H2020 – ICT-13-2018-2019

Machine Learning to Augment Shared Knowledge in Federated Privacy-Preserving Scenarios (MUSKETEER)

Grant No 824988

D4.1 Investigative overview of targeted architecture and algorithms

May 19

Imprint

Contractual Date of Delivery to the EC: 31 May 2019

Author(s): Roberto Diaz (TREE), Ángel Navia (UC3M), Francisco J. González (UC3M)
Participant(s): Jaime Medina (TREE); Other
Reviewer(s): Luis Muñoz (Imperial College); Petros Papachristou (Hygeia)

Project: Machine learning to augment shared knowledge in federated privacy-preserving scenarios (MUSKETEER)

Work package: WP4
Dissemination level: Public
Version: 0.1

Contact: roberto.diaz@treelogic.com
Website: www.MUSKETEER.eu

Legal disclaimer

The project Machine Learning to Augment Shared Knowledge in Federated Privacy-Preserving Scenarios (MUSKETEER) has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 824988. The sole responsibility for the content of this publication lies with the authors.

Copyright

Executive Summary

This deliverable D4.1 – Investigative overview of targeted architecture and algorithms - is the first outcome of WP4. It includes a technical report with the state of art of privacy preserving machine learning and the technical description of the different Privacy Operation Modes (POMs) that will be implemented in the MUSKETEER platform. This deliverable also contains examples about how machine learning algorithms can be implemented on top of every POM.

Document History

Version	Date	Status	Author	Comment
1	01 May 2019	For internal review	Roberto Diaz	First draft
2	05 May 2019	Review inputs	Luis Muñoz	Update
3	28 May 2019	Review inputs	Petros Papachristou	Update
4	30 May 2019	Final Version	Roberto Diaz	Update

Table of Contents

List of Figures

List of Acronyms and Abbreviations

Abbreviation	Definition
CA	Consortium Agreement
DP	Differential Privacy
FHE	Fully Homomorphic Encryption
GA	Grant Agreement
HE	Homomorphic Encryption
MK	Master Key
ML	Machine Learning
PHE	Partial Homomorphic Encryption
PK	Public Key
POM	Privacy Operation Mode
PP	Privacy Preserving
PPML	Privacy Preserving Machine Learning (a.k.a. Privacy Preserving Data Mining)
SDP	Secure Dot Product
SFE	Secure Function Evaluation
SMC	Secure Multiparty Computing
SSS	Shamir Secret Sharing
SMM	Secure Matrix Multiplication
SSP	Secure Sum Protocol
SVM	Support Vector Machine

1 Introduction

1.1 Purpose

This deliverable includes a technical report with the description of the different Privacy Operation Modes (POMs), their related state-of-art and shows how machine learning algorithms can be implemented on top of every POM.

1.2 Related Documents

D4.1 will serve as a basis for D3.1 Architecture design – Initial version. The machine learning algorithms description and the POM communications schemas will set the groundwork for the architecture definition, to be provided in D3.1. D4.1 will also be taken into consideration for D4.2, D4.4 and D4.6.

1.3 Document Structure

This document is organized as follows. In Section 2, we briefly revisit the concepts of Machine Learning. Section 3 summarizes the motivations and current barriers encountered in data sharing. The description of some cryptographic and privacy preserving concepts is described in section 4. In section 5 we introduce the POMs and also describe how ML algorithms can be implemented over them. Finally, we provide some conclusions obtained during the development of this document.

2 Machine Learning concepts

2.1 Machine Learning 'task' definition

As a previous step, before running a privacy preserving distributed training procedure in MUSKETEER, we need to define a Machine Learning (ML) **task**, which is one of the basic elements in MUSKETEER. It can be considered as a problem statement that feeds from data and produces a trained machine learning model as an outcome. Any end user can create one or more new tasks with an associated unique task identifier (***task_id***), such that many of them can be run in parallel. The definition step requires the specification of the main task characteristics:

General description: a high level description of the task (the problem to be solved) is necessary for a rapid identification of existing tasks by other users, to ascertain if they can contribute/enhance it or not with more data.

Data: it is recommended (but not mandatory[1]) to facilitate an initial dataset, i.e., some data illustrating the task to be solved. Without loss of generality we will assume that the data comprises input feature vectors (**x**) (for non-supervised tasks) and pairs of input feature vectors and target values (**x**, **t**) (for supervised tasks).

Input features description: a general description of the input features (like defining the fields in a Table) is necessary to unify the data representation among users and finally being able to combine all the contributed data during the learning stage. In the above-mentioned general case where input data is represented as a vector, the meaning of every field in such a vector must be explicitly described, for compatibility purposes. *One feature vector could comprise, for instance: gender (man/woman), study_level (primary/graduate/phd) and age (years). Two sample instances of that feature vector could be: [man, graduate, 47] and [woman, phd, 36].* After some preprocessing (with common parameters to all end users), the features must eventually be converted into a numerical vector. More details about the preprocessing options will be given in Section 1.4.

Target values: the problem to be solved is defined by the target values (at least in supervised tasks). For instance, given the above described features, the target could be to estimate the annual income in euros, or to estimate if that person is unemployed or not. The definition and nature of the target must also be shared among all the participants during the task definition such that they can contribute with new pairs (**x**, **t**) to the training process.

Privacy requirements: it is important that the end user determines which are the privacy restrictions that apply to his/her data, because those restrictions will determine the kind of operations that can be used. It is more operative that every user declares the privacy restrictions that apply to his/her data and then the platform offers the available mechanisms to solve the ML task. The ***data_privacy*** parameter can be chosen among several options, described in natural language to facilitate the specification of the task to the end user, for example, one end user may adhere to some of the following statements:

- my data is open and can be freely distributed
- my data can be shared after anonymization

[1] One end user may want to obtain a model but has no data for training it. Just having a task definition in mind it would possible to ask MUSKETEER to find other users with data that want to contribute to that task, expecting to obtain some reward.

- my data can be shared only with the MUSKETEER platform under some confidentiality agreement with the platform
- my data can be shared with other end users under some confidentiality agreement with the end users
- my raw data cannot leave my facilities (only the MUSKETEER client can see it and obtain some operations on it: gradients, dot products, etc., but never reveal individual data points.)
- my data can be used for a given task, but not for other tasks

2.2 Pre-processing, normalization and data alignment:

Prior to the training process itself, sometimes it is necessary/convenient to perform some sort of data normalization or pre-processing. We will distinguish between the pre-processing tasks that can be locally done at every end user without information from other end users, and data pre-processing that requires global information about all the end users (to be secretly shared).

2.2.1 Data merging and scaling algorithms

The most straightforward pre-processing is what we can describe as an "Ad hoc" local pre-processing algorithm. In such cases the feature vector is the result of applying some (possibly complex) pre-processing to a raw piece of information (for instance an image, a text, a voice recording, etc.). We assume that in those cases a "item-wise" pre-processing algorithm exists and is able to transform such a raw data into a useful numerical vector. By "item-wise" we mean that the pre-processing algorithm can be applied to a single input data without knowledge of the other input elements (either in the same end user or in other end users). In those cases, assuming that the raw data cannot be transmitted to a central location and batch processed in a single place due to privacy restrictions, it is necessary to share the pre-processing algorithm such that every end user is able to transform its own raw data into a common representation (the feature vector **x)**. For instance, in the case where input data are images, some transformations can be applied to every image before feeding it into a machine learning model, such as high pass filtering followed by an edge detection, textures parameterization, a specific feature extraction, etc. In the case of texts, the pre-processing could be a bag of words with TFIDF weighting, etc. The casuistic can be extremely large and problem dependant, so it is important to guarantee that the pre-processing module always produces an output vector with the expected content and format and it is recommended that the "ad hoc" (non-standard) pre-processing algorithms be defined and implemented by the end users that define a specific task, such that the algorithm can be shared with other users contributing to the task as a "pre-processing object".

Once the input data has been transformed into a manageable feature vector, it could be necessary to apply a second level of pre-processing/normalization, this time taking into account all the data from all the users. For instance, it is common to feed a machine learning algorithm with zero mean, unit standard deviation data. Obviously, the global mean depends on all the patterns from all users, and therefore, privacy preserving mechanisms such as Secure Sum Protocols (SSP) are needed to estimate such global mean values without revealing individual values of a particular end user to the others. Secure Multiparty Computing protocols for implementing secure sums are available in the literature and can thus be used for this normalization purpose [Zhu_2011] [Mehnaz_2017].

2.2.2 Data alignment

The data alignment process aims at detecting if all the contributing users are providing data that serve to the same machine learning task. This is a preliminary check before proceeding with the full training process, possibly by means of a fast evaluation of a linear model. It would be a waste of time to deploy a -possibly costly- training procedure without a minimal verification of the suitability of the available data for a given task. At this stage, only very clear deviations from the common objective can be detected, more subtle data perturbations will be detected in the “attack detection” modules.

One possible way of detecting such data misalignment is to evaluate if the gradients obtained by a given user are minimally correlated with the gradients of the other (specially with respect to the gradients of the user that has defined the task). Under POM6, it is possible to evaluate the data alignment by means of the correlation analysis among the exchanged covariance matrices.

Finally, a more general approach is to evaluate performance of a locally trained model on data provided by other users. If the expected merit figures largely deviate from the local results, possibly a task misalignment is present.

2.3 Supervised Learning

This is the area of learning a function that maps an input to an output based on training example pairs. It can also be subdivided into classification (the output variable takes categorical values) and regression (the output variable takes continuous values). The library will contain algorithms capable to infer functions of different nature.

2.3.1 Linear models

A simple but widely used class of Machine Learning models, able to make a prediction by using a linear combination of the input features. We will include alternatives for classification (logistic classifier) and regression (linear regression) with different regularization alternatives and cost functions.

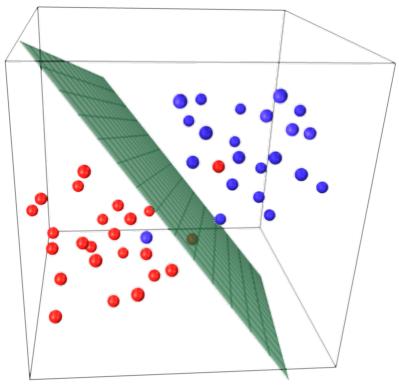

Figure 1 A linear classifier in a 3 dimensions dataset

2.3.2 Kernel methods

They comprise a very popular family of Machine Learning models. The main reason of their success is their ability to easily adapt linear models to create non-linear solutions by transforming the input data space onto a high dimensional one where the inner product between projected vectors can be computed using a kernel function. We will provide solutions for classification (SVMs) and regression (Gaussian Processes), possibly under model complexity restrictions (budgeted models).

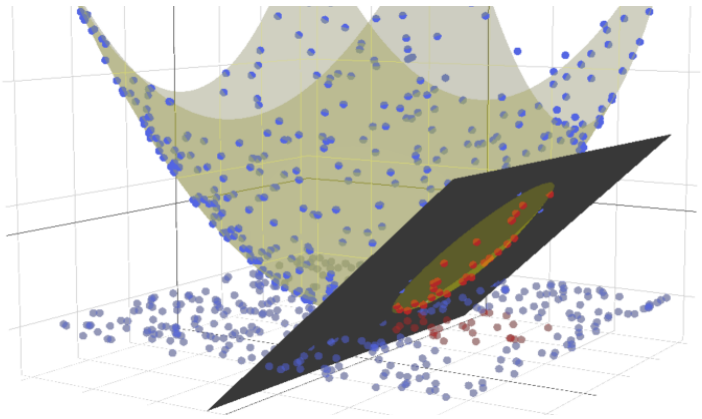

Figure 2 Non linear classification using a kernel method

2.3.3 Trees

They can be used for decision support while allowing a visual representation and explicit interpretation of the results. As the name goes, they use a tree-like model of decisions. They are a commonly used tool in machine learning when some model interpretation is needed. They can be used for both classification and regression problems.

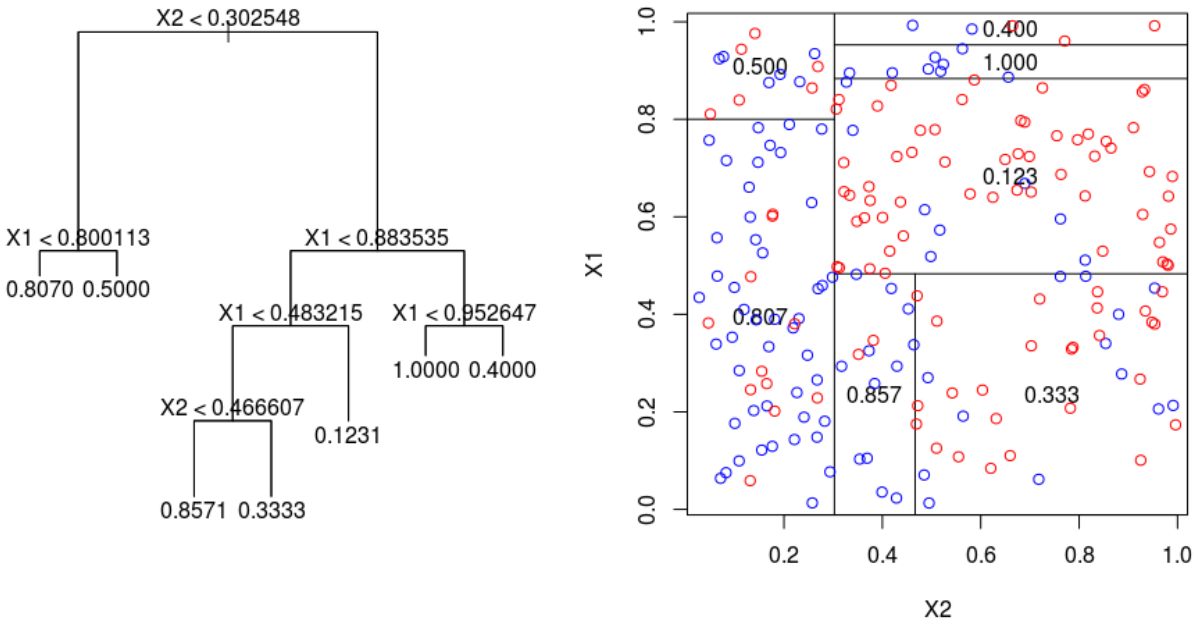

Figure 3 A decision tree classification

2.3.4 Deep Neural Networks

Deep learning architectures such as recurrent neural networks or convolutional neural networks are currently the state of art over a wide variety of fields including computer vision, speech recognition, natural language processing, audio recognition, machine translation, bioinformatics and drug design, where they have produced results comparable to and in some cases superior to human experts.

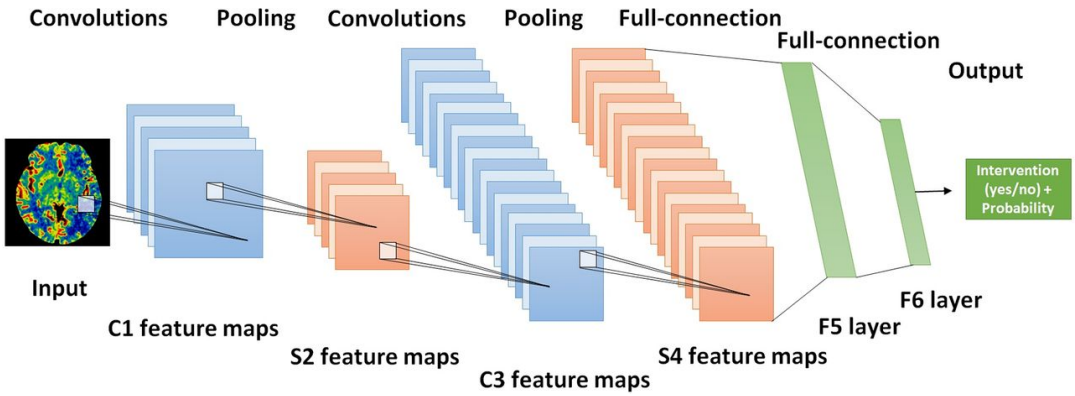

Figure 4 A convolutional neural network for medical images

2.4 Unsupervised Learning

This is the machine learning task of inferring a function to describe hidden structure from "unlabeled" data. The library will include algorithms for clustering and input space component decomposition.

2.4.1 Clustering

Is the task of dividing the population or data into a number of groups such that data points in the same groups are more similar to other data points in the same group than those in other groups. In simple words, the aim is to segregate groups with similar characteristics and assign them into clusters. The library will include general purpose clustering algorithms such as k-means.

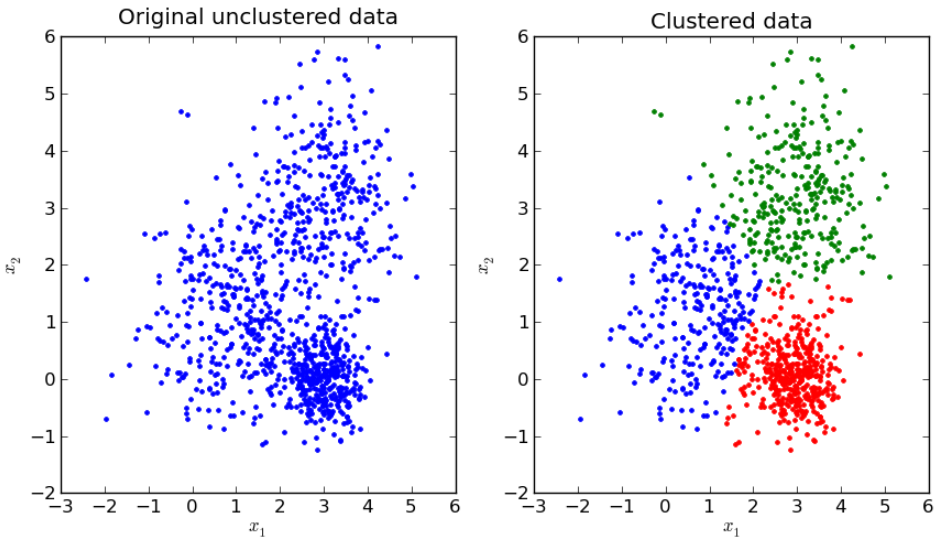

Figure 5 Example of data clustering

2.4.2 Component Decomposition

Is the task of dividing a single data into different subcomponents. The library will include algorithms such as Principal Component Analysis (PCA).

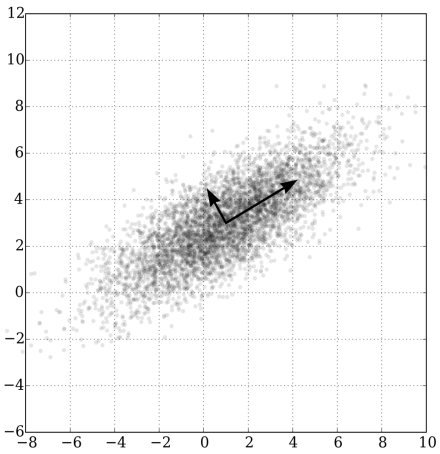

Figure 6 Main components of a 2 dimensional dataset

3 Data sharing motivation and barriers

3.1 Motivation to Share Data

Machine learning algorithms create predictive models using **training datasets** composed of historical records**.** In general, complex tasks require complex models. Since creating complex models requires large datasets, one **common barrier** to use advanced machine learning techniques is the **amount of data** needed to train a model. With insufficient training data, the models fail to correctly generalize from training data to unseen data (a problem called overfitting), and they achieve poor accuracy.

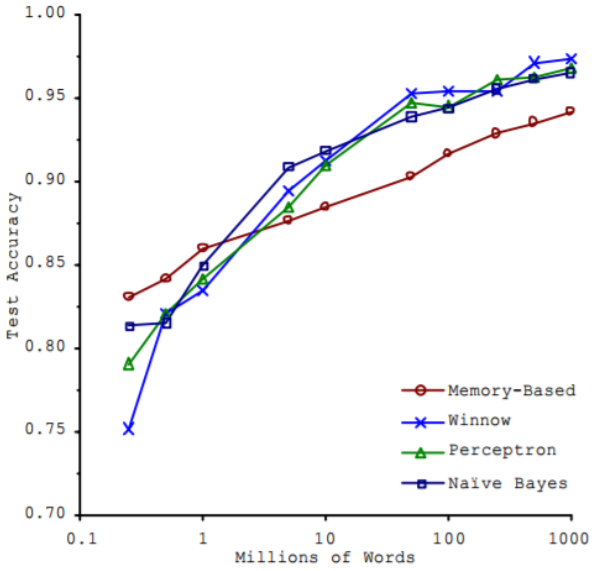

Figure 7 Accuracy as a function of the number of training data from [Banko_2001]

One approach to mitigate overfitting is to limit the complexity of the models by using regularization techniques. A more favourable approach is to increase the amount of training data. This is illustrated by a famous quote of Google's Research Director Peter Norvig: "*We don't have better algorithms. We just have more data.*" The effect described by Norvig had been known in the literature for a long time; it is illustrated e.g. in [Banko_2001], where it is shown that for a given problem, adding more examples to the training set increases the accuracy of the model (see Figure 7).

Generally speaking, the larger the amount of accurately labelled training data, the better the performance of the machine learning models and hence the quality of the data-driven business insights. This need for huge volumes of data has driven over the past two decades the emergence of so-called "Big Data Platforms", which are especially designed (typically distributed) databases capable of storing and efficiently indexing tera- or even petabytes of raw data in various formats (e.g. columnar, image/video, text).

In practice, however, **compiling large high-quality datasets** is **labour intensive and time consuming**. One problem is the accurate labelling of the data, which often requires manual data inspection, cleansing and tagging by domain experts. Another problem, particularly when dealing with sensor data, is the limited availability of historical records. Therefore, we can identify two main scenarios where **partners** in a **data economy** can benefit from **sharing** or **acquiring datasets** in order to improve the quality of their own respective machine learning models. The first one is **increasing the number of data samples**, since a machine learning model can improve its accuracy if different datasets are merged into a bigger one (horizontal partition). The second one is **increasing the number of variables**, in case different partners own complementary information to solve a single problem (vertical partition). A machine learning model that can use all the variables collected by different partners can arrive at a better capturing of the problem to be solved.

3.2 Main Barriers

How to train machine learning algorithms using data collected from different data providers while mitigating privacy concerns is a challenging problem. Indeed, **data sharing** and **trading** are seen by the Big Data Value Association (BDVA) as important ecosystem **enablers** in the **data economy** in its Strategic Research and Innovation Agenda[2].

We can find different barriers that limit the free flow of data[3]:

- **Legal barriers:** Data localisation stems from legal rules or administrative guidelines or practices that dictate or influence the localisation of data for its storage or processing. Such requirements restrict the free flow of data between regions or countries.
- **Data ownership:** Information stored in digital form can be easily copied and redistributed. The main concern of data providers when they provide access to their datasets is how to avoid that digital copies could be re-distributed.
- **Data confidentiality:** Although a company can benefit from common predictive models joining efforts with another company in creating the dataset, these data may contain confidential information about internal industrial processes that cannot be exposed publicly without revealing business secrets that would benefit competitors and put the company itself into a disadvantage.
- **Personal Information leakage:** Many data sources contain personal information (e.g. images from cameras, client records) which raises concerns and fears in the population about possible information leakages.
- **Different Privacy Policies:** Different partners may have **different privacy policies**, and they may not be willing (or allowed) to explicitly and publicly **share their data**, even

[2] http://www.bdva.eu/sites/default/files/BDVA_SRIA_v4_Ed1.1.pdf
[3] http://eur-lex.europa.eu/legal-content/EN/TXT/PDF/?uri=CELEX:52017SC0002&from=EN

though a great benefit in terms of improving business processes could be obtained by sharing the data among them.

In MUSKETEER different Privacy Operation Modes (POMs) will be implemented and over them we are going to develop the machine learning algorithms. These POMs have been designed to remove some of these barriers. Each one describing a potential scenario with different privacy preserving demands, but also with different computational, communication, storage and accountability features.

4 Cryptographic concepts

4.1 Concepts

In this section we introduce some concepts that are needed to understand the privacy operation modes included in MUSKETEER.

4.1.1 Public key cryptography

It is a cryptographic system that uses pairs of keys: public keys which may be disseminated widely, and private keys which are known only to the owner. The generation of such keys depends on cryptographic algorithms based on mathematical problems to produce one-way functions. Effective security only requires keeping the private key private; the public key can be openly distributed without compromising security.

In such a system, any person can encrypt a message using the receiver's public key, but that encrypted message can only be decrypted with the receiver's private key.

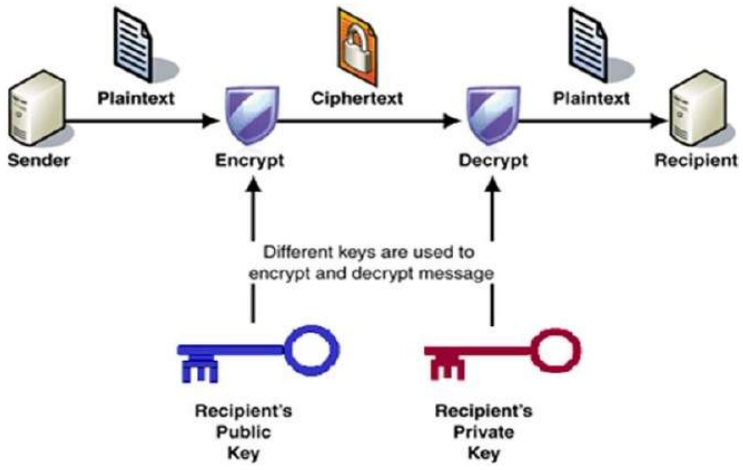

Figure 8 Public key encryption and decryption schema

4.1.2 Homomorphic encryption

Homomorphic encryption can be viewed as an extension of public-key cryptography with an additional evaluation capability for computing over encrypted data without access to the secret key. The result of such a computation remains encrypted. The word homomorphic refers to homomorphism in algebra: the encryption and decryption functions can be thought as homomorphisms between plaintext and ciphertext spaces.

This encryption includes multiple types of encryption schemes that can perform different classes of computations over encrypted data. [Armknecht_2015] Some common types of homomorphic encryption are partially homomorphic, somewhat homomorphic, levelled fully homomorphic, and fully homomorphic encryption.

4.1.3 Paillier Cryptosystem

The Paillier cryptosystem [Paillier_1999] [San_2016] is a probabilistic asymmetric algorithm for public key cryptography. The scheme is an additive homomorphic cryptosystem; this means that, given only the public key and the encryption of m1 and m2, one can compute the encryption of m1+m2.

An additive homomorphic operation allows the computation of the linear combination of two plaintexts through ciphertext manipulation:

$$\left[\left[\alpha m_1 + \beta m_2\right]\right] = [[m_1]]^{\alpha} * [[m_2]]^{\beta}$$

where m_1 and m_2 are plaintext signed integer constants, $[[m_1]]^{\alpha}$ is the modular exponential, and $[[m_1]]^{-1}$ is the modular multiplicative inverse: $[[m_1]]^{-1} * [[m_1]]^{1} = 1\ mod\ N$.

4.1.4 Proxy Re-Encryption

Proxy re-encryption (PRE) is a special type of public-key encryption that permits a "proxy" (or a user) to transform ciphertexts from one public key to another, without the proxy being able to learn any information about the original message. In more detail, PRE enables us to convert a ciphertext under public key PK_f ("f" denotes "from") into another ciphertext under public key PK_t ("t" denotes "to") by using a re-encryption key $RK_{f \to t}$ without decrypting the original ciphertext by the secret key SK_f.

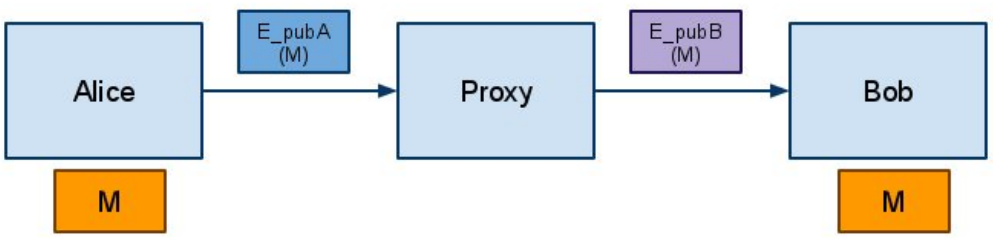

Figure 9 Proxy Re-Encryption schema

[Blaze_1998] proposed the first PRE scheme in such a semi-honest framework. This one is based on the ElGamal cryptosystem and on a set of secret pieces of information, referred as secret re-encryption key, entity A has to send to the proxy so as to make possible the change of the public key encryption (i.e., re-encrypt data with entity B's public key). One main issue of this proposal is that this scheme is inherently bidirectional, that is to say that the re-encryption key which allows transferring cipher-texts from A to B, enables the proxy to convert all B's cipher-texts under A's public key. There are also unidirectional approaches. In [Dodis_2003] the re-encryption key provided by A is split into two parts, one for the proxy and the other for B. [Deng_2008] proposed an asymmetric cross cryptosystem re-encryption scheme instead of pairing. Beyond, if the above approaches allow one user to share data with another one, they do not make possible the processing of encrypted data by the cloud or proxy. This capacity is usually achieved with the help of homomorphic cryptosystems. The first homomorphic based PRE attempt has been proposed by [Bresson_2003], using the Paillier cryptosystem. However, even though their solution makes possible data sharing, it cannot be seen as a pure proxy re-encryption scheme. Indeed, data are not re-encrypted with the public key of the delegate. If this one wants to ask the cloud to process the data he receives from A, he has: i) first to download A data, ii) decrypt them based on some secret pieces of information provided by A; iii) re-encrypt them with his public key and send them back to the cloud. Recently, in [Bellafqira_2017], it has been proposed the first homomorphic based proxy re-encryption (HPRE) solution that allows different users to share data they outsourced homomorphically encrypted using their respective public keys with the possibility to process such data remotely.

5 Privacy Operation Modes

5.1 Federated Collaborative Privacy Operation Modes

Under these modes, data never leaves the data owners' facilities, since training takes place under the Federated Machine Learning paradigm, where the model is transferred among the users, and everyone contributes by locally updating the model, using their data. The resulting model is unique, common to all the users and at the end all users get access to the trained model in unencrypted form.

5.1.1 POM1 (ARAMIS)

Recently, Federated Machine Learning (FML) (and other related decentralized approaches) [McMahan_2017][Konečný_2016][Shokri_2015] have been proposed as an alternative to a traditional local or cloud computing for training predictive models using machine learning.

Under this paradigm, a shared global model is trained under the coordination of a central node, from a federation of participating devices.

FML enables different devices to collaboratively learn a shared prediction model while keeping all the training data on device, decoupling the ability to perform machine learning from the need to store the data in the cloud.

Using this approach, data owners can offer their data to train a predictive model without being exposed to data leakage or data attacks. In addition, since the model updates are specific to improving the current model, there is no reason to store them on the server once they have been applied.

FML turns the update of Machine Learning models upside-down by allowing the devices with data on the edge to participate in the training. Instead of sending the data in the client to a centralised location, Federated Learning sends the model to the devices participating in the federation. The model is then improved with the local data. And the data never leaves the local device. After that, the clients send updates to the model to the central node that can aggregate the different partial updates to globally update the model.

The following figure shows the iterative process that is needed to train a model.

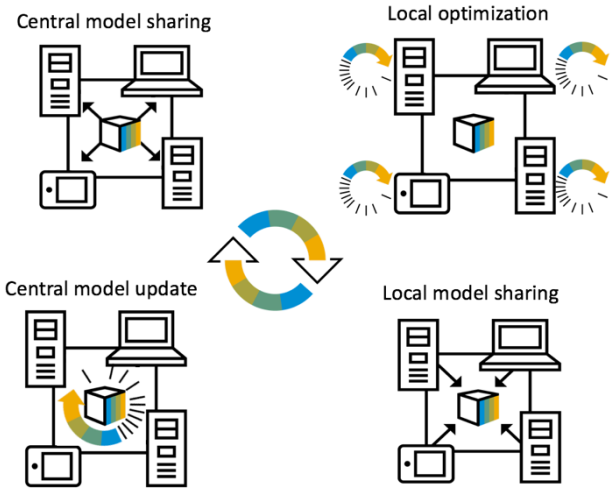

Figure 10 FML schema

A simple implementation requires that each client sends a full model (or a full model update) back to the server in each round. For large models, this step is likely to be the bottleneck of Federated Learning due to multiple factors.

To reduce the bottleneck, there are some model compressions schemes such as [Han_2015][Lin_2017] that can reduce the bandwidth necessary to download the current model. In [Konečný_2016], to reduce the communication costs they propose two alternatives: structured updates, where we directly learn an update from a restricted space parametrized

using a smaller number of variables; and sketched updates, where we learn a full model update and then compress it using a combination of quantization, random rotations, and subsampling before sending it to the server.

Federated learning can be seen as an operating system for edge computing, as it provides the learning protocol for coordination and security. In [Wang_2018], authors considered generic class of machine learning models that are trained using gradient-descent based approaches. They analyze the convergence bound of distributed gradient descent from a theoretical point of view, based on which they propose a control algorithm that determines the best trade-off between local update and global parameter aggregation to minimize the loss function under a given resource budget.

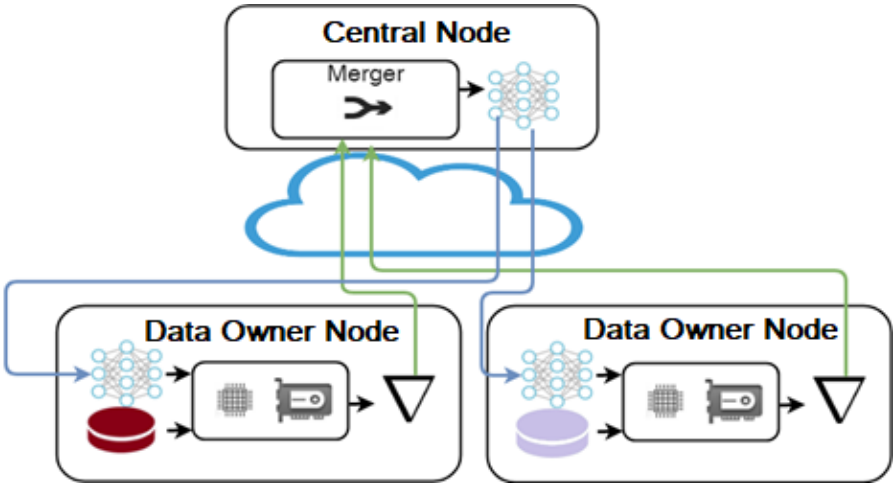

Figure 11 POM1 communication schema

Machine learning algorithms over this POM:

Linear models and neural networks:

Linear models and neural networks can be trained using Stochastic Gradient Descent (SGD) variants. This POM is a good scenario for linear models such as logistic regression or linear regression and also for neural networks that are trained obtaining the gradients of all the weights

To illustrate the concept of SGD, Machine learning considers the problem of minimizing an objective function that has the form of a sum:

$$Q(w) = \frac{1}{n}\sum_{i=1}^{n} Q_i(w),$$

where the parameter w that minimizes Q(w) is to be estimated. w is the set of weights of a linear model or a neural network and Q(w) is typically the error obtained with the model over a data.

Since we want to obtain the w that obtains the minimum error, a standard (or "batch") gradient descent method would perform the following update of w in an iterative process:

$$w := w - \eta \nabla Q(w) = w - \eta \sum_{i=1}^{n} \nabla Q_i(w)/n,$$

For example, if we want to fit a straight line y=w1+w2x to a training set with observations { x_1,x_2,...,x_n} and their corresponding correct predictions are {y_1, y_2,...,y_n} using least squares, the objective function to be minimized is:

$$Q(w) = \sum_{i=1}^{n} Q_i(w) = \sum_{i=1}^{n} (\hat{y}_i - y_i)^2 = \sum_{i=1}^{n} (w_1 + w_2 x_i - y_i)^2.$$

Then, to improve the model we must perform the following update of the weights iteratively:

$$\begin{bmatrix} w_1 \\ w_2 \end{bmatrix} := \begin{bmatrix} w_1 \\ w_2 \end{bmatrix} - \eta \begin{bmatrix} \frac{\partial}{\partial w_1}(w_1 + w_2 x_i - y_i)^2 \\ \frac{\partial}{\partial w_2}(w_1 + w_2 x_i - y_i)^2 \end{bmatrix} = \begin{bmatrix} w_1 \\ w_2 \end{bmatrix} - \eta \begin{bmatrix} 2(w_1 + w_2 x_i - y_i) \\ 2x_i(w_1 + w_2 x_i - y_i) \end{bmatrix}.$$

This is a good training procedure for a federated environment since the gradients are computed over individual data and can be computed in the edge. Then they can be sent to a central server that can average them and update the model.

Clustering algorithms:

Some clustering algorithms are also easy to implement over this POM. For example, K-means. In this algorithm, given a dataset (x1, x2, …, xn), k-means creates k sets of data (k ≤ n) by minimizing the following function.

$$\arg\min_{\mathbf{S}} \sum_{i=1}^{k} \sum_{\mathbf{x}_j \in S_i} \|\mathbf{x}_j - \boldsymbol{\mu}_i\|^2$$

Where μi is the average of the data that belongs to the set Si.

The most common algorithm uses an iterative refinement technique. Due to its ubiquity, it is often called the k-means algorithm; it is also referred to as Lloyd's algorithm, particularly in the computer science community.

Given an initial set of k means m1(1), …, mk(1), the algorithm proceeds by alternating between two steps:

- Assignment step: Assign each observation to the cluster whose mean has the least squared Euclidean distance; this is intuitively the "nearest" mean.

$$S_i^{(t)} = \{x_p : \|x_p - m_i^{(t)}\|^2 \leq \|x_p - m_j^{(t)}\|^2 \ \forall j, 1 \leq j \leq k\},$$

- Update step: Calculate the new means of the observations in the new clusters.

$$m_i^{(t+1)} = \frac{1}{|S_i^{(t)}|} \sum_{x_j \in S_i^{(t)}} x_j$$

Over this POM every data owner can compute partial averages and the central server can merge them.

Kernel methods:

In the kernel methods such as SVMs, a predictive model contains training data (e.g. support vectors in the SVM algorithm). To keep training data privacy, we will make use of semiparametric approximations [Diaz_2016][Diaz_2017][Diaz_2018], where a group of centroids are selected (for example using the previous k-means algorithm to ensure privacy) and, after that, the weights are obtained using an stochastic gradient descent procedure.

5.1.2 POM2 (ATHOS)

In POM1 data information may be leaked to an honest-but-curious server since the server has access to the predictive model. In some use cases, data owners belong to the same company (e.g. different factories of the same company) and the server that orchestrates the training is in the cloud.

POM2 fixes that problem with two properties:

- No information is leaked to the server: POM2 leaks no information of participants to the honest-but-curious cloud server.
- The accuracy is kept intact compared to POM1: Achieves identical accuracy to a corresponding system trained using stochastic gradient descent.

The work proposed in [Aono_2018] shows that having access to the predictive model and to the gradients it is possible to leak information. Since the orchestrator has access to this information in POM1, if is not completely under our control (e.g. Azure or AWS cloud), POM2 solves the problem by protecting the gradients over the honest-but-curious cloud server.

POM2, as proposed in [Aono_2018] uses additively homomorphic encryption. All gradients are encrypted and stored on the cloud server and the additive property enables the computation across the gradients.

This protection of gradients against the server comes with the cost of increased computational and communication between the learning participants and the server.

POM2 operation description:

The different data owners jointly set up the public key pk and the secret key for an additively homomorphic encryption scheme (e.g. using the Paillier cryptosystem). The secret key sk is kept confidential from the server, but is known to all learning participants (e.g. manufacturing robots that belongs to the same company).

Every participant:

- Locally holds her/his datasets.
- Runs a replica of the predictive model.
- Establishes a secure channel to communicate the encrypted gradients to the server.

The server:

Recursively update the encrypted weight parameters in the encrypted domain. Since the operation that needs to be used to update the model is an addition, as in the case of the stochastic gradient descent, it can be done directly over the encrypted Paillier domain.

$$w := w - \eta \nabla Q(w) = w - \eta \sum_{i=1}^{n} \nabla Q_i(w)/n,$$

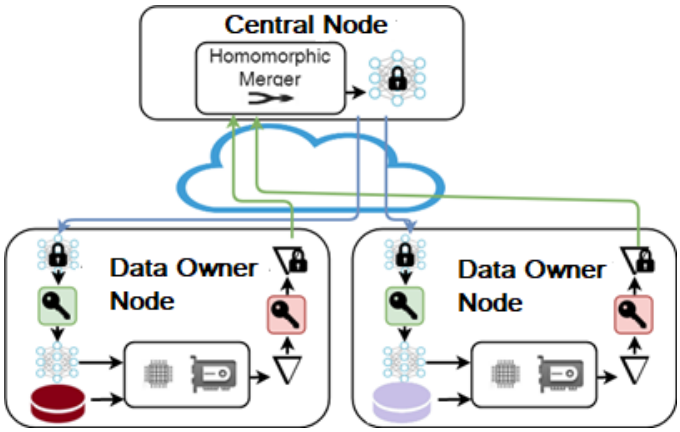

Figure 12 POM2 communication schema

Machine learning algorithms over this POM:

As in POM1 the function of the server is to compute the addition of the data set by the different data owners, and homomorphic encryption allow the addition operation, the ML algorithms described in POM1 can be directly implemented here.

5.1.3 POM3 (PORTHOS)

In POM2, every data owner trusts each other and they can share the private key of the homomorphic encryption (e.g. different servers with data that belongs to the same owner). Using the same key, every data owner uses the same encrypted domain. In many situations it is not possible to transfer the private key in a safe way.

POM3 is an extension of POM2 that makes use of a proxy re-encryption protocol to allow that every data owner can handle her/his own private key.

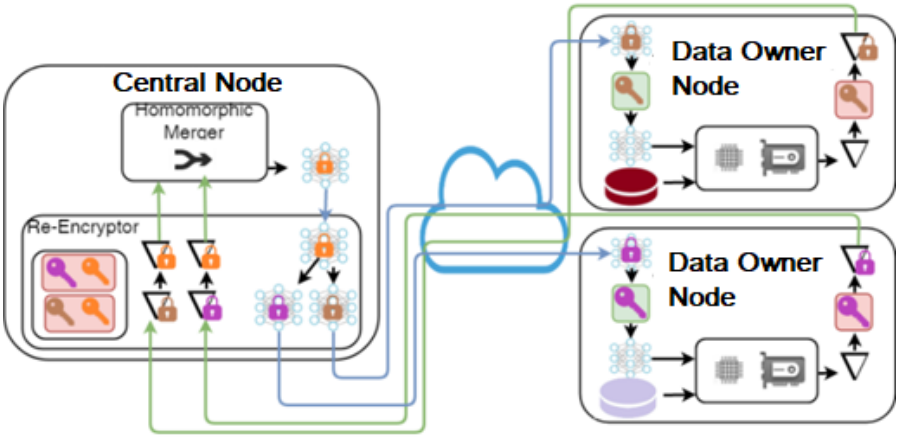

Figure 13 POM3 communication schema

5.2 Privacy Operation Modes in a Semi-Honest scenario

The main Machine Learning activity takes place on the central node side, and the protection of the resulting model is at a maximum. The server may ask the clients to compute some specific operations to complete the model estimation, but the end users never see the complete model during training (only at the end of the process if they ask or pay for it). The three POMs devised under this scheme are POM4 (Rochefort), POM5 (deWinter) and POM6 (Richelieu).

5.2.1 POM4 (ROCHEFORT)

Under this scheme, the IDP provides, what we call, Secure Cloud Services.

First, the IDP acts as a trusted generator of public parameters (PP) for multiple (public and secret) key encryption. From these PP, each participant (Users and Application) runs a KeyGen() algorithm that outputs the corresponding pair of public and secret keys. The IDP also generates a secret "master key", **MK**, and a global public key **PK** under which Applications perform the operations corresponding to a given Machine Learning Algorithm.

Second, the IDP offers private and secure storage services to Users. Additionally, the IDP transforms the Users' data (encrypted under their respective public-keys) into encryptions under the global public key **PK** without disclosing nor changing the underlying cleartext information (ReEncrypt()).

And third, the IDP assists Applications in those operations not supported by the homomorphic property of the cryptosystem. In order to protect the privacy of the encrypted operands, cryptographic blinding is applied before any outsourced computation. The following Figure illustrates the expected interactions.

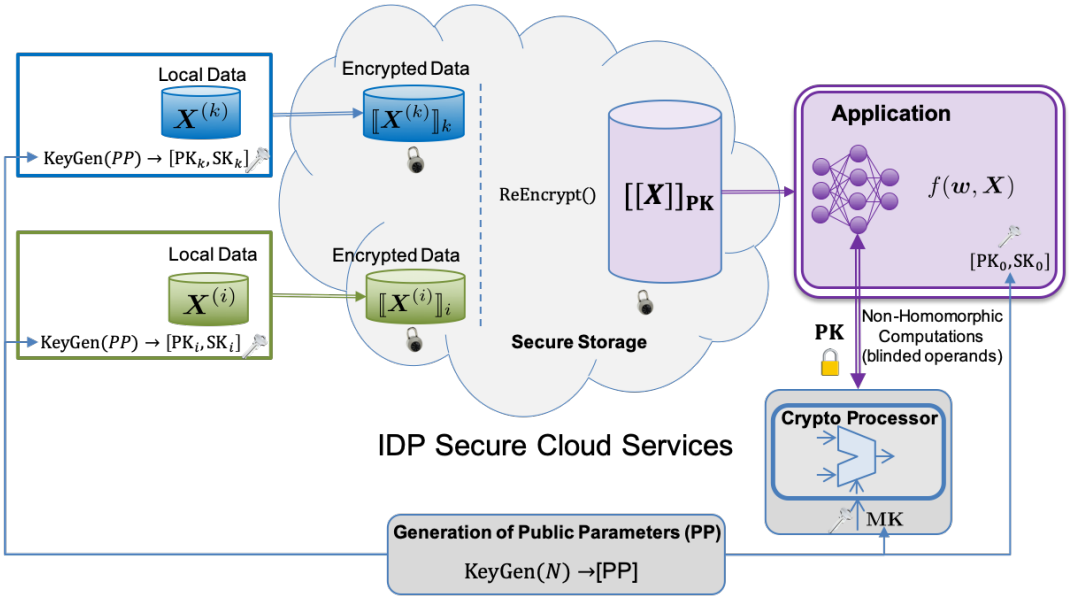

Figure 14 POM4 Communication schema

Regarding the security model, we will consider that POM4 and POM5 operate under the semi-honest adversarial model (honest-but-curious) [Goldreich_2004], meaning that all participants agree to follow a given protocol/algorithm, but trying to gather information about other parties' inputs, intermediate results, or overall outputs. This model uses blinding and randomization techniques to assure privacy protection and it is widely used in the privacy protection community. Nevertheless, if more aggressive attackers (malicious adversaries) participating in the computation are considered, any secure protocol in the semi-honest model can be transformed into a secure one against malicious adversaries using commitment schemes and zero-knowledge proofs. However, these techniques increase the interactions between the participants making slower the protocol (up to an order of ten [Lagendijk_2013]).

The POM4 description

In the POM4 we suggest using the BCP cryptosystem, by Bresson, Catalano and Pointcheval [Bresson_2013]. The BCP is additively homomorphic (i.e., it allows the addition of plaintexts

in the encrypted domain), and offers two independent decryption mechanisms: the first one depends on the secret key of each User and the second decryption mechanism depends on a master secret key, **MK**, that is stored on the IDP.

The Bresson, Catalano and Pointcheval (BCP) cryptosystem involves four algorithms:

1. **KeyGen**(N). Let N be a product of two primes p and q of same bitlength, where $p = 2p' + 1$ and $q = 2q' + 1$, for distinct primes p' and q'. Choose a random integer $g \in \mathbb{Z}_{N^2}^*$ of order $p \cdot p' \cdot q \cdot q'$ such that $g^{p' \cdot q'} mod\ N^2 = 1 + kN$, with $k \in [1, N-1]$. The master secret key is **MK** = $(p'; q')$, and the public parameters for user's key generation are given by the triplet PP = $(N;\ k;\ g)$. With these public parameters, each participant runs a private KeyGen() algorithm that outputs the public key $PK_i = g^{a_i} \bmod N^2$, where the random integer $a_i \in \mathbb{Z}_{N^2}$ is the secret key SK_i ($\mathbb{Z}_{N^2}$ is the set of integers modulo N^2 and $\mathbb{Z}_{N^2}^*$ is the group of invertible integers modulo N^2).
2. **Enc**(m, PP, PK_i). Given a plaintext $m \in \mathbb{Z}_N$, the encryption algorithm outputs the ciphertext $(A_i, B_i) = \left(g^{r_i} \bmod N^2, h_i^{r_i}(1 + m \cdot N) \bmod N^2\right)$, where = $r_i \in \mathbb{Z}_{N^2}$.
3. User decryption **Dec**(A_i, B_i, PP, SK_i). The plaintext message m is computed as $m = \left({B_i}/{A_i^{a_i}} - 1\ mod\ N^2 \right)/N$.
4. Master decryption **mDec**($A_i, B_i PK_i$, **MK**). First, compute $\tilde{a}_i = k^{-1} \cdot \frac{\left(h_i^{p' \cdot q'} - 1\ mod\ N^2\right)}{N} \bmod N$, Second, obtain $\tilde{r}_i = k^{-1} \cdot \frac{\left(A_i^{p' \cdot q'} - 1 \bmod N^2\right)}{N} \bmod N$. Then, compute $D_i = {B_i}/{(g^{\gamma_i})} \bmod N^2$, where $\gamma_i = \tilde{a}_i \cdot \tilde{r}_i \bmod N$. Finally, the plaintext message is $m = \delta \left(D_i^{p' \cdot q'} - 1 \bmod N^2\right)/N$,where $\delta = (p' \cdot q')^{-1} \bmod N$.

Detailed Operation

Set up

The IDP sets up the selected cryptosystem and distributes the public encryption parameters PP to Users and Application. Users, with their KeyGen() primitives, generate their respective public and secret keys PK_i and SK_i. Then, they encrypt the data, and send the encrypted information to the IDP's Cloud Store.

The Application is considered as another user, having its own pair of public and secret keys (PK_0, SK_0).

Re-encryption

After collecting the multiple key encrypted data, the IDP runs a Secure Protocol to re-encrypt the data into new encryptions under the global public key **PK**$= \prod_i \quad PK_i$. The re-encryption under **PK** is performed without disclosing nor changing the underlying information [Peter_2013].

Secure Learning

The IDP will assist the Application, with no collusion between them, in a limited set of (previously agreed) secure computations on encrypted data not supported by the partially homomorphic cryptosystem.

In particular, we propose to use a minimal set of instructions that includes the other operation that preserves the ring structure of input plaintexts, which in the case under consideration (BCP cryptosystem) is the modular multiplication ($a \cdot b \, mod \, N$), a few basic logical (bit-shifting) and conditional branching instructions (comparison).

This tightly limited set of instructions could be executed directly on the special-purpose hardware that implements the encryption/decryption operations (CryptoProcessor [Bossuet_2013, Sakiyama_2007]).

With these transformed ciphertexts (under the same public key), and making use of the homomorphic property and the reduced set of instructions available at the CryptoProcessor, it is possible to design a secure protocol that encodes a given high-level cleartext machine learning algorithm into the adequate sequence of low-level instructions in the encrypted domain for the particular cryptographic hardware.

Result Retrieval

Once the secure learning phase is completed, the output of the function (encrypted under **PK**) is ready to be sent back to the Application. And to do this, the Application runs a final SMC protocol with IDP in order to transform the output back into encryptions under the Application's public key PK_0.

5.2.2 POM5 (deWINTER)

In POM5 the sensitive data is encrypted, stored and processed in the users' facilities. In order to deal with the processing of multiple-key encrypted data, we will borrow some ideas from what is known as "proxy re-encryption".

Proxy re-encryption (PRE) is a special type of public-key encryption that permits a "proxy" (or a user) to transform ciphertexts from one public key to another, without the proxy being able to learn any information about the original message. In more detail, PRE enables us to convert a ciphertext under public key PK_f ("f" denotes "from") into another ciphertext under public key PK_t ("t" denotes "to") by using a re-encryption key $RK_{f \to t}$ without decrypting the original ciphertext by the secret key SK_f. PRE is a very useful cryptographic primitive that has been used to achieve encrypted email forwarding [Blaze_1998], encrypted file storage [Ateniese_2005], and secure payment systems for credit cards [Gaddam_2019].

In our POM5 scenario, the re-encryption keys would permit to implement a Machine Learning algorithm in a sequential way: first, the Application adjusts the parameters of its model interacting with the first user (user #1); once the full dataset of user #1 has been processed, he/she issues a re-encryption key to the Application, which obtains the cleartext version of the partially updated model; then, the Application interacts with user #2 to update again its model; etc...

The POM5 cryptosystem

The cryptosystem suggested for POM5 combines elements taken from the well-known ElGamal cipher [ElGamal_1985], and from Bilinear Maps on Elliptic Curves [Joux_2000, Ateniese_2005].

The security of the ElGamal cryptosystem comes from the fact of the difficulty of finding the discrete logarithm in a cyclic group. The discrete logarithm problem (DLOG) can be stated as follows: it is very easy to compute h=g^x for a given x, but very hard to find x given h.

Basically, ElGamal cryptosystem takes a large prime p (preferably of the form 1+2q where q is also prime), a generator g of Z_q, and randomly choose x, $1 \leq x \leq p-1$, and calculate y = g^x mod p. The public parameters (or Public Key) is the triplet y, p, and g, and the Secret Key is x.

For the encryption, ElGamal cipher picks a random r and let $\alpha = g^r \bmod p$, and $\beta = m \cdot y^r \bmod p$, where m is the cleartext message. The ciphertext is given by [[m]] = (α, β).

Decryption can be done if the Secret Key x is known (in addition to [[m]] =(α,β)) as follows: $\beta/(\alpha^x) = m \cdot y^r /(g^r)^x = m \cdot (g^x)^r/(g^r)^x = m \cdot g^{(rx)}/g^{(rx)} = m$, where all operations are taking modulo p.

Bilinear Maps gained great popularity in cryptography in 2001 when Boneh and Franklin used them for Identity-Based Encryption [Boneh_2001]. A bilinear map can be defined as follows.

Let G_1, G_2, G_3 be cyclic groups of prime order q. The function e:G_1 x G_2→ G_3 is a bilinear map if for all g_1∈G_1, g_2 C, and a, b∈Z_q, e(g_1^a, g_2^b) = [e(g_1,g_2)]^ab.

POM5 Operation

The basic cryptographic primitives required for the POM5 operation can be summarized as follows:

- **IDP: Generation of Public Parameters**
  $< g >= \mathbb{G}_1$ of prime order q.
  $Z = e(g, g)$
- **Alice and Bob: KeyGen() (Alice ⇔ Application; Bob ⇔ User)**
  $SK_a = a \in \mathbb{Z}_q^*$, randomly selected. $SK_b = b \in \mathbb{Z}_q^*$, randomly selected.
  $PK_a = g^a$ $PK_b = g^b$
- **Encryption**:
  $m \in \mathbb{G}_2$. random $r \in \mathbb{Z}_q^*$
  $C_a = (Z^r \cdot m,\ g^{ra})$
- **Decryption (Alice)**:
  $m = \frac{Z^r \cdot m}{e(g^{ra}, g^{1/a})} = \frac{Z^r \cdot m}{Z^r}$
- **Re-encryption**:

$$\begin{aligned} C_a &= (Z^r \cdot m,\ g^{ra}) \\ C_b &= (Z^r \cdot m,\ e(g^{ra}, RK_{A \to B})) \\ &= (Z^r \cdot m,\ e(g^{ra}, g^{b/a})) \\ &= (Z^r \cdot m,\ Z^{rb}) \end{aligned}$$

- **Decryption (Bob)**:
  $m = \frac{Z^r \cdot m}{(Z^{rb})^{1/b}}$

Taking into account the previous operations, the final block diagram for POM5 is represented in the next Figure.

As in POM4, POM5 has three participants: the Users, which own private encrypted data, the Application, which owns the ML model, and the IDP Server, which issues the Public Parameters (PP) of the (multiplicative homomorphic) ElGamal Cryptosystems, and gives support to the non-homomorphic operation (which in this case is the Sum).

Initially, the IDP outputs the PP (generator g, bilinear map e, and Z=e(g,g)).

Then, the Application, Users and the IDP itself generate the Public and Secret Keys of their respective cryptosystems (Application, (PK_M, SK_M); Users, (PK_i,SK_i), i=1,..., N; IDP Server, (PK_S, SK_S)).

In the first place, the Application interacts with User 1. First, it requests a re-encryption key (RK_{M→1}} to operate under PK_1. Then, it sends to User 1 the ML algorithm as a sequence of basic arithmetic operations: in the case of the multiplications, they can be performed using the homomorphic property of the ElGamal cryptosystem; but in the case of additions, a Secure Sum protocol must be implemented with the assistance of the IDP (which also request a re-encryption key from User 1, RK_{S→1}, to produce a blinding factor under PK_1). Once the ML code has been completed, the Application deciphers the parameters of its partially adjusted model by using the re-encrypting key RK_{1→M}.

This protocol is repeated N times, where N is the number of participant Users.

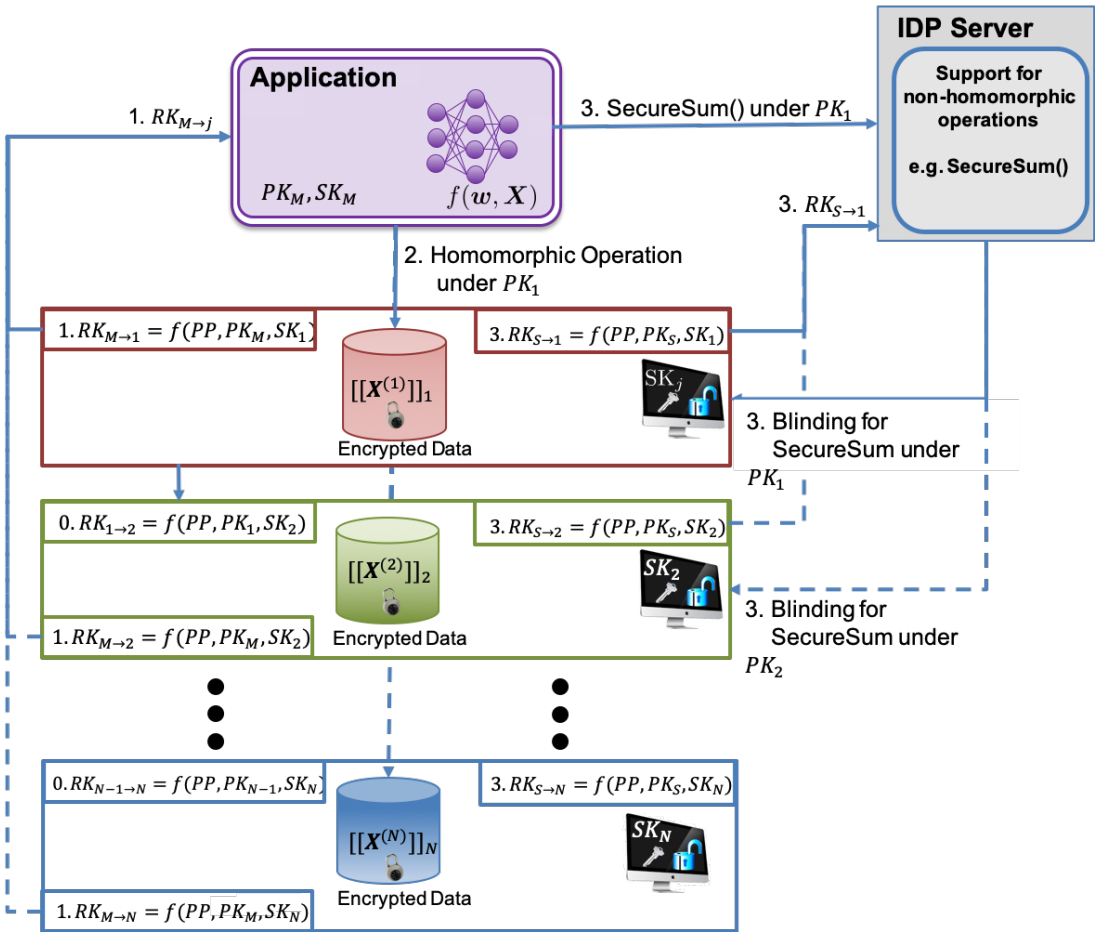

Figure 15 POM5 Communication schema

5.2.3 POM6 (RICHELIEU)

Privacy Preserving Machine Learning (PPML) is always under a trade-off between acceptable privacy level and computational/communication cost (which ultimately determines its real-world applicability). The ideal scenario is that of fully outsourcing all of the computations to a cloud service, such that users, after protecting their data (by means of encryption, for instance) they can relax and even go offline while a cloud service carries out all of the computations before producing the final model. This described behaviour is the "holy grail" of the distributed private machine learning but, unfortunately, it has not been implemented yet to a practical level. Outsourcing the whole raw dataset requires higher security measures than strictly needed (since in the fully outsourced data scenario many attacks are possible) and compromises the viability of the ML task due to the increased computational cost.

A feasible and practical alternative to data encryption is that provided by the Differential Privacy approaches, that rely on modifications of the raw data (by transformations or noise addition) before outsourcing them such that no information is revealed about every pattern from a statistical point of view [Dinur_2003][Dwork_2006]. The most obvious limitation of this

approach is that it modifies the original data and, therefore, the expected final performance of the model can be reduced.

As an alternative, we can resort to distribute computations over several participating nodes, such that only when all the partial computations are combined in every node, the final result is obtained. This is the approach proposed by the Secure Multiparty Computing (SMC) in its Information Theoretic instantiation. One of the best-known implementations relies on the Shamir Secret Sharing approach [Shamir_1979]. The main drawback of this approach is that we have to control the level of end user collusion, i.e., always verify that the end users do not communicate any extra information than the required by the SMC protocol, otherwise, the private information could be revealed. This assumption is not always easy to hold/guarantee. Also, the required protocols are very communication intensive.

Another SMC approach relying on encryption is stated as Secure Function Evaluation (SFE), where data is encrypted by the owner, and high privacy restrictions are imposed such that no intermediate result can be revealed apart from the output of the function itself. This approach has been inspired by the work of Yao on garbled circuits [Yao_1986], that allow to securely compute any function that can be represented as a boolean circuit, usually consisting of XOR and AND gates. The transformation of any function into suitable circuits is not a trivial task, and the associated computational and communication costs are usually high.

Another perspective on SMC is that of Fully Homomorphic Encryption (FHE). Such relationship has been established in [Choudhury_2013], where it is shown that both schemes tend to be equivalent by exchanging communication steps by computation. If we adopt a Homomorphic Encryption (HE) scheme, then (some) operations can be directly obtained on encrypted data [Rivest_1978]. In the Partial Homomorphic Encryption (PHE) schemes, the operation on encrypted data is usually restricted to the operations supported by the homomorphic scheme (unlimited number of operations: either sum or multiplication, but not both simultaneously). More elaborated schemes exist that provide unlimited operations of one type (sums) and a limited number of operations of the other type (products), which are known as SomeWhat Homomorphic Encryption (SWHE). The ideal approach is that of Fully Homomorphic Encryption (FHE), that allows an unlimited number of operations thanks to the bootstrapping operation [Gentry_2009]. Unfortunately, this approach implies a huge amount of extra computation and increased code lengths that hamper its use in many practical applications. Although impressive advances have been achieved in this direction during the last years, the state-of-the-art techniques are far from being practical in real world scenarios.

It is not easy to find a good equilibrium point among all the possibilities in the PPML ecosystem. Under POM6 we propose to achieve a reasonable compromise solution between data outsourcing, privacy of intermediate results, computational cost, and end user active participation during the training process, such that practical implementations are possible. We will

also assume an honest-but-curious (semi-honest) scenario, where all participants follow the protocols, but yet the information must be protected from each other's curiosity. The communication model will be either a star, with the central node being the connection among the end users, or a chain, where a message sequentially visits all end users, the central node being the initial/final point.

We propose to protect privacy by aggregating operations on the data, such that no individual training data is transferred outside the owner facilities (no identification of any singular pattern is possible). Such aggregation takes place on the end user side, no raw data is transferred outside the end user facilities. The ML model training takes place in the central node by using the aggregated information from the end users. The end users must declare if some of the aggregation operations (dot products, covariance matrices, etc.) are acceptable for his/her privacy and do not represent a threat to the raw data security, under a disclosure risk / data utility trade-off scheme [Duncan_2001][Loukides_2011]. The cooperation of the end users is necessary during training, but this is not a problem since the MUSKETEER client running on the end users' side will always be active.

After a preliminary exploration of possibilities, we have observed that, depending on the required privacy level, the computational cost may largely vary. We will consider the following situations that lead to different POM6 implementations (with different computational requirements):

- **Public model (PM)**: the model is known to all participants, in the same line as the Federated approaches of previous POMs
- **Secret model (SM)**: the model is only known to the central node. This option represents a significant advantage over Federated schemes, where the model is sent to all nodes and therefore cannot be kept secret.
- **Secret input data**: the input features are always kept secret, only revealing averages or covariances to the central node
- **Public targets (PT):** under some circumstances, the targets (supervised scenario) could be revealed to the central node, which largely speeds up the training procedure.
- **Secret targets (ST)**: the targets (supervised scenario) are kept secret.

Anyhow, the user will always be in position of deciding which privacy level is adequate for a given task.

The algorithms under POM6 will be completely modular, such that any operation can be transparently exchanged by another with the same characteristics, without affecting the algorithm. This is a very desirable characteristic that will facilitate the improvement or replacement of

specific operations. The algorithms will be built upon a series of basic privacy preserving operations on the data, to name a few:

- secure dot product: this is a basic operation especially relevant to linear models. We could use any of the available protocols, for instance:
  - protocols relying on homomorphic encryption, such as [Zhang_2017]
  - protocols relying on SMC, such as [Zhu_2015]
- secure matrix multiplication: under some circumstances it is possible to obtain the matrix product of two matrices without revealing any of them to the other party. This approach is especially relevant to implement algorithms under a vertical partition scheme. [Karr_2009]
- iterated reweighted least squares procedures. Useful to transform non Least-Squares cost functions into an iterated Least Squares formulation. [Barreto_1994]

By a careful design and combination of the above-mentioned procedures, we foresee to start implementing the following machine learning models under POM6, although a deeper analysis is necessary before extending to more complex models:

- Logistic classifier
- Robust Ridge regression with Huber cost (outlier resistant)
- k-means clustering
- Support Vector Machines (linear and kernel based)
- Support Vector regression (linear and kernel based)

Simple ML training example under POM6

To help to understand how ML training will occur in MUSKETEER, we will illustrate here a simple example: we will solve a regression task using a linear model trained under a Mean Squared Error (MSE) cost function, to be adjusted using training data from 5 users. The user **end_user_id** = 1 will provide training patterns ($\underline{\mathbf{x}}^{(1)}_i$, $\mathbf{y}^{(1)}_i$), i = 1, …, N1, and so on for the other users.

Therefore, given an input data $\underline{\mathbf{x}}^{(1)}_i$, the model output is[4]:

$$o^{(1)}_i = \underline{\mathbf{w}}^T \underline{\mathbf{x}}^{(1)}_i$$

[4] Without loss of generality, we assume that the model bias is included as the first element of $\underline{\mathbf{w}}$, such a formulation is equivalent to using an extra input feature with constant value of 1. This is a common practice in many ML models.

and the associated error is:

$$e^{(1)}_i = (y^{(1)}_i - o^{(1)}_i)$$

The corresponding SSE cost at every user is:

$$C^{(1)}(\underline{\mathbf{w}}) = \sum_{i=1}^{N1} (e^{(1)}_i)^2 = \sum_{i=1}^{N1} (y^{(1)}_i - \underline{\mathbf{w}}^T \underline{\mathbf{x}}^{(1)}_i)^2$$

and we want to minimize the global MSE cost:

$$C(\underline{\mathbf{w}}) = \frac{1}{N1+N2+N3+N4+N5}(C^{(1)}(\underline{\mathbf{w}}) + C^{(2)}(\underline{\mathbf{w}}) + C^{(3)}(\underline{\mathbf{w}}) + C^{(4)}(\underline{\mathbf{w}}) + C^{(5)}(\underline{\mathbf{w}}))$$

The central node asks the end users to compute some local data aggregation, in this case, the sum of outer products of the feature vectors and targets, such that at **end_user_id** = i, we need to compute:

$$\mathbf{R}_{xx}^{(i)} = \underline{\mathbf{X}}^{(i)T} \underline{\mathbf{X}}^{(i)}$$

$$\mathbf{r}_{xy}^{(i)} = \underline{\mathbf{X}}^{(i)T} \mathbf{y}^{(i)}$$

and send them back to the central node. The central node collects these data matrices and vectors from all end users and it computes:

$$\mathbf{R}_{xx} = \mathbf{R}_{xx}^{(1)} + \mathbf{R}_{xx}^{(2)} + \mathbf{R}_{xx}^{(3)} + \mathbf{R}_{xx}^{(4)} + \mathbf{R}_{xx}^{(5)}$$

$$\mathbf{r}_{xy} = \mathbf{r}_{xy}^{(1)} + \mathbf{r}_{xy}^{(2)} + \mathbf{r}_{xy}^{(3)} + \mathbf{r}_{xy}^{(4)} + \mathbf{r}_{xy}^{(5)}$$

$$\underline{\mathbf{w}}^* = \mathbf{R}_{xx}^{-1} \mathbf{r}_{xy}$$

To evolve from this simple scheme for training a linear regression model under a Least Squares cost function, at some point it is necessary to compute estimation errors at every node. If the model $\underline{\mathbf{w}}$ can be freely distributed to all the nodes, then they can locally compute such errors. If the model is expected to be private and cannot be sent to the end users, then the errors can be obtained after the application of Secure Dot Product (SDP) or Secure Matrix Multiplication (SMM) protocols, as described earlier [Zhang_2017], [Zhu_2015], [Karr_2009]. The combination of local errors with local targets allows to obtain weighting values to transform many cost functions into Least Squares ones under the Iterated Reweighted Least Squares approach [Barreto_1994], such that more elaborate models can be obtained at the central node. Although some preliminary -successful- experiments have already been conducted in laboratory conditions, we expect to develop ML algorithms under the POM6 scheme ready to be integrated in the MUSKETEER platform during the next months.

5.3 Unrestricted Data-Sharing POMs:

To complete the list of POMs, we have included other approaches where users do not have any restriction concerning data sharing, nor with respect to data privacy. We include modes

where data can leave the users' facilities (using secure communication protocols) and remain unencrypted in the cloud and on the client side (using secure access mechanisms).

These POMs work under the assumption that every user will respect the privacy rules and use it properly. However, the data sovereignty will be strictly supervised and the data transfer will be secure using the protocols standardized by the Industrial Data Space Association (IDSA).

5.3.1 POM7 (PLANCHET)

Traditional cloud computing schema where every data owner can store their datasets in the cloud and predictive models can be trained, with the possibility of selectively sharing with the users the resulting models. The IDSA standards will be follow for authentication and data transfer.

5.3.2 POM8 (DARTAGNAN)

Traditional local computing, where data consumers can download different datasets to locally train predictive models. We will make use of the IDSA mechanisms for authentication and data transfer.

6 Conclusion

The present report presents an overview of the privacy operation modes in the MUSKETEER project. Despite the coverage is far from being exhaustive, we show how these POMs are supported by the current academic state-of-art. The report also provides information about how to implement ML algorithms over every POM. Although some design criteria and POMs can be modified along the project lifecycle, this will be out starting point to develop the libraries.

7 References

[Aono_2018] Aono, Yoshinori, et al. "Privacy-preserving deep learning via additively homomorphic encryption." *IEEE Transactions on Information Forensics and Security* 13.5 (2018): 1333-1345.

[Armknecht_2015] Armknecht, Frederik, et al. "A Guide to Fully Homomorphic Encryption." IACR Cryptology ePrint Archive 2015 (2015): 1192.

[Ateniese_2005] Giuseppe Ateniese, Kevin Fu, Matthew Green, and Susan HohenBerger. Improved proxy re-encryption schemes with applications to secure distributed storage. In NDSS 2005. The Internet Society, February 2005.

[Banko_2001] Banko, M., & Brill, E. (2001, July). Scaling to very very large corpora for natural language disambiguation. In Proceedings of the 39th annual meeting on association for computational linguistics (pp. 26-33). Association for Computational Linguistics.

[Barreto_1994] Barreto, J.A. & Burrus, Charles. (1994). Iterative reweighted least squares and the design of two-dimensional FIR digital filters. 775 - 779 vol.1.

[Bellafqira_2017] Bellafqira, Reda, et al. "Sharing Data Homomorphically Encrypted with Different Encryption Keys." arXiv preprint arXiv:1706.01756 (2017).

[Blaze_1998] Matt Blaze, Gerrit Bleumer, and Martin Strauss. Divertible protocols and atomic proxy cryptography. In Kaisa Nyberg, editor, EUROCRYPT'98, volume 1403 of LNCS, pages 127–144. Springer, Heidelberg, May / June 1998.

[Boneh_2001] Boneh, Dan, Franklin, Matt, 2001. Identity-based encryption from the Weil pairing. In: Advances in Cryptology—CRYPTO 2001, Chiu, pp. 213–229.

[Bossuet_2013] L. Bossuet, M. Grand, L. Gaspar, V. Fischer, G. Gogniat, Architectures of flexible symmetric key crypto engines survey: From hardware coprocessor to multi-crypto-processor system on chip, ACM Computing Surveys(CSUR) 45 (4) (2013) 41.

[Bresson_2003] E. Bresson, D. Catalano, D. Pointcheval, A simple public-key cryptosystem with a double trapdoor decryption mechanism and its applications, in: Advances in Cryptology-ASIACRYPT 2003, Springer, 2003, pp. 37-54.

[Diaz_2016] Díaz-Morales, R., & Navia-Vázquez, Á. (2016). Improving the efficiency of IRWLS SVMs using parallel Cholesky factorization. Pattern Recognition Letters, 84, 91-98.

[Diaz_2017] Díaz-Morales, R., & Navia-Vázquez, Á. (2017). LIBIRWLS: A parallel IRWLS library for full and budgeted SVMs. Knowledge-Based Systems, 136, 183-186.

[Diaz_2018] Díaz-Morales, R., & Navia-Vázquez, Á. (2018). Distributed Nonlinear Semiparametric Support Vector Machine for Big Data Applications on Spark Frameworks. IEEE Transactions on Systems, Man, and Cybernetics: Systems.(Accepted, In press)

[Deng_2008] Deng, J. Weng, S. Liu, and K. Chen, "Chosen-ciphertext secure proxy re-encryption without pairings," in International Conference on Cryptology and Network Security. Springer, 2008, pp. 1–17.

[Dinur_2003] Dinur, Irit & Nissim, Kobbi. (2003). Revealing Information while Preserving Privacy. Proceedings of the Twenty-second ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems. 202-210.

[Dodis_2003] Y. Dodis and A. Ivan, "Proxy cryptography revisited," in Proceedings of the Tenth Network and Distributed System Security Symposium, 2003, pp. 514–532

[Duncan_2001] Duncan, George & A. Keller-McNulty, Sallie & Stokes, Lynne. (2001). Disclosure Risk vs. Data Utility: The RU Confidentiality Map.

[Dwork_2006] Dwork, Cynthia. (2006). Differential Privacy. Proc. ICALP'06 33rd international conference on Automata, Languages and Programming - Volume Part II, Pages 1-12, Venice, Italy. July 10 - 14, 200610.1007/11787006_1.

[ElGamal_1985] ElGamal T. (1985). A public key cryptosystem and a signature scheme based on discrete logarithms. IEEE transactions on information theory, 31(4), 469-472.

[Gaddam_2019] Sivanarayana Gaddam, Rohit Sinha, and Atul Luykx. Applying proxy-re-encryption to payments. Real World Crypto 2019, 2019. https://rwc.iacr.org/2019/slides/Applying_PRE_Payments.pdf.

[Gentry_2009] Craig Gentry. 2009. A fully homomorphic encryption scheme. Ph.D. Dissertation. Stanford University.

[Goldreich_2004] O. Goldreich, Foundations of Cryptography II, Cambridge University Press,Cambridge, UK, 2004.

[Han_2015] Han, Song, Huizi Mao, and William J. Dally. "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding." arXiv preprint arXiv:1510.00149 (2015).

[Joux_2000] Antoine Joux. A one round protocol for tripartite Diffie–Hellman. In Wieb Bosma, editor, Algorithmic Number Theory — ANTS-IV, volume 1838 of Lecture Notes in Computer Science, pages 385–393, Berlin, 2000. Springer-Verlag.

[Karr_2009] Karr, Alan & Lin, Xiaodong & P. Reiter, Jerome & P. Sanil, Ashish. (2004). Privacy Preserving Analysis of Vertically Partitioned Data Using Secure Matrix Products. J. Off. Statist.. 25.

[Konečný_2016] Konečný, Jakub, et al. "Federated optimization: Distributed machine learning for on-device intelligence." arXiv preprint arXiv:1610.02527 (2016).

[Lagendijk_2013] R. Lagendijk, Z. Erkin, M. Barni, Encrypted signal processing for privacy protection: Conveying the utility of homomorphic encryption and multiparty computation, Signal Processing Magazine, IEEE 30 (1) (2013)82-105. doi:10.1109/MSP.2012.2219653.

[Lin_2017] Lin, Yujun, et al. "Deep gradient compression: Reducing the communication bandwidth for distributed training." arXiv preprint arXiv:1712.01887 (2017).

[Loukides_2011] Loukides G., Gkoulalas-Divanis A., Shao J. On balancing disclosure risk and data utility in transaction data sharing using R-U confidentiality map. Joint UNECE/Eurostat work session on statistical data confidentiality (Tarragona, Spain, 26-28 October 2011)

[McMahan_2017] McMahan, Brendan, and Daniel Ramage. "Federated learning: Collaborative machine learning without centralized training data." Google Research Blog 3 (2017).

[Mehnaz_2017] Mehnaz, Shagufta & Bellala, Gowtham & Bertino, Elisa. (2017). A Secure Sum Protocol and Its Application to Privacy-preserving Multi-party Analytics. 219-230.

[Paillier_1999] Paillier, Pascal. Public-Key Cryptosystems Based on Composite Degree Residuosity Classes. Paillier, Pascal. J. Stern (Ed.): EUROCRYPT'99, LNCS 1592, pp. 223{238,10.1007/3-540-48910-X_16.

[Peter_2013] A. Peter, E. Tews, S. Katzenbeisser, Efficiently outsourcing multiparty computation under multiple keys, Information Forensics and Security, IEEE Transactions on 8 (12) (2013) .doi:10.1109/TIFS.2013.2288131.

[Regev_2005] Regev O. (2005). On Lattices, Learning with Errors, Random Linear Codes, and Cryptography. Journal of the ACM (JACM). 56. 84-93. 10.1145/1568318.1568324.

[Rivest_1978] R. Rivest, L. Adleman, and M. Dertouzos, On data banks and privacy homomorphisms, In: Foundations of Secure Computation, New York: Academic Press, pp.169{179, 1978.

[Sakiyama_2007] K. Sakiyama, L. Batina, B. Preneel, I. Verbauwhede, Multicore curve-based cryptoprocessor with reconfigurable modular arithmetic logic units over GF(2^n), IEEE Transactions on computers (9) (2007) 1269-1282.

[San_2016] San, Ismail, et al. "Efficient paillier cryptoprocessor for privacy-preserving data mining." Security and communication networks 9.11 (2016): 1535-1546.

[Shamir_1979] Shamir, Adi (noviembre de 1979). «How to share a secret». Communications of the ACM 22 (11). ISSN 0001-0782., pp. 612-613

[Shokri_2015] Shokri, Reza, and Vitaly Shmatikov. "Privacy-preserving deep learning." Proceedings of the 22nd ACM SIGSAC conference on computer and communications security. ACM, 2015.

[Wang_2018] Wang, Shiqiang, et al. "When edge meets learning: Adaptive control for resource-constrained distributed machine learning." IEEE INFOCOM 2018-IEEE Conference on Computer Communications. IEEE, 2018.

[Yao_1986] Andrew C. Yao. How to generate and exchange secrets. In Proc. 27th IEEE Symp. on Foundations of Comp. Science, pages 162–167, Toronto, 1986.

[Zhu_2011] Zhu, Youwen & Liusheng, Huang & Wei, Yang & Xing, Yuan. (2011). Efficient Collusion-Resisting Secure Sum Protocol. Chinese Journal of Electronics. 20.

[Zhu_2015] Youwen Zhu, Tsuyoshi Takagi. Efficient scalar product protocol and its privacy-preserving application. Int. J. Electronic Security and Digital Forensics, Vol. 7, No. 1, 2015