The doctor decides on the appropriate medical action, taking into account his medical background and the information supplied by the platform.

- **PostConditions:** The platform has received data from the institution. The platform has trained different instantiations of ML algorithms.

4.7 Summary

Summary chapter 3.

5

Evaluation

In this chapter we describe the experiments that were conducted regarding the implementation of Machine Learning (ML) algorithms using privacy-preserving techniques. In section 5.1, we present the metrics used in the experiments. Section 5.2 describes the setup that was used to run the experiments, as well as the toolkits used and the parameters chosen for those toolkits. In section 5.3, we present the best baseline results obtained for the datasets in question and in section 5.4 we compare those results with the ones obtained using the toolkits. In section 5.5, we present the execution times for the combinations of ML algorithms and privacy-preserving techniques that were implemented, and in section 5.6 we present the communication costs for those combinations. Finally, in section 5.7 we make some final observations about the results obtained.

5.1 Evaluation Metrics

To evaluate our implementation, it is important to understand the metrics that were considered. To define the evaluation functions bellow, we present the notation in table 5.1.

Table 5.1: Notation.

Real label	Predicted label	Event
+1	+1	True Positive (TP)
+1	-1	False Negative (FN)
-1	+1	False Positive (FP)
-1	-1	True Negative (TN)

Accuracy is defined as how much of the measurements of a value differ to the real value. In our

implementation, it represents how many times the predictions calculated by the ML generated models match the class of the testing samples. In mathematical terms, it is represented by:

$$accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{5.1}$$

Precision is defined by the fraction of relevant instances among the retrieved instances 5.2. *Recall* is defined by the fraction of relevant instances that have been retrieved over the total of the relevant instances 5.3.

$$precision = \frac{TP}{TP + FP} \tag{5.2}$$

$$recall = \frac{TP}{TP + FN} \tag{5.3}$$

F-measure is a measure of a accuracy of a test. It considers both the precision and the recall of the test to compute the score:

$$F_1 = 2 \cdot \frac{precision \cdot recall}{precision + recall} \tag{5.4}$$

A *confusion matrix* is a specific table layout that allows visualization of the performance of a ML algorithm. Each row of the matrix represents the instances in a predicted class while each column represents the instances in an actual class. In 5.1 we give an example on how to compute a confusion matrix for the binary case.

$$confusion\_matrix = \begin{array}{cc} & \begin{array}{cc} +1 & -1 \end{array} \\ \begin{array}{c} +1 \\ -1 \end{array} & \begin{bmatrix} TP & FN \\ FP & TN \end{bmatrix} \end{array}$$

Besides these metrics, we also used additional ones in order to understand how much the computational overhead due to the use of cryptography influences the system. We compared the results obtained by the privacy-preserving versions of ML algorithms with the ones obtained using the baseline.S We also take into account the execution times of the system, which shows the overhead caused by the additional computational cost added by cryptography. Finally, we also show the increase in communication costs that happen when cryptography is involved, as all values must be represented by ciphertext, instead of integer or float values.

5.2 Experimental Setup

All the experiments were performed using a machine with an Inter Core i5-4300M CPU @2.60Gz with a 3MB L3 cache memory and a 12 GB RAM memory.

For obtaining the experimental results, we started by applying the pre-processing techniques mentioning in 4.2 in the datasets described in table 4.1. As mentioned, all datasets that were composed by a single file were split into three sets, training, validation and testing sets, with the proportion 70/15/15. Each ML model was trained using the training set, the best model configuration was chosen using the validation set, and the model performance was evaluated using the testing set.

5.2.1 Parameters - Baseline

For the experiments with Decision Trees (DT), we tested the values for *max_depth* of 5%, 10%, 20%, 50%, 100%, 200%, and 500% of the total number of features, and the values for *min_samples_leaf* of 0.001%, 0.002%, 0.005%, 0.01%, 0.02%, 0.05%, 0.1%, 0.2%, 0.5%, 1%, 2% and 5% of the total number of training samples.

For the experiments with Support Vector Machines (SVM), we used *kernel* values of *linear*, *poly* and Radial Basis Function (RBF). For all kernels, we used C values of 2^{-10}, 2^{-6}, 2^{-2}, 2^2, 2^6 and 2^{10}. For the polynomial kernel, we used *degree* values of 2, 3 and 4. For the RBF kernel, we used γ values of 2^{-9}, 2^{-5}, 2^{-1}, 2^1 and 2^3.

For the experiments with k-Means (k-M), we tested with a variable number of clusters, *i.e.*, with *num_clusters* values of 2, 3, 4, 5, 6, 7, 8, 9, 10, 20, 30, 40, 50, 60, 70, 80, 90 and 100.

For the experiments with Logistic Regression (LR), we used the *liblinear* solver with C values of 2^{-10}, 2^{-6}, 2^{-2}, 2^2, 2^6 and 2^{10}.

These variations on parameters allowed to train models with all possible configurations without the need to specifically adapt the parameters to the different datasets.

5.2.2   Parameters - Garbled Circuits

In the experiments using Garbled Circuits (GC), we tested values of 8, 12, 16, 20 and 24 bits for the numeric precision of the data and model parameters. This will be reflected in the circuit size and the accuracy of the results. Larger values were not considered because 24 bits is already sufficient for an exact representation of the input values and model parameters.

5.2.3   Parameters - Homomorphic Encryption

For the experiments using Partial Homomorphic Encryption (PHE), we used the values of 128, 256, 512, 1024 and 2048 bits for the length of the cryptographic keys.

For the experiments using Fully Homomorphic Encryption (FHE), no parameter search was made, since adequate default values were pre-determined for most parameters, and we chose a large enough number for the modulus ($(2^{15})^4 = 2^{60}$). Changing these parameters should not affect the obtained predicted labels in any way, although they may have minimal effects on the actual output of the evaluation functions of LR and SVM algorithms. The values considered are too large to affect the results and correspond to a cryptographic security factor.

5.2.4 Garbled Circuits toolkits

For the experiments with GC, we used the toolkit developed by VIPP group from the University of Siena[1]. This toolkit was not our first choice, since it has known issues in computation times, but the other toolkits that we tested contained limitations that we could not overcome, as stated below:

- **ABY**[13]: We found it impossible to define gate-to-gate wires, and that removed the ability for fine control on how to use and combine wires.

- **JustGarble**[2]: This toolkit could not be fully compiled due to conflicts with current versions of the GNU gcc compiler.

- **Ciphermed**[6]: This toolkit is efficient for small DT, but is exponentially slower for larger trees (above 10 nodes).

- **TinyGarble**[42]: The current version of this toolkit does not support the open source synthesis tool (Yosis[3]) recommended by the authors, and only supports a paid one.

- **CompGC**[21]: The implementation of all the examples of ML algorithms in this toolkit are hardcoded, making it extremely difficult to adapt to our needs.

5.2.5 Homomorphic Encryption toolkits

For the experiments using PHE, we implemented our own version of the Paillier cryptosystem [33]. We decided to do this way instead of using an existing toolkit because it was both rather simple to implement and also because this provided a good learning experience on the inner workings of a cryptographic system.

---

[1] http://clem.dii.unisi.it/~vipp/index.php/home
[2] https://github.com/irdan/justGarble
[3] http://www.clifford.at/yosys/

For the experiments using FHE, we used the HElib toolkit [23]. Two methods were considered in the implementation, method 1 (M1: multiply and sum arrays without packing) and method 2 (M2: use coefficient packing, invert and multiply polynomials), that were inspired in a HElib tutorial[4]. It is also important to note that, due to intrinsic limitations in HElib, we had to pre-compute $\alpha_i \cdot x_{SV}^{(i)}$ when evaluating the SVM. However, this does not negatively affect the experiments, since these model parameters are both owned by the same party.

5.3   Experimental Results - Baseline

We now show the experimental results obtained by applying different ML algorithms to the datasets mentioned above. We show in each subsection below, corresponding to each dataset, the different ML algorithms and parameters, and mention results found in the literature. Each result that we show in the following tables are the ones obtained by the best combination of parameters, *i.e.*, the parameters that provided the best accuracy or F-Measure results with the validation set.

5.3.1   Breast Cancer Wisconsin Dataset

We present the best baseline results obtained in the testing set for the Breast Cancer Wisconsin Dataset in table 5.2.

Table 5.2: Baseline results for Breast Cancer Wisconsin Dataset. "A" represents Accuracy, "F" represents F-Measure.

ML algorithm	DT	k-Means	LR	SVM		
				Linear	Poly	RBF
Baseline	A: 92.94%	A: 91.76%	A: 95.29%	A: 94.12%	A: 94.12%	A: 94.12%
	F: 90.91%	F: 90.91%	F: 93.75%	F: 92.06%	F: 92.31%	F: 92.06%
Literature	A: 95.13%	A: 92.79%	A: 93.50%	A: -	A: 97.54%	A: 97.13%
	F: 94.88%	F: -	F: -	F: -	F: -	F: 96.25%

---

[4]https://mshcruz.wordpress.com/2016/09/27/scalar-product-using-helib/

As we see, our baseline results are near the ones found in the literature.

5.3.2 Pima Indians Diabetes Dataset

We present the best baseline results obtained in the testing set for the Pima Indians Diabetes Dataset in table 5.3.

Table 5.3: Baseline results for Pima Indians Diabetes Dataset. "A" represents Accuracy, "F" represents F-Measure.

ML algorithm	DT	k-Means	LR	SVM		
				Linear	Poly	RBF
Baseline	A: 73.04% F: 63.53%	A: 72.17% F: 52.94%	A: 75.65% F: 58.82%	A: 75.65% F: 61.11%	A: 76.52% F: 59.70%	A: 77.39% F: 62.86%
Literature	A: 75.39% F: -	A: 73.7 F: -	A: 77.95% F: -	A: - F: -	A: -% F: -	A: 80.2% F: -

We can see that our results are comparable to the ones found in the literature.

5.3.3 Credit Approval Dataset

We present the best baseline results obtained in the testing set for the Credit Approval Dataset in table 5.4.

Table 5.4: Baseline results for Credit Approval Dataset. "A" represents Accuracy, "F" represents F-Measure.

ML algorithm	DT	k-Means	LR	SVM		
				Linear	Poly	RBF
Baseline	A: 78.64% F: 75.00%	A: 83.50% F: 81.32%	A: 85.44% F: 84.21%	A: 85.44% F: 84.54%	A: 85.43% F: 83.87%	A: 84.47% F: 83.33%
Literature	A: 85.5% F: -	A: 86.3 F: -	A: 87.9% F: -	A: 86.2 F: -	A: 84.8% F: -	A: 85.5% F: -

We obtained results that are similar to the ones found in the literature.

5.3.4 Adult Income Dataset

We present the best baseline results obtained in the testing set for the Adult Income Dataset in table 5.5.

Table 5.5: Baseline results for Adult Income Dataset. "A" represents Accuracy, "F" represents F-Measure.

ML algorithm	DT	k-Means	LR	SVM		
				Linear	Poly	RBF
Baseline	A: 85.56%	A: 81.95%	A: 85.08%	A: 69.67%	A: 80.82%	A: 82.79%
	F: 67.37%	F: 55.80%	F: 66.87%	F: 57.04%	F: 65.91%	F: 61.15%
Literature	A: 82.20%	A: -	A: 80.00%	A: -	A: 84.55%	A: 84.93%
	F: -	F: -	F: -	F: -	F: -	F: -

We can see that our baseline results are comparable to the ones found in literature, and therefore are a good reference for the privacy-preserving platform.

5.4 Experimental Results - Comparison with the Baseline

> **Missing the references for literature in these tables. look for better results.**

It is important to mention that, after analyzing the results obtained using the toolkit, we verified that changing the amount of bits for the actual numeric precision of the data and model parameters affects the accuracy of the results. The degree of this error is depicted in table 5.6 and 5.7, for the experiments for DT and k-M respectively. It is to be noted that this error is computed versus the baseline prediction results, not the prediction labels from the dataset.

Table 5.6: Average prediction error for DT when compared with the baseline.

bits	Pima Indians	Breast Cancer	Credit Approval	Adult Income
8	1.88%	0.55%	8.70%	0.00%
12	0.00%	0.13%	1.11%	0.00%
16	0.00%	0.13%	0.31%	0.00%
20	0.00%	0.13%	0.31%	0.00%
24	0.00%	0.13%	0.31%	0.00%

Table 5.7: Average prediction error for k-M when compared with the baseline.

bits	Pima Indians	Breast Cancer	Credit Approval	Adult Income
8	2.03%	3.07%	0.05%	0.02%
12	0.39%	0.85%	0.00%	0.00%
16	0.29%	0.72%	0.00%	0.00%
20	0.29%	0.72%	0.00%	0.00%
24	0.00%	0.00%	0.00%	0.00%

By observing these tables, we can conclude that the loss of prediction performance caused by using the privacy-preserving versions of the ML algorithms is not relevant, as long as at least 16 bits are used to represent the data. Since both DT and k-M only output an integer representing the label, and not a real number, the visible effect of changing the amount of bits is minimal.

After analyzing the results obtained using the PHE and FHE systems, we verified that all predicted labels and almost all function evaluation outputs match the baseline. The few examples when an exact match does not happen come mostly from the SVM scoring evaluation function implemented in HElib, and are most likely caused by accumulation of the intrinsic noise generated every time an operation is performed between two ciphertexts. Therefore, we can conclude that our privacy-preserving versions of the ML algorithms using PHE and FHE have no relevant loss of prediction performance.

5.5   Experimental Results - Execution Time

In order to better assess the execution time required by each privacy-preserving version of the different ML algorithms, we will analyze each of the combinations tested separately.

5.5.1   Garbled Circuits and Decision Trees

We present the execution times obtained in using the toolkit to build a GC implementation of DT for all the datasets in the tables below. The results are presented in terms of average pre-computation times per data sample and runtimes per data sample. Table 5.8 presents the average pre-computation times for each data sample for all datasets.

Table 5.8: Average pre-computation times per data sample, in seconds.

dataset	numeric precision				
	8 bits	12 bits	16 bits	20 bits	24 bits
Breast	0.205	0.240	0.281	0.325	0.356
Pima	0.219	0.285	0.310	0.344	0.356
Credit	0.224	0.253	0.271	0.290	0.315
Adult	0.233	0.271	0.313	0.355	0.373

In table 5.8 we can observe that average pre-computation times are very similar to one another, despite the slight dependence on the size of the GC, which is defined by the numeric precision. This means that pre-computation poses no restrictions regarding the scalability of our approach.

Table 5.9: Runtime per data sample, in seconds. Breast Cancer Wisconsin Diagnostic dataset.

DT depth	numeric precision				
	8 bits	12 bits	16 bits	20 bits	24 bits
1	1.015	1.383	1.735	2.082	2.442
3	1.057	1.425	1.804	2.187	2.550
4	1.107	1.533	1.920	2.319	2.724
5	1.307	1.713	2.122	2.665	2.974
6	1.538	2.030	2.522	2.939	3.330
7	1.966	2.393	2.823	3.507	3.787

Table 5.10: Runtime per data sample, in seconds. Pima Indians Diabetes dataset.

DT depth	numeric precision				
	8 bits	12 bits	16 bits	20 bits	24 bits
1	0.374	0.502	0.663	0.782	0.901
4	0.457	0.548	0.677	0.888	1.087
6	0.654	0.877	1.051	1.177	1.195
8	1.622	1.689	1.889	2.099	2.180
10	3.469	3.644	3.112	4.300	4.343
12	7.884	9.727	12.459	16.270	17.488
13	16.289	18.353	20.926	25.120	33.508

Table 5.11: Runtime per data sample, in seconds. Credit Approval dataset.

DT depth	numeric precision				
	8 bits	12 bits	16 bits	20 bits	24 bits
2	1.598	2.333	2.956	3.591	4.190
5	1.852	2.603	3.295	3.953	4.576
7	2.702	3.600	4.359	5.743	5.848
8	3.365	4.280	6.541	7.337	8.998
9	4.511	6.930	8.202	10.990	12.227

Table 5.12: Runtime per data sample, in seconds. Adult Income dataset.

DT depth	numeric precision				
	8 bits	12 bits	16 bits	20 bits	24 bits
5	3.777	5.169	6.502	8.171	9.424
6	4.111	5.774	7.074	8.813	10.816
9	9.256	13.864	20.294	24.314	27.236
10	17.167	23.056	30.553	39.114	54.987
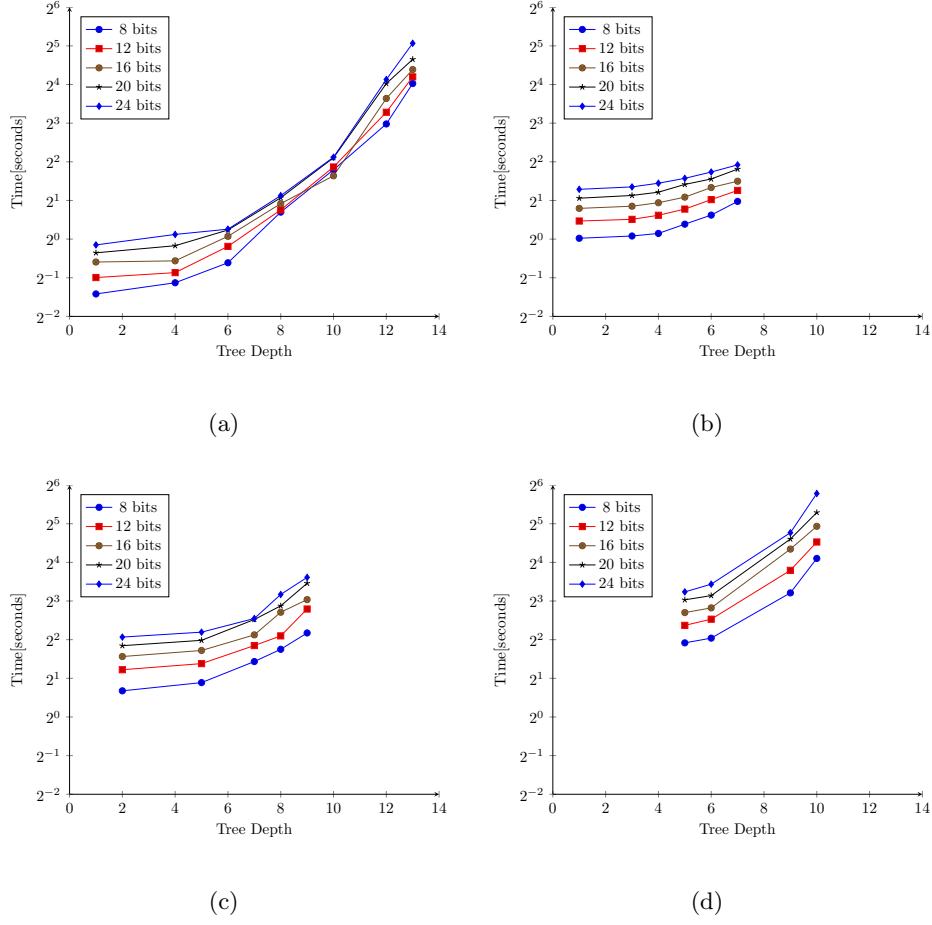
Figure 5.1: Runtime per data sample, in seconds. (a) Pima Indians Diabetes Dataset; (b) Breast Cancer Wisconsin Diagnostic Dataset; (c) Credit Approval Dataset; and, (d) Adult Income Dataset.

Regarding the runtimes per data sample (tables 5.10, 5.9, 5.11, 5.12, and figure 5.1), we can observe that the results obtained eclipse the pre-computation times. Although the scale slightly sub-linearly with the numeric precision and the number of features, they scale super-linearly with the number of nodes in the DT. However, this does not compromise the scalability of our approach, as increasing DT depth only leads to an exponential increase in the number of nodes when fully expanded DT are considered.

5.5.2   Garbled Circuits and k-Means

We present the execution times obtained in using the toolkit to build a GC implementation of k-M for all the datasets in the tables below. The results are presented in terms of average pre-computation times per data sample and runtimes per data sample. Table 5.13 presents the average pre-computation times for each data sample for all datasets.

Table 5.13: Average pre-computation times per data sample, in seconds.

dataset	numeric precision				
	8 bits	**12 bits**	**16 bits**	**20 bits**	**24 bits**
Breast	0.225	0.251	0.274	0.289	0.301
Pima	0.260	0.283	0.307	0.331	0.339
Credit	0.226	0.249	0.253	0.265	0.272
Adult	0.214	0.214	0.232	0.266	0.262

We can see in table 5.13 that average pre-computation times are all very similar to one another despite the slight dependence on the GC size, meaning that it does not impact the scalability of our solution.

Table 5.14: Runtime per data sample, in seconds. Breast Cancer Wisconsin Diagnostic dataset.

number clusters	numeric precision				
	8 bits	**12 bits**	**16 bits**	**20 bits**	**24 bits**
2	1.619	2.102	2.788	3.616	3.838
3	1.868	2.451	3.037	3.550	5.674
4	1.800	3.082	3.180	5.028	5.604
5	1.912	2.598	4.760	5.410	7.402
6	2.062	2.707	4.872	5.275	7.951
7	2.351	2.882	5.133	7.149	7.906
8	2.930	2.805	4.978	6.946	9.124
9	2.570	3.917	4.935	7.044	9.136
10	2.291	4.215	5.990	7.349	10.802
20	4.223	6.797	9.580	14.877	25.232
30	5.269	9.234	15.693	24.228	36.767
40	7.126	14.724	25.754	37.984	49.574
50	7.885	14.701	24.245	40.602	67.651
60	9.156	19.862	35.970	56.269	-
70	11.585	23.550	41.306	-	-
80	11.433	23.367	45.067	-	-
90	13.642	27.030	53.124	-	-
100	16.684	32.046	59.372	-	-

Table 5.15: Runtime per data sample, in seconds. Pima Indians Diabetes dataset.

number clusters	numeric precision				
	8 bits	12 bits	16 bits	20 bits	24 bits
2	0.629	0.848	1.163	1.350	1.533
3	0.813	1.055	1.270	1.494	1.761
4	0.852	1.143	1.429	1.664	1.969
5	0.943	1.246	1.531	1.824	2.066
6	1.140	1.261	1.513	2.266	2.145
7	1.124	1.319	1.620	2.319	2.188
8	1.178	1.339	1.697	2.130	2.337
9	1.227	1.464	2.356	2.178	2.504
10	1.228	1.544	2.439	2.497	3.717
20	1.568	1.896	3.374	3.952	5.751
30	2.142	2.223	3.794	5.315	7.346
40	1.762	3.379	5.099	6.941	9.697
50	1.803	3.670	5.979	8.300	12.376
60	3.005	4.927	7.025	12.218	16.839
70	3.420	5.250	7.616	15.795	19.339
80	3.513	6.187	8.649	15.795	21.302
90	3.490	6.239	12.278	16.691	23.669
100	3.580	7.876	14.497	23.599	32.248

Table 5.16: Runtime per data sample, in seconds. Credit Approval dataset.

number clusters	numeric precision				
	8 bits	12 bits	16 bits	20 bits	24 bits
2	2.541	3.652	4.151	5.729	6.837
3	2.615	3.752	5.547	6.369	8.512
4	2.902	3.663	5.934	8.114	9.692
5	3.016	5.036	5.825	8.633	10.725
6	3.129	5.359	7.474	9.820	12.961
7	3.058	5.216	7.463	10.031	16.429
8	3.033	5.360	7.912	11.342	15.633
9	3.082	5.326	8.911	13.911	19.112
10	4.196	6.569	9.412	14.731	18.997
20	6.368	10.641	19.121	28.797	44.666
30	9.060	17.660	28.409	49.931	-
40	11.152	25.673	42.835	-	-
50	14.681	27.539	54.795	-	-
60	18.326	37.088	-	-	-
70	19.318	41.164	-	-	-
80	23.371	56.422	-	-	-
90	27.361	62.862	-	-	-
100	27.908	67.835	-	-	-

Table 5.17: Runtime per data sample, in seconds. Adult Income dataset.

number clusters	numeric precision				
	8 bits	12 bits	16 bits	20 bits	24 bits
2	4.703	5.536	8.098	11.307	13.672
3	4.361	7.535	9.620	14.500	18.896
4	4.396	6.986	10.364	15.635	20.343
5	6.083	8.688	11.681	17.928	25.292
6	5.587	11.307	15.884	23.006	31.464
7	6.441	10.278	15.981	24.306	31.277
8	6.394	10.871	17.157	27.155	38.732
9	8.029	12.958	21.050	32.607	48.497
10	8.706	14.956	24.468	32.010	47.241
20	14.834	24.567	41.386	-	-
30	19.150	39.870	-	-	-
40	27.248	60.023	-	-	-
50	33.243	-	-	-	-
60	42.933	-	-	-	-
70	52.514	-	-	-	-
80	61.952	-	-	-	-
90	-	-	-	-	-
100	-	-	-	-	-
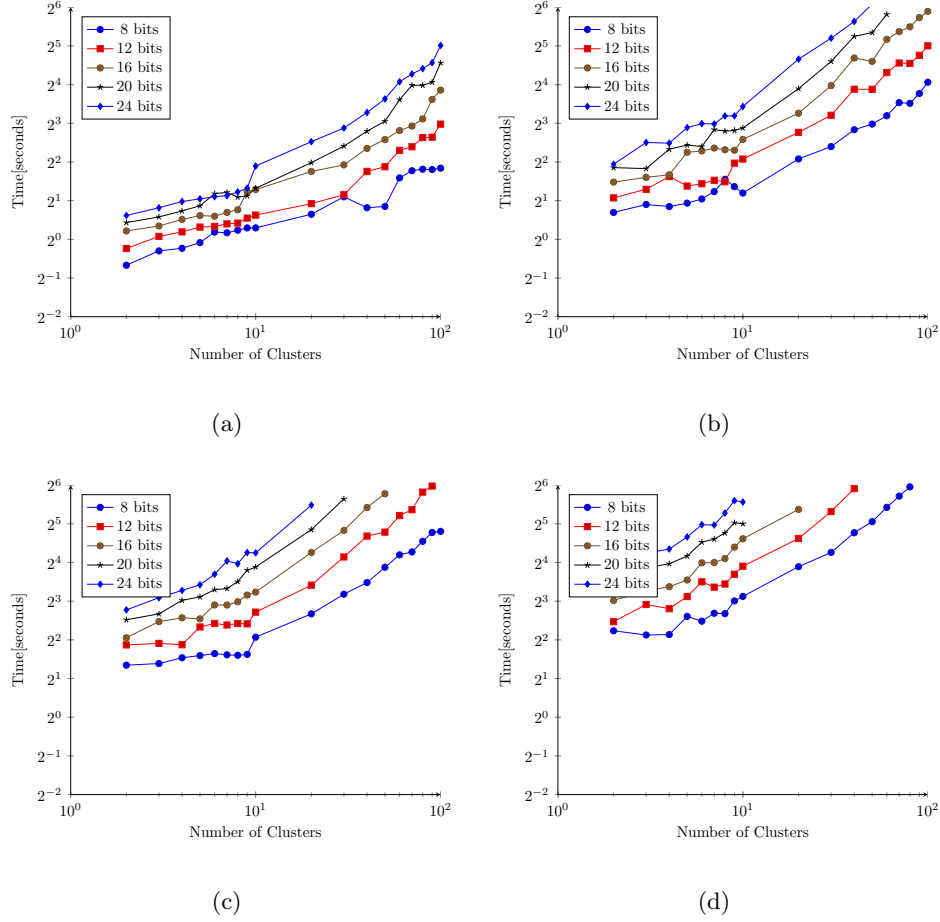
Figure 5.2: Runtime per data sample, in seconds. (a) Pima Indians Diabetes Dataset; (b) Breast Cancer Wisconsin Diagnostic Dataset; (c) Credit Approval Dataset; and, (d) Adult Income Dataset.

Regarding the runtimes per data sample (tables 5.15, 5.14, 5.16, 5.17, and figure 5.2), we observe again that the results are much larger that the pre-computation times. They scale linearly with the number of features and slightly super-linearly with the number of clusters, none of which compromises the scalability of our approach. However, runtimes scale quadratically with the numeric precision, which is caused by the multiplications required to compute the Euclidean distances. Although this causes scalability issues for larges values of numeric precision, the results in table 5.7 showed that the loss of accuracy is negligible even when only 12 bits are considered, allowing us to safely ignore this issue. The runtimes per data

sample are also considerably large for the instances with a large numbber of clusters, but in our baseline system we verified that the best results were always obtained when less than 10 clusters were considered. Even if this was not the case, we could sacrifice a bit of accuracy by lowering the amount of clusters considered in order to obtain much faster runtimes.

5.5.3   Homomorphic Encryption and Logistic Regression

We present the execution times obtained in using the PHE and FHE systems for all datasets in the tables bellow. For the FHE system, we present the times for methods 1 (M1) and 2 (M2) in the same cell, for ease of comparison. The results are presented in terms of total execution time for the whole datasets and execution time per data sample.

Table 5.18: LR+PHE. Pima Indians Diabetes Dataset. Execution time in seconds.

NBits	Total execution time (seconds)			Execution time / data sample (seconds)		
	encryption	computation	decryption	encryption	computation	decryption
128	0.224	0.088	0.023	0.002	0.001	0.000
256	0.528	0.167	0.051	0.005	0.001	0.000
512	1.563	0.329	0.189	0.014	0.003	0.002
1024	4.010	0.482	0.595	0.035	0.004	0.005
2048	19.330	1.569	2.828	0.168	0.014	0.025

Table 5.19: LR+PHE. Breast Cancer Wisconsin Dataset. Execution time in seconds.

NBits	Total execution time (seconds)			Execution time / data sample (seconds)		
	encryption	computation	decryption	encryption	computation	decryption
128	0.466	0.164	0.010	0.005	0.002	0.000
256	1.137	0.345	0.035	0.013	0.004	0.000
512	2.020	0.434	0.082	0.024	0.005	0.001
1024	8.467	1.491	0.434	0.100	0.018	0.005
2048	48.454	2.410	1.716	0.570	0.028	0.020

Table 5.20: LR+PHE. Credit Approval Dataset. Execution time in seconds.

NBits	Total execution time (seconds)			Execution time / data sample (seconds)		
	encryption	computation	decryption	encryption	computation	decryption
128	0.825	0.388	0.012	0.008	0.004	0.000
256	1.253	0.636	0.025	0.012	0.006	0.000
512	3.614	1.299	0.084	0.035	0.013	0.001
1024	17.526	2.707	0.393	0.170	0.026	0.004
2048	99.917	4.598	1.890	0.970	0.045	0.018

Table 5.21: LR+PHE. Adult Income Dataset. Execution time in seconds.

NBits	Total execution time (seconds)			Execution time / data sample (seconds)		
	encryption	computation	decryption	encryption	computation	decryption
128	78.530	59.299	0.667	0.011	0.008	0.000
256	149.288	115.984	1.479	0.020	0.016	0.000
512	512.082	171.196	4.963	0.070	0.023	0.001
1024	2446.53	423.473	22.342	0.334	0.058	0.003
2048	15430.6	907.304	139.051	2.106	0.124	0.019
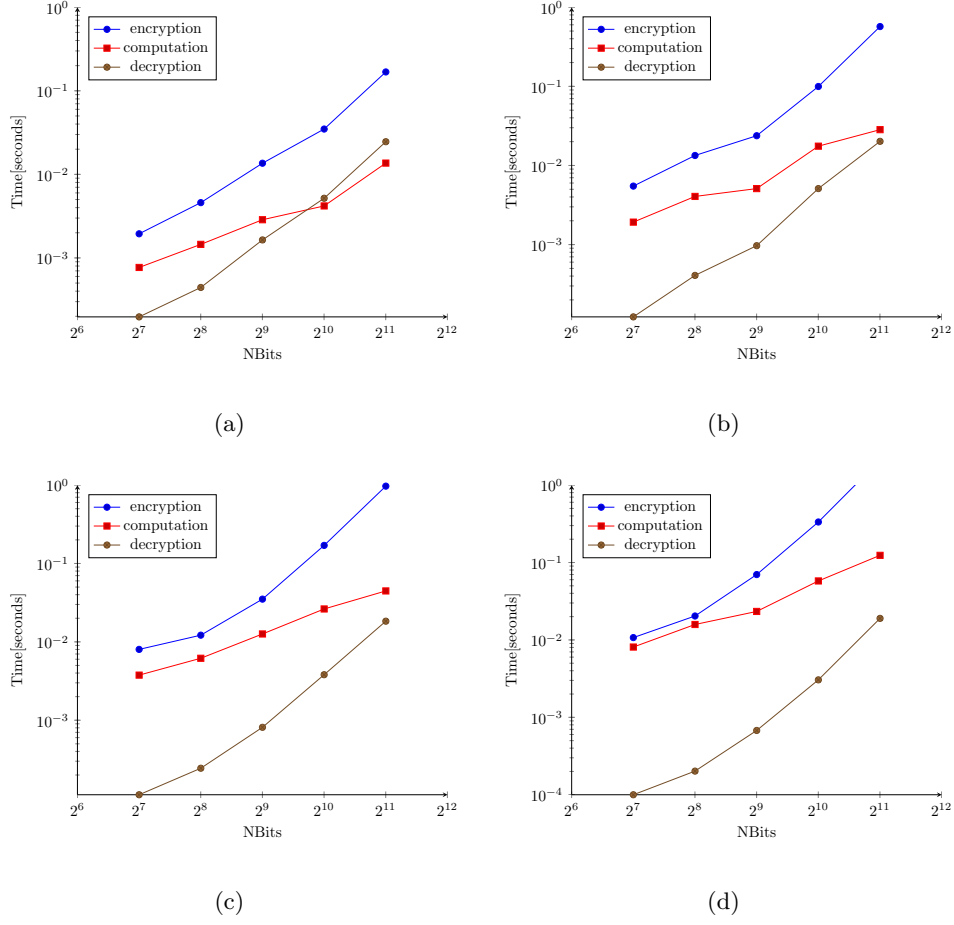
Figure 5.3: Execution time per data sample, in seconds. (a) Pima Indians Diabetes Dataset; (b) Breast Cancer Wisconsin Diagnostic Dataset; (c) Credit Approval Dataset; and, (d) Adult Income Dataset.

When observing the execution times obtained using PHE (tables 5.18, 5.19, 5.20, 5.21, and figure 5.3), we see that a linear increase in encryption and computation times when the number of features in the samples increases, but a constant decryption time, independent of the number of features. We can also observe a linear increase in computation times, and a super-linear increase in encryption and decryption times, when the value of $NBits$ (key length) increases. This can cause a problem of scalability, but it can be safely ignored since the execution times per sample are still very small.

Table 5.22: LR+FHE. All Datasets. Execution time in seconds.

dataset	Total execution time (seconds)			Execution time / data sample (seconds)		
	encryption	computation	decryption	encryption	computation	decryption
Pima	M1: 252.211	M1: 5.657	M1: 4.087	M1: 2.193	M1: 0.049	M1: 0.036
	M2: 19.303	M2: 0.630	M2: 4.116	M2: 0.168	M2: 0.005	M2: 0.036
Breast	M1: 636.954	M1: 14.352	M1: 3.032	M1: 7.494	M1: 0.169	M1: 0.036
	M2: 14.254	M2: 0.460	M2: 3.021	M2: 0.168	M2: 0.005	M2: 0.036
Credit	M1: 1284.14	M1: 30.441	M1: 3.921	M1: 12.467	M1: 0.296	M1: 0.038
	M2: 18.500	M2: 0.581	M2: 3.951	M2: 0.180	M2: 0.006	M2: 0.038
Adult	M1: 177675	M1: 4322.36	M1: 262.462	M1: 24.253	M1: 0.590	M1: 0.036
	M2: 1235.73	M2: 39.835	M2: 260.562	M2: 0.169	M2: 0.005	M2: 0.036
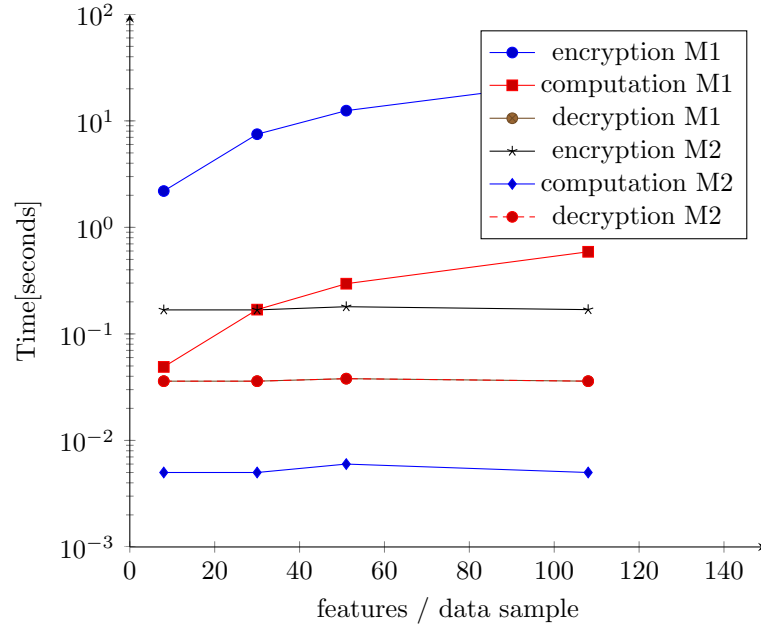
Figure 5.4: All Datasets. Execution time per data sample, in seconds.

When analyzing the results obtained using FHE (table 5.22 and figure 5.5.3), we can observe that the packing used by method M2 greatly decreases the encryption and computation times,

when compared to method M1. We also observe that method M2 makes the encryption and computation times independent from the number of features. Overall, we see that method M2 is much more efficient that method M1, showing the obvious advantage of packing packing the features for each dataset in a single ciphertext before performing any computation.

When comparing PHE and FHE, we observe that method M2 of FHE has lower execution times than PHE, despite the complexity of the algorithm behind it. The only possible justification for this is the positive effect cause by feature packing, especially due to the gains in encryption time.

5.5.4 Homomorphic Encryption and Support Vector Machines

We present the execution times obtained for using the PHE and FHE systems for all datasets in the tables below. For the FHE system, we present the times for methods 1 (M1) and 2 (M2) in the same cell, for ease of comparison. The results are presented in terms of total execution time for the whole datasets and execution time per data sample for the PHE case, and execution time per sample and support vector for the FHE case.

Table 5.23: SVM+PHE. Pima Indians Diabetes Dataset. Execution time in seconds.

dataset	Total execution time (seconds)			Execution time / data sample (seconds)		
	encryption	computation	decryption	encryption	computation	decryption
128	0.080	3.783	0.006	0.001	0.033	0.000
256	0.143	6.207	0.015	0.001	0.054	0.000
512	0.537	10.987	0.064	0.005	0.096	0.001
1024	2.787	25.419	0.352	0.024	0.221	0.003
2048	21.330	85.946	2.835	0.185	0.747	0.025

Table 5.24: SVM+PHE. Breast Cancer Wisconsin Diagnostic Dataset. Execution time in seconds.

dataset	Total execution time (seconds)			Execution time / data sample (seconds)		
	encryption	computation	decryption	encryption	computation	decryption
128	0.178	2.182	0.004	0.002	0.026	0.000
256	0.408	3.661	0.012	0.005	0.043	0.000
512	1.556	7.235	0.050	0.018	0.085	0.001
1024	7.872	18.094	0.258	0.093	0.213	0.003
2048	48.600	53.822	1.598	0.572	0.663	0.019

Table 5.25: SVM+PHE. Credit Approval Dataset. Execution time in seconds.

dataset	Total execution time (seconds)			Execution time / data sample (seconds)		
	encryption	computation	decryption	encryption	computation	decryption
128	0.347	2.637	0.005	0.003	0.026	0.000
256	0.783	4.371	0.015	0.008	0.042	0.000
512	3.117	8.703	0.059	0.030	0.084	0.001
1024	16.010	21.413	0.314	0.155	0.208	0.003
2048	99.340	61.076	1.927	0.964	0.593	0.019

Table 5.26: SVM+PHE. Adult Income Dataset. Execution time in seconds.

dataset	Total execution time (seconds)			Execution time / data sample (seconds)		
	encryption	computation	decryption	encryption	computation	decryption
128	48.270	2322.71	0.372	0.007	0.317	0.000
256	118.029	3603.13	1.020	0.016	0.492	0.000
512	484.839	7702.16	4.363	0.066	1.051	0.001
1024	2522.01	18342.4	22.714	0.344	2.504	0.003
2048	14920.8	51150.3	138.389	2.057	6.982	0.019
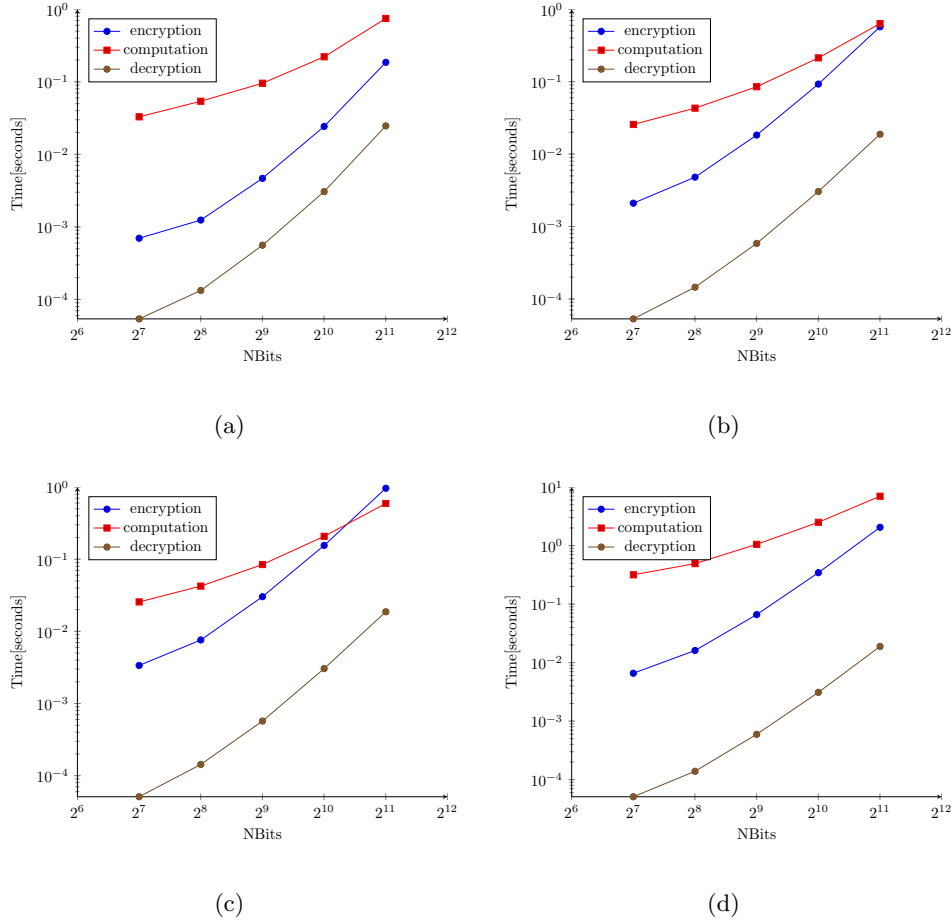
Figure 5.5: Execution time per data sample, in seconds. (a) Pima Indians Diabetes Dataset; (b) Breast Cancer Wisconsin Diagnostic Dataset; (c) Credit Approval Dataset; and, (d) Adult Income Dataset.

When observing the execution times obtained using PHE (tables 5.23, 5.24, 5.25, 5.26, and figure 5.5), we can see that computation times have a significant overhead for small amounts of number of features and number of Support Vectors, as they only affect the results in the Adult Income Dataset, where they seem to have a linear dependency. We also observe a linear increase in encryption times with increasing the number of features, and a linear increase in computation times with increasing the key length ($NBits$). Finally, we can see that decryption times are constant with increasing number of features and number of Support Vectors, but we verify that the is a slightly super-linear increase in encryption and decryption

times with increasing $NBits$. This can cause a problem of scalability, but it can be safely ignored since the execution times per sample are still very small.

Additionally, for SVM we obtained similar encryption times to the ones obtained for LR, which is not surprising since the encryption is only done in one side of the protocol. Also, we can observe that decryption times are comparable in both cases, as decryption only occurs once, when all operations have been performed. Additionally, for SVM we obtained similar encryption times to the ones obtained for LR, which is not surprising since only Alice's data needs to be encrypted (due to the way the Paillier cryptosystem works). Also, the decryption times are also comparable in both cases, as decryption only occurs once (when all operations have been performed).

Table 5.27: SVM+FHE. All Datasets. Execution time in seconds.

dataset	Total execution time (seconds)			Execution time / data sample (seconds)		
	encryption	computation	decryption	encryption/SV	computation/SV	decryption
Pima	M1: 39253.9	M1: 3583.28	M1: 6.652	M1: 1.127	M1: 0.104	M1: 0.058
	M2: 8038.90	M2: 6884.06	M2: 6.035	M2: 0.231	M2: 0.199	M2: 0.052
Breast	M1: 35625.5	M1: 3143.37	M1: 4.900	M1: 4.100	M1: 0.359	M1: 0.058
	M2: 2072.25	M2: 1751.68	M2: 4.503	M2: 0.233	M2: 0.200	M2: 0.053
Credit	M1: 113455	M1: 11714.2	M1: 6.120	M1: 5.892	M1: 0.615	M1: 0.059
	M2: 4512.37	M2: 3838.78	M2: 5.472	M2: 0.235	M2: 0.201	M2: 0.053
Adult	M1: N/A	M1: N/A	M1: N/A	M1: 11.895	M1: 1.296	M1: 0.057
	M2: N/A	M2: N/A	M2: N/A	M2: 0.232	M2: 0.199	M2: 0.053
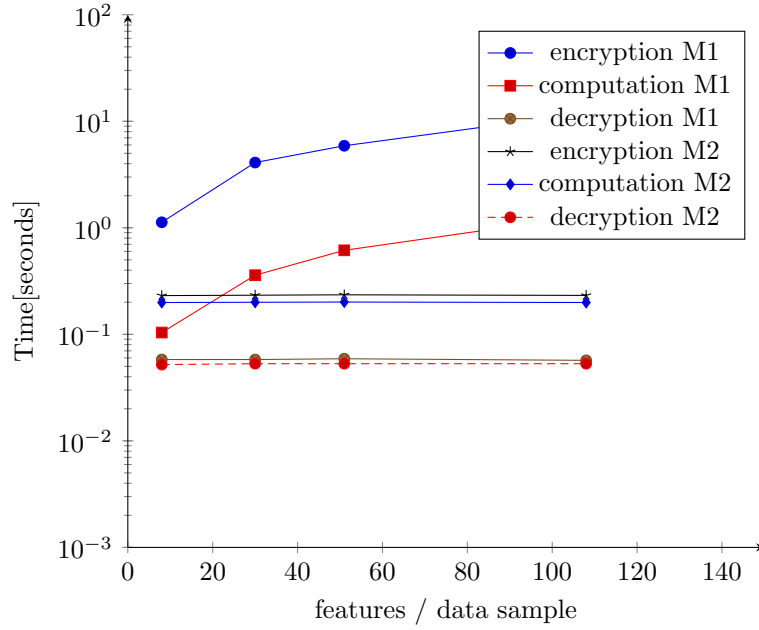
Figure 5.6: All Datasets. Execution time per data sample, in seconds.

When observing the execution times obtained using PHE (table 5.27 and figure 5.5.4), we can see similar results to those obtained for LR. In particular, we see that the packing used by method M2 greatly decreases the encryption and computation times, when compared with method M1, as well as making the encryption and computation times independent from both the number of features and the number of Support Vectors. We can also observe that the decryption times are independent on the method used, the number of features and the number of Support Vectors.

Once again, we can see that the method M2 is much more efficient than method M1, showing the obvious advantage of packing the features for each dataset in a single ciphertext before performing the computation.

When comparing PHE and FHE, unlike what was observed for LR, here the former has lower execution times that the latter. Even considering the feature packing of method M2, the fact that still many multiplications have to be made (one for each Support Vector) overwhelms FHE when compared with PHE for evaluating an SVM.

5.6   Experimental Results - Communication Cost

Given that the communication cost is primarily defined by the cryptographic techniques considered and only secondarily by the ML algorithms, we will focus on the former and address each of the specifics of the latter as needed. Given that cryptographic keys only need to be sent once and the ciphertext containing the desired result also only needs to be sent once, the bulk of the communication cost comes from the transmitting and receiving ciphertexts containing the actual data values. Since the amount of bytes sent by one of the parties is equal to the amount received by the other, and vice-versa, we will only present costs from one of the parties.

5.6.1   Garbled Circuits and Decision Trees

We present the communication costs obtained for using a GC implementation of DT for all datasets in the tables below. We present results in terms of average amount of bytes per data sample sent during pre-computation, received during pre-computation and sent during runtime, and the amount of bytes per data sample received during runtime by the GC evaluator. We do not present total communication costs for the whole datasets because communication costs per sample are independent from the dataset size, and communication costs per sample are the expected costs in a real-life scenario where a large computer cluster is available and data samples are supplied in a continuous fashion.

Table 5.28: DT+GC. Average amount of bytes per data sample (in kB) sent during pre-computation (PC-S), received during pre-computation (PC-R) and sent during runtime (R-S) by the GC evaluator.

dataset	numeric precision				
	8 bits	**12 bits**	**16 bits**	**20 bits**	**24 bits**
Pima	PC-S: 1.055	PC-S: 1.583	PC-S: 2.110	PC-S: 2.638	PC-S: 3.166
	PC-R: 2.342	PC-R: 3.513	PC-R: 4.685	PC-R: 5.856	PC-R: 7.027
	R-S: 8.441	R-S: 12.662	R-S: 16.883	R-S: 21.104	R-S: 25.324
Breast	PC-S: 1.055	PC-S: 1.583	PC-S: 2.110	PC-S: 2.638	PC-S: 3.165
	PC-R: 2.342	PC-R: 3.514	PC-R: 4.685	PC-R: 5.856	PC-R: 7.027
	R-S: 31.656	R-S: 47.483	R-S: 63.311	R-S: 79.139	R-S: 94.967
Credit	PC-S: 1.055	PC-S: 1.583	PC-S: 2.110	PC-S: 2.638	PC-S: 3.166
	PC-R: 2.342	PC-R: 3.514	PC-R: 4.685	PC-R: 5.812	PC-R: 7.027
	R-S: 53.815	R-S: 80.722	R-S: 107.628	R-S: 134.536	R-S: 161.444
Adult	PC-S: 1.055	PC-S: 1.583	PC-S: 2.110	PC-S: 2.638	PC-S: 3.166
	PC-R: 2.342	PC-R: 3.514	PC-R: 4.685	PC-R: 5.855	PC-R: 7.027
	R-S: 113.961	R-S: 170.940	R-S: 227.919	R-S: 284.901	R-S: 341.881

Table 5.29: DT+GC. Pima Indians Diabetes Dataset. Amount of bytes per data sample (in kB) received during runtime by the GC evaluator.

DT depth	numeric precision				
	8 bits	**12 bits**	**16 bits**	**20 bits**	**24 bits**
1	23.542	34.941	46.339	57.743	69.137
4	90.858	131.485	172.120	212.741	253.369
6	321.667	462.496	603.318	744.170	885.012
8	1244.83	1786.52	2328.19	2869.91	3411.55
10	4937.70	7082.78	9227.74	11372.8	13517.9
12	19708.8	28267.3	36825.7	45384.2	53942.9
13	39403.7	56513.5	73623.3	90733.1	107843

Table 5.30: DT+GC. Breast Cancer Wisconsin Diagnostic Dataset. Amount of bytes per data sample (in kB) received during runtime by the GC evaluator.

DT depth	numeric precision				
	8 bits	**12 bits**	**16 bits**	**20 bits**	**24 bits**
1	85.280	127.547	169.814	212.084	254.348
3	175.738	261.128	346.551	431.980	517.377
4	296.296	439.239	582.210	725.153	868.094
5	537.460	795.483	1053.44	1311.47	1569.51
6	1019.83	1507.91	1996.08	2484.10	2972.24
7	1984.46	2932.86	3881.12	4829.36	5777.78

Table 5.31: DT+GC. Credit Approval Dataset. Amount of bytes per data sample (in kB) received during runtime by the GC evaluator.

DT	numeric precision				
depth	8 bits	12 bits	16 bits	20 bits	24 bits
2	193.908	289.759	385.628	481.480	577.331
5	889.564	1323.15	1756.74	2190.37	2623.99
7	3274.64	4866.18	6457.81	8049.41	9640.96
8	6454.73	9590.25	12725.9	15861.4	18997.0
9	12815.1	19038.6	25262.1	31485.6	37709.3

Table 5.32: DT+GC. Adult Income Dataset. Amount of bytes per data sample (in kB) received during runtime by the GC evaluator.

DT	numeric precision				
depth	8 bits	12 bits	16 bits	20 bits	24 bits
5	1843.72	2753.97	3664.21	4574.43	5484.64
6	3485.83	5205.30	6924.63	8643.95	10363.3
9	26476.2	39523.2	52569.9	65617.1	78663.8
10	52751.1	78743.8	104736	130729	156721
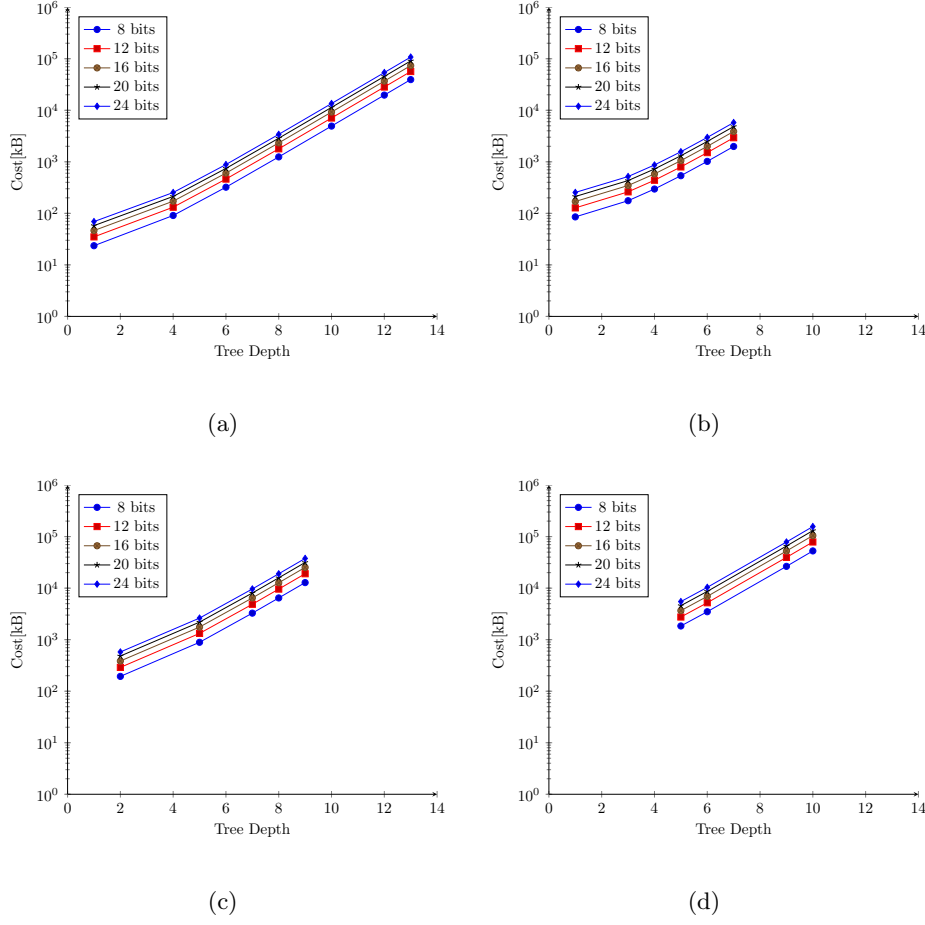
Figure 5.7: Amount of bytes per data sample (in kB) received during runtime by the GC evaluator. (a) Pima Indians Diabetes Dataset; (b) Breast Cancer Wisconsin Diagnostic Dataset; (c) Credit Approval Dataset; and, (d) Adult Income Dataset.

In table 5.28, we can see that all communication costs per data sample increase linearly with the variables of interest. Both the amount of bytes sent and received by the GC evaluator during pre-computation depend only on the numeric precision, and the amount of bytes sent during runtime depends only on the numeric precision and the number of features.

Regarding the amount of bytes per data sample received during runtime (tables 5.29, 5.30, 5.31, 5.32, and figure 5.7), they depend linearly on the numeric precision, the number of features and on the number of DT nodes, and therefore do not compromise the scalability of our approach. For larger DT, the communication cost gets considerable large, but can be

easily minimized by using the original DT instead of the fully expanded ones.

5.6.2   Garbled Circuits and *k*-Means

We present the communication costs obtained for using a GC implementation of k-M for all datasets in the tables below. We present results in terms of the average amount of bytes per data sample sent during pre-computation, received during pre-computation and sent during runtime, and the amount of bytes per data sample received during runtime by the GC evaluator. We do not present total communication costs for the whole datasets because communication costs per sample are independent from the dataset size, and communication costs per sample are the expected costs in a real-life scenario where a large computer cluster is available and data samples are supplied in a continuous fashion.

Table 5.33: k-M+GC. Average amount of bytes per data sample (in kB) sent during pre-computation (PC-S), received during pre-computation (PC-R) and sent during runtime (R-S) by the GC evaluator.

dataset	numeric precision				
	8 bits	**12 bits**	**16 bits**	**20 bits**	**24 bits**
Pima	PC-S: 1.055 PC-R: 2.342 R-S: 8.442	PC-S: 1.583 PC-R: 3.514 R-S: 12.662	PC-S: 2.110 PC-R: 4.685 R-S: 16.883	PC-S: 2.638 PC-R: 5.856 R-S: 21.104	PC-S: 3.166 PC-R: 7.027 R-S: 25.324
Breast	PC-S: 1.055 PC-R: 2.342 R-S: 31.656	PC-S: 1.583 PC-R: 3.514 R-S: 47.483	PC-S: 2.110 PC-R: 4.685 R-S: 63.311	PC-S: 2.638 PC-R: 5.856 R-S: 79.139	PC-S: 3.166 PC-R: 7.027 R-S: 94.967
Credit	PC-S: 1.055 PC-R: 2.342 R-S: 53.815	PC-S: 1.583 PC-R: 3.514 R-S: 80.722	PC-S: 2.110 PC-R: 4.685 R-S: 107.629	PC-S: 2.638 PC-R: 5.856 R-S: 134.536	PC-S: 3.166 PC-R: 7.027 R-S: 161.444
Adult	PC-S: 1.055 PC-R: 2.342 R-S: 113.960	PC-S: 1.583 PC-R: 3.514 R-S: 170.941	PC-S: 2.110 PC-R: 4.685 R-S: 227.921	PC-S: 2.638 PC-R: 5.856 R-S: 284.901	PC-S: 3.166 PC-R: 7.027 R-S: 341.881

Table 5.34: *k*-M+GC. Pima Indians Diabetes Dataset. Amount of bytes per data sample (in kB) received during runtime by the GC evaluator.

number clusters	numeric precision				
	8 bits	12 bits	16 bits	20 bits	24 bits
2	309.849	591.326	961.850	1421.45	1970.15
3	456.502	874.347	1425.81	2110.88	2929.57
4	603.113	1157.36	1889.79	2800.34	3889.04
5	749.726	1440.35	2353.72	3489.77	4848.50
6	896.376	1723.41	2817.74	4179.15	5807.92
7	1042.96	2006.43	3281.61	4868.58	6767.40
8	1189.58	2289.42	3745.53	5557.99	7726.80
9	1336.19	2572.47	4209.50	6247.40	8686.30
10	1482.85	2855.49	4673.53	6936.81	9645.61
20	2949.03	5685.65	9312.94	13831.1	19240.0
30	4415.20	8515.81	13952.4	20725.3	28834.3
40	5881.34	11345.9	18592.0	27619.6	38428.7
50	7347.55	14176.2	23231.6	34514.1	48023.0
60	8813.84	17006.3	27871.1	41408.0	57617.9
70	10280.0	19836.4	32510.5	48302.4	67211.9
80	11746.1	22665.5	37510.1	55196.6	76806.3
90	13212.4	25496.9	41789.7	62090.9	86400.4
100	14678.6	28326.8	46429.1	68985.2	95995.1

Table 5.35: k-M+GC. Breast Cancer Wisconsin Diagnostic Dataset. Amount of bytes per data sample (in kB) received during runtime by the GC evaluator.

number	numeric precision				
clusters	8 bits	12 bits	16 bits	20 bits	24 bits
2	1160.22	2215.08	3604.00	5326.97	7383.92
3	1706.36	3271.44	5337.51	7904.74	10973.1
4	2252.49	4327.76	7071.10	10482.5	14562.2
5	2798.68	5384.18	8804.74	13060.4	18151.1
6	3344.74	6440.50	10538.2	15638.3	21740.2
7	3890.90	7496.85	12271.9	18215.9	25329.3
8	4437.04	8553.14	14005.4	20793.9	28918.7
9	4983.15	9609.47	15739.1	23371.7	32507.7
10	5529.26	10666.0	17472.6	25949.5	36096.7
20	10990.4	21229.2	34808.3	51728.0	71987.3
30	16451.7	31792.6	52144.1	77506.3	107878
40	21913.0	42356.3	69479.9	103284	143769
50	27374.2	52919.6	86815.3	129062	179660
60	32835.5	63482.7	104151	154841	-
70	38296.9	74046.3	121487	-	-
80	43758.2	84609.9	138822	-	-
90	49219.1	95173.1	156159	-	-
100	54680.5	105737	173494	-	-

Table 5.36: k-M+GC. Credit Approval Dataset. Amount of bytes per data sample (in kB) received during runtime by the GC evaluator.

number	numeric precision				
clusters	8 bits	12 bits	16 bits	20 bits	24 bits
2	1972.05	3765.09	6126.07	9054.88	12551.6
3	2899.50	5559.67	9071.71	13435.4	18650.7
4	3827.13	7354.19	12017.0	17815.7	24750.1
5	4754.49	9148.68	14962.5	22196.1	30849.3
6	5682.00	10943.2	17908.1	26576.5	36948.4
7	6609.56	12737.8	20853.7	30956.8	43047.6
8	7536.99	14532.3	23799.2	35337.2	49146.9
9	8464.34	16326.9	26744.6	39717.9	55246.2
10	9391.91	18121.3	29690.1	44098.0	61345.4
20	18666.6	36066.6	59145.0	87901.8	122338
30	27941.7	54011.9	88600.0	131706	-
40	37216.4	71957.2	118055	-	-
50	46491.1	89902.4	147510	-	-
60	55765.9	107848	-	-	-
70	65040.7	125793	-	-	-
80	74315.8	143738	-	-	-
90	83590.5	161683	-	-	-
100	92865.1	179629	-	-	-

Table 5.37: k-M+GC. Adult Income Dataset. Amount of bytes per data sample (in kB) received during runtime by the GC evaluator.

| number	numeric precision				
clusters	8 bits	12 bits	16 bits	20 bits	24 bits
2	4174.78	7971.66	12971.1	19173.0	26577.5
3	6136.87	11769.3	19205.7	28445.8	39489.5
4	8098.94	15567.1	25440.1	37718.2	52401.5
5	10061.2	19364.8	31675.0	46990.8	65313.5
6	12023.2	23162.6	37909.2	56263.7	78225.4
7	13985.3	26960.3	44143.4	65536.3	91137.4
8	15947.4	30757.9	50378.0	74808.8	104049
9	17909.6	34555.5	56612.9	84081.3	116961
10	19871.7	38353.1	62847.2	93354.2	129873
20	39493.0	76330.2	125193	-	-
30	59114.1	114307	-	-	-
40	78735.3	152284	-	-	-
50	98356.6	-	-	-	-
60	117978	-	-	-	-
70	137598	-	-	-	-
80	157221	-	-	-	-
90	-	-	-	-	-
100	-	-	-	-	-
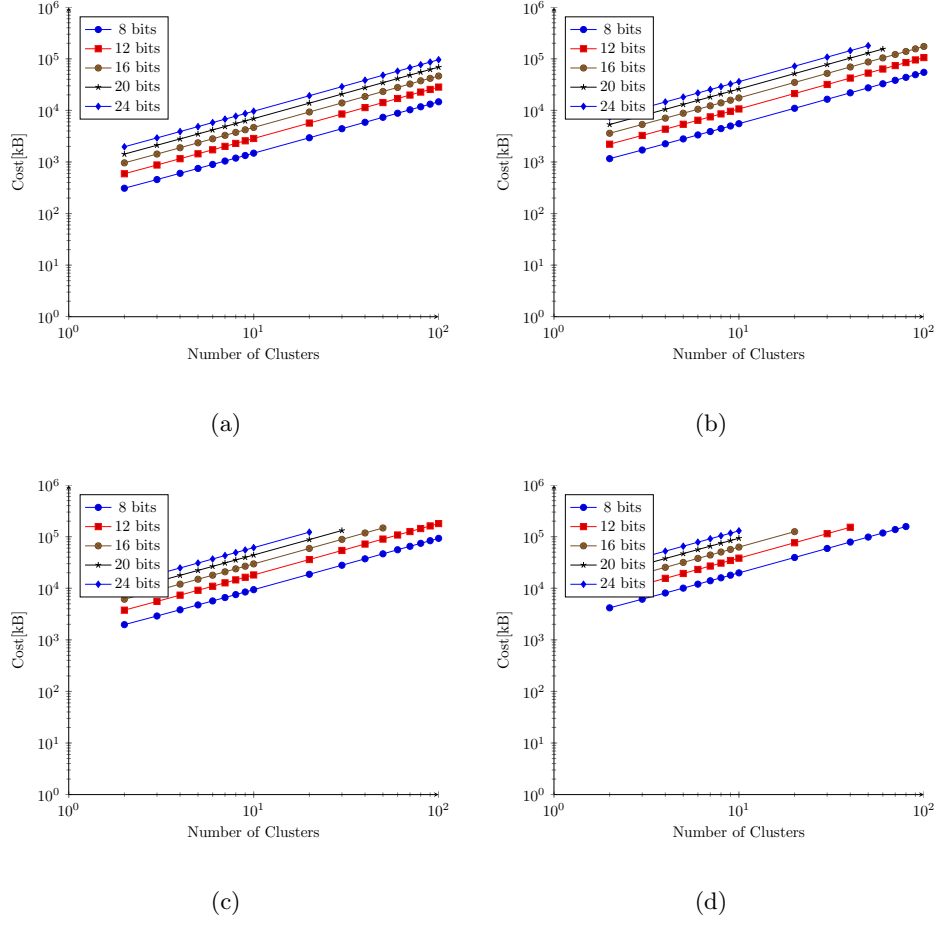
Figure 5.8: Amount of bytes per data sample (in kB) received during runtime by the GC evaluator. (a) Pima Indians Diabetes Dataset; (b) Breast Cancer Wisconsin Diagnostic Dataset; (c) Credit Approval Dataset; and, (d) Adult Income Dataset.

In table 5.33, we can see that all communication costs per data sample increase linearly with the variables of interest. Both the amount of bytes sent and received by the GC evaluator during pre-computation depend only on the numeric precision, and the amount of bytes sent during runtime depends only on the numeric precision and the number of features.

Regarding the amount of bytes per data sample received during runtime (tables 5.34, 5.35, 5.36, 5.37, and figure 5.8), they depend linearly on the number of features and the number of clusters, and quadratically on the numeric precision. However, as we have seen before, the results on table 5.7 showed that the loss of accuracy is negligible even when only 12 bits are

considered, meaning we can easily minimize its effects.

5.6.3   Partial Homomorphic Encryption

For the PHE systems, both the key size and the ciphertexts size depend only on the number of bits chosen ($NBits$). For the Paillier cryptosystem in particular, both the public and private keys are composed of two $2*NBits$ numbers and any ciphertext is a $2*NBits$ number.

Under the assumption that one of the parties owns the data to be evaluated and the other owns the evaluation model and has the computational power to perform the evaluation, we only need to determine the communication cost of transmitting the data to be evaluated from one party to the other. This cost is independent from the ML algorithm considered.

For each of the data samples, each individual feature value needs to be encrypted. The communication cost, in bits, is therefore given by:

$$cost_{comm} = \underbrace{2(2NBits)}_{publickey} + \underbrace{Nn(2NBits)}_{ciphereddata} + \underbrace{2NBits}_{cipheredresult} \tag{5.5}$$

where N is the number of samples and n is the number of features per sample. As mentioned before, the ciphertexts containing the actual data overwhelm the other contributions. We present the communication costs for the datasets considered in table 5.38.

Table 5.38: PHE. All datasets. Communication costs in kilobytes (kB).

NBits	Pima		Breast Cancer		Credit Approval		Adult Income	
	total comm cost	cost / sample	total comm cost	cost / sample	total comm cost	cost / sample	total comm cost	cost / sample
128	29.536	0.352	81.696	1.056	168.192	1.728	25318.75	3.552
256	59.072	0.704	163.392	2.112	336.384	3.456	50637.50	7.104
512	118.144	1.408	326.784	4.224	672.768	6.912	101275.1	14.208
1024	236.288	2.816	635.568	8.448	1345.536	13.824	202550.0	28.416
2048	472.576	5.632	1307.136	16.896	2691.072	27.648	405100.0	56.832

As expected, the communication costs increase linearly with increasing the number of bits ($NBits$), the number of samples and the number of features. The communication costs for most datasets are considerably small, around a few megabytes. Even for the larger dataset, the Adult Income Dataset, the larger costs are due only to the much higher number of samples considered; the cost per sample is still around a few kilobytes.

5.6.4   Fully Homomorphic Encryption

Considering the FHE system used by the HElib toolkit, there was no easy way to precisely compute the total communication cost. The details of the cryptographic key generation process are not included in the toolkit documentation, and both the cryptographic keys and the ciphertexts are represented using their own structure. By printing several examples of cryptographic keys and ciphertexts, we estimated that each key is composed of approximately $400,000$ 64-bit values (total:$w_{key} \approx 3200kB = 3.2MB$) and each ciphertext is composed of approximately $100,000$ 64-bit values (total:$w_{ciphertext} \approx 800kB = 0.8MB$).

Under the assumption that one of the parties owns the data to be evaluated and the other owns the evaluation model and has the computational power to perform the evaluation, we

only need to determine the communication cost of transmitting the data to be evaluated from one party to the other. This cost is independent from the ML algorithm considered.

For each of the data samples, each individual feature value needs to be encrypted. The communication cost, in bits, is therefore given by:

$$cost_{comm} = \underbrace{w_{key}}_{publickey} + \underbrace{Nnw_{ciphertext}}_{ciphereddata} + \underbrace{w_{ciphertext}}_{cipheredresult} \tag{5.6}$$

where N is the number of samples and n is the number of features per sample. As mentioned before, the ciphertexts containing the actual data overwhelm the other contributions. We present the communication costs for the datasets considered in table 5.39.

Table 5.39: FHE. All datasets. Communication costs in kilobytes (kB).

Pima		Breast Cancer		Credit Approval		Adult Income	
total comm cost	cost / sample	total comm cost	cost / sample	total comm cost	cost / sample	total comm cost	cost / sample
∼740	∼10.4	∼2044	∼28.0	∼4206	∼44.8	∼632970	∼90.4

Once again, the communication costs increase linearly with increasing the number of bits ($NBits$), the number of samples and the number of features. However, we can observe the negative effect of the extremely long keys required by the FHE system. The communication costs for all datasets are extremely high. Even if only a single data sample is considered, several megabytes are required for transmitting the corresponding ciphertext.

5.7   Experimental Results - Conclusions

To conclude this chapter, we now make some final observations on the obtained results, analyzing the pros and cons of our GC and Homomorphic Encryption (HE) approaches.

Although we did not compare the performance of GC and HE directly, for instance by choosing

a ML algorithm and implementing it using both privacy-preserving techniques, it is clear that the HE approach is more adequate for ML algorithms that rely on arithmetic operations, and the GC approach is more adequate for ML algorithms that rely on non-arithmetic operations. An example pointing in this direction is the quadratic increase in runtime verified in the k-M+GC experiments, due to the need to perform multiplications to compute the Euclidean distance.

An important remark on our experiments with GC is related to our choice to only analyze fully expanded DT instead of the original ones 4.5.1, in order to prevent any information leakage regarding the shape of the original tree. However, in most cases this causes an exponential growth of the number of nodes with increasing tree depths, leading to proportional increases in both the execution times and the communication costs.

An important remark on our experiments with HE is when each of the techniques should be used. We verified that PHE is in fact usable in practice but under some restrictions (*e.g.*: if there is no need for complex composition of operations, if data is separated between client and server), while FHE is more flexible but still too computationally expensive. However, due to the data packing "trick", FHE can be more efficient than PHE for evaluating some ML algorithms (*e.g.*: LR).

5.8 Summary

6

Conclusions and Future Work

6.1   Conclusions

6.2   Future Work

Todo list

Bibliography

[1] Rakesh Agrawal and Ramakrishnan Srikant. Privacy-preserving data mining. In *ACM Sigmod Record*, volume 29, pages 439–450. ACM, 2000.

[2] Ross Anderson. *Security engineering*. John Wiley & Sons, 2008.

[3] Ian F Blake and Vladimir Kolesnikov. Strong conditional oblivious transfer and computing on intervals. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 515–529. Springer, 2004.

[4] Dan Bogdanov, Liina Kamm, Baldur Kubo, Reimo Rebane, Ville Sokk, and Riivo Talviste. Students and taxes: a privacy-preserving study using secure computation. *Proceedings on Privacy Enhancing Technologies*, 2016(3):117–135, 2016.

[5] Dan Boneh, Amit Sahai, and Brent Waters. Functional encryption: Definitions and challenges. In *Theory of Cryptography Conference*, pages 253–273. Springer, 2011.

[6] Raphael Bost, Raluca Ada Popa, Stephen Tu, and Shafi Goldwasser. Machine learning classification over encrypted data. In *NDSS*, volume 4324, page 4325, 2015.

[7] Ljiljana Brankovic and Vladimir Estivill-Castro. Privacy issues in knowledge discovery and data mining. In *Australian institute of computer ethics conference*, pages 89–99, 1999.

[8] Justin Brickell and Vitaly Shmatikov. Privacy-preserving classifier learning. In *International Conference on Financial Cryptography and Data Security*, pages 128–147. Springer, 2009.

[9] Kamalika Chaudhuri and Claire Monteleoni. Privacy-preserving logistic regression. In *Advances in Neural Information Processing Systems*, pages 289–296, 2009.

[10] Giuseppe D'Acquisto, Josep Domingo-Ferrer, Panayiotis Kikiras, Vicenç Torra, Yves-Alexandre de Montjoye, and Athena Bourka. Privacy by design in big data: An overview of privacy enhancing technologies in the era of big data analytics. *European Union Agency for Network and Information Security*, 2015.

[11] George Danezis, Josep Domingo-Ferrer, Marit Hansen, Jaap-Henk Hoepman, Daniel Le Metayer, Rodica Tirtea, and Stefan Schiffner. Privacy and data protection by design-from policy to engineering. *European Union Agency for Network and Information Security*, 2015.

[12] Yves-Alexandre De Montjoye, César A Hidalgo, Michel Verleysen, and Vincent D Blondel. Unique in the crowd: The privacy bounds of human mobility. *Scientific reports*, 3: 1376, 2013.

[13] Daniel Demmler, Thomas Schneider, and Michael Zohner. Aby-a framework for efficient mixed-protocol secure two-party computation. In *NDSS*, 2015.

[14] Mahir Can Doganay, Thomas B Pedersen, Yücel Saygin, Erkay Savaş, and Albert Levi. Distributed privacy preserving k-means clustering with additive secret sharing. In *Proceedings of the 2008 international workshop on Privacy and anonymity in information society*, pages 3–11. ACM, 2008.

[15] Cynthia Dwork. Differential privacy. *Proceedings of the 33rd International Colloquium on Automata, Languages and Programming*, pages 1–12, 2006. ISSN 03029743. doi: 10.1007/11787006_1.

[16] Cynthia Dwork, Aaron Roth, et al. The algorithmic foundations of differential privacy. *Foundations and Trends in Theoretical Computer Science*, 9(3–4):211–407, 2014.

[17] Taher ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE transactions on information theory*, 31(4):469–472, 1985.

[18] Craig Gentry et al. Fully homomorphic encryption using ideal lattices. In *Symposium on the Theory of Computing*, volume 9, pages 169–178, 2009.

[19] Oded Goldreich. Secure multi-party computation. *Manuscript. Preliminary version*, pages 86–97, 1998.

[20] Shafi Goldwasser. Multi party computations: past and present. In *Proceedings of the sixteenth annual ACM symposium on Principles of distributed computing*, pages 1–6. ACM, 1997.

[21] Adam Groce, Alex Ledger, Alex J Malozemoff, and Arkady Yerukhimovich. Compgc: Efficient offline/online semi-honest two-party computation. *IACR Cryptology ePrint Archive*, 2016:458, 2016.

[22] Munawar Hafiz, Paul Adamczyk, and Ralph E Johnson. Organizing security patterns. *IEEE software*, 24(4), 2007.

[23] Shai Halevi and Victor Shoup. Helib-an implementation of homomorphic encryption. *Cryptology ePrint Archive, Report 2014/039*, 2014.

[24] Jiawei Han, Jian Pei, and Micheline Kamber. *Data mining: concepts and techniques.* Elsevier, 2011.

[25] Marit Hansen, Meiko Jensen, and Martin Rost. Protection goals for privacy engineering. *Proceedings - IEEE Security and Privacy Workshops, SPW*, pages 159–166, 2015. doi: 10.1109/SPW.2015.13.

[26] David Harris and Sarah Harris. *Digital design and computer architecture.* Morgan Kaufmann, 2010.

[27] Vladimir Kolesnikov and Thomas Schneider. Improved garbled circuit: Free xor gates and applications. *Automata, Languages and Programming*, pages 486–498, 2008.

[28] Lei Xu, Chunxiao Jiang, Jian Wang, Jian Yuan, Yong Ren, Lei Xu, Chunxiao Jiang, Jian Wang, Jian Yuan, and Yong Ren. Information Security in Big Data: Privacy and Data Mining. *IEEE Access*, 2:1149–1176, 2014. ISSN 2169-3536.

[29] Ninghui Li, Tiancheng Li, and Suresh Venkatasubramanian. t-closeness: Privacy beyond k-anonymity and l-diversity. In *Data Engineering. ICDE. IEEE 23rd International Conference on*, pages 106–115. IEEE, 2007.

[30] Rongxing Lu, Hui Zhu, Ximeng Liu, Joseph Liu, and Jun Shao. Toward efficient and privacy-preserving computing in big data era. *IEEE Network*, 28(4):46–50, 2014. ISSN 08908044. doi: 10.1109/MNET.2014.6863131.

[31] Ashwin Machanavajjhala, Daniel Kifer, Johannes Gehrke, and Muthuramakrishnan Venkitasubramaniam. l-diversity: Privacy beyond k-anonymity. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 1(1):3, 2007.

[32] Moni Naor and Benny Pinkas. Computationally secure oblivious transfer. *Journal of Cryptology*, 18(1):1–35, 2005.

[33] Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *International Conference on the Theory and Applications of Cryptographic Techniques*, pages 223–238. Springer, 1999.

[34] Benny Pinkas, Thomas Schneider, Nigel P Smart, and Stephen C Williams. Secure two-party computation is practical. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 250–267. Springer, 2009.

[35] Michael O Rabin. How to exchange secrets with oblivious transfer. *IACR Cryptology ePrint Archive*, 2005:187, 2005.

[36] Balaji Raghunathan. *The Complete Book of Data Anonymization: From Planning to Implementation*. CRC Press, 2013.

[37] Ronald L Rivest, Len Adleman, and Michael L Dertouzos. On data banks and privacy homomorphisms. *Foundations of secure computation*, 4(11):169–180, 1978.

[38] Ronald L Rivest, Adi Shamir, and Leonard Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978.

[39] Paul M Schwartz and Daniel J Solove. The pii problem: Privacy and a new concept of personally identifiable information. *NYUL rev.*, 86:1814, 2011.

[40] Adi Shamir. How to share a secret. *Communications of the ACM*, 22(11):612–613, 1979.

[41] Mark Shaneck, Yongdae Kim, and Vipin Kumar. Privacy preserving nearest neighbor search. In *Data Mining Workshops. ICDM Workshops. Sixth IEEE International Conference on*, pages 541–545. IEEE, 2006.

[42] Ebrahim M Songhori, Siam U Hussain, Ahmad-Reza Sadeghi, Thomas Schneider, and Farinaz Koushanfar. Tinygarble: Highly compressed and scalable sequential garbled circuits. In *Security and Privacy (SP), 2015 IEEE Symposium on*, pages 411–428. IEEE, 2015.

[43] Carolin Strobl, James Malley, and Gerhard Tutz. An introduction to recursive partitioning: rationale, application, and characteristics of classification and regression trees, bagging, and random forests. *Psychological methods*, 14(4):323, 2009.

[44] Latanya Sweeney. k-anonymity: A model for protecting privacy. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 10(05):557–570, 2002.

[45] Jaideep Vaidya, Basit Shafiq, Anirban Basu, and Yuan Hong. Differentially private naive bayes classification. In *Proceedings of the 2013 IEEE/WIC/ACM International Joint Conferences on Web Intelligence (WI) and Intelligent Agent Technologies (IAT)-Volume 01*, pages 571–576. IEEE Computer Society, 2013.

[46] Stanley L Warner. Randomized response: A survey technique for eliminating evasive answer bias. *Journal of the American Statistical Association*, 60(309):63–69, 1965.

[47] Rüdiger Wirth and Jochen Hipp. Crisp-dm: Towards a standard process model for data mining. In *Proceedings of the 4th international conference on the practical applications of knowledge discovery and data mining*, pages 29–39. Citeseer, 2000.

[48] Andrew C Yao. Protocols for secure computations. In *Foundations of Computer Science, 1982. SFCS'08. 23rd Annual Symposium on*, pages 160–164. IEEE, 1982.

[49] Andrew Chi-Chih Yao. How to generate and exchange secrets. In *Foundations of Computer Science, 1986., 27th Annual Symposium on*, pages 162–167. IEEE, 1986.

[50] Hwanjo Yu, Jaideep Vaidya, and Xiaoqian Jiang. Privacy-preserving svm classification on vertically partitioned data. *Advances in Knowledge Discovery and Data Mining*, pages 647–656, 2006.