Big Data Privacy by Design Computation Platform

Extended Abstract

Rui Nuno Lopes Claro
Instituto Superior Técnico, Universidade de Lisboa
Lisbon, Portugal
rui.claro@tecnico.ulisboa.pt

ABSTRACT

We live in the age of Big Data. Personal user data, in particular, are 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 are not made explicit to the users, but they are central to the business model of the companies. However, the right to privacy of each individual 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 regarding 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 to encapsulate algorithms that can, for example, monitor the vital signs of patients (without exposing the data to other people), produce real-time recommendations based on location (without disclosing the location to others), etc. This proof-of-concept implemented privacy-preserving versions of Machine Learning algorithms and compared them against a baseline reference, allowing a better understanding of the trade-offs of using privacy-preserving technology.

KEYWORDS

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

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]. 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 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 [8].

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[2]. Due to these breaches, and despite the value that Data Mining (DM) adds to businesses and medical systems, consumers show an increasing concern in the privacy threats posed by it [2]. 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 DM, a sub-field known as Privacy-preserving Data Mining (PPDM) has been gaining influence over the last years [3]. The objective of PPDM is to guarantee the privacy of sensitive information, while at the same time preserve the utility of the data for knowledge learning purposes [1]. This can be achieved by using one or more privacy-preserving techniques, such as Garbled Circuits (GC) or Homomorphic Encryption (HE) [3].

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 DM is *Classification* [7], 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. For example, DeepMind (Google) building ML algorithms to process admissions in hospitals[3].

By combining ML algorithms and privacy-preserving techniques, it is possible to create DM 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 [3].

In this work, we present 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

[1] http://www.vcloudnews.com/every-day-big-data-statistics-2-5-quintillion-bytes-of-data-created-daily/

[2] https://www.networkworld.com/article/2185187/security/15-worst-internet-privacy-scandals-of-all-time.html

[3] https://deepmind.com/applied/deepmind-health/

platform without them having to build their own, meaning fewer maintenance costs derived from maintaining code developed or dedicated servers. Since the platform has its foundations on privacy-preserving techniques, it addresses satisfactorily the privacy demands that those users want for their data.

We provide a detailed comparison of four ML algorithms (DT, *k*-M, LR and SVM) combined with two privacy-preserving techniques (GC and HE), showing when to use each combination, depending on the context of data and on the operations done by the algorithms.

We performed an experimental evaluation by implementing the privacy-preserving algorithms and evaluating them using publicly available datasets, and comparing the results with a baseline. We show the overhead created by using privacy-preserving techniques, both in terms of runtime and communication costs.

2 BACKGROUND AND RELATED WORK

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 limits the access that other entities, being them the government or private companies, have to personal data.

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.

DM is the process of discovering interesting patterns and knowledge from large amounts of data [5]. We divided the DM process according to the widely used CRISP-DM model [11].

To enforce privacy in the processing of data, we used privacy-preserving techniques, namely GC and HE, that we now detail.

Garbled Circuits [12] allow two mutually mistrusting parties to evaluate a function over their private inputs without resorting to a trusted third party. In other words, GC allow two parties holding inputs 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.

Homomorphic Encryption [9] 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 Encryption (PHE), Where there is only one operation that is allowed by the homomorphic property; and Fully Homomorphic Encryption (FHE), where it is possible to make two different operations on the ciphertext, namely addition and multiplication.

In this work we will focus on using these two techniques in combination with different ML algorithms to build classifiers for publicly available data.

Table 1: The datasets used in the evaluation.

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

[a]https://www.kaggle.com/uciml/pima-indians-diabetes-database
[b]https://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+(Diagnostic)
[c]http://archive.ics.uci.edu/ml/datasets/credit+approval
[d]https://archive.ics.uci.edu/ml/datasets/adult

3 IMPLEMENTATION

When building a platform for Privacy-preserving ML, we must consider not only the traditional steps in data processing, but also have an increased care when preprocessing data to incorporate the cryptographic techniques.

Datasets

For running the experiments, we used datasets that are highly used in the literature. These datasets are detailed in Table 1 and represent three of the most important subjects in DM.

Preprocessing

Although our data is obtained from publicly available data sources, some preprocessing on the data was still necessary. We used two techniques for this, One-hot Encoding [6] and Feature Scaling.

One-hot Encoding. In ML, One-hot Encoding is used to deal with the problems created in using datasets with categorical data, since most ML algorithms require numerical data. This technique allows to circumvent this problem, creating additional binary variables for each unique value. For

example, 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, if the original sample has the value "dog", the resulting three binary values will be $(1, 0, 0)$,

Feature Scaling. For some ML algorithms, having a broad range of values in one of the features may cause this particular feature to govern the modeling. We used *rescaling* to prevent this problem, by scaling the values to a range between $[-1, 1]$.

We also split the datasets into a training, validation and testing set, using a proportion of 70/15/15, respectively. 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.

Baseline

A baseline was implemented using scikit-learn for Python, so that meaningful comparisons could be achieved. For that, we implemented the following learning models using the toolkit:

Decision Trees (DT). A decision support tool that uses a tree-like graph to represent decisions and possible outcomes of those decisions [10]. DT are composed of nodes and leaves, with each node representing the decisions to take (i.e., thresholds that a feature is compared against), and each leaf representing class labels. The classification process of a sample in a DT is accomplished by traversing the tree from the top, comparing the features selected on each node with its respective threshold, and choosing one branch or the other accordingly, repeating the process until a leaf is reached. The class label of this leaf is assigned to the sample, ending the classification process.

Support Vector Machines (SVM). A supervised learning model used for classification and regression analysis. A SVM model represents the samples as points in space, mapped so that the margin between the two classes is as wide as possible. The vectors that define this margin, or *hyperplane*, are called Support Vectors (SVs). The linear classification of SVM depends on the data, i.e., if the data is linearly separable or not. In the case of a *hard margin*, the hyperplanes are selected so that the distance between then is as large as possible. In the case of a *soft margin*, we minimize the following *hinge* loss function:

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

where the parameter λ determines the trade off 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. this function can be Linear: $k(\vec{x}_i, \vec{x}_j) = (\vec{x}_i \cdot \vec{x}_j)$, Polynomial: $k(\vec{x}_i, \vec{x}_j) = (\vec{x}_i \cdot \vec{x}_j)^d$, or a Radial Basis Function (RBF): $k(\vec{x}_i, \vec{x}_j) = \exp(-\gamma \|\vec{x}_i - \vec{x}_j\|^2)$.

The prediction of a SVM is done according to the following equation:

$$f(x) = \sum_{i=1}^{m} \alpha_i K(x_{SV}^{(i)}, x) + b \quad (2)$$

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.

k-Means (k-M). A clustering algorithm commonly used to partition a dataset. *k*-M is an iterative algorithm, with two distinct steps. 1) Each instance is assigned to a cluster, by calculating the Euclidean Distance between that instance and each centroid. Then, the lowest distance indicates which cluster the instance is assigned to. 2) Each centroid is updated to be the mean of all the instances assigned to it. The algorithm stops when the centroids no longer change position. The classification of a new sample is done by computing the Euclidean Distance of the new sample with each centroid, discovering which is closer. The label of the cluster becomes the predicted label of the sample.

Logistic Regression (LR). A regression model where the dependent variable is categorical. This binary LR model is used to estimate the probability of a binary response based on one or more variables. The classification of samples is done using the following equation

$$f(x) = \beta_0 + \sum_{i=1}^{m} \beta_i x_i \quad (3)$$

where 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.

Cryptographic Domain

The final step of the implementation consisted of adjustments to the evaluation processes of the ML algorithms in order to be compatible with privacy-preserving techniques, GC and HE. These two techniques offer different means to obtain privacy-preserving computations, and considerations must be made when choosing which ML algorithm to use with each cryptographic algorithm. GC builds ciphered Boolean circuits, which means that most operations are possible to implement. however, arithmetic operations require a large number of logic gates, creating an overhead that makes GC

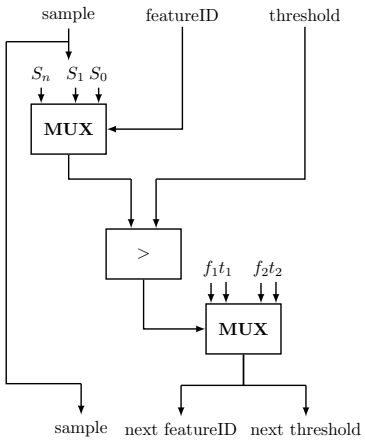


Figure 1: Boolean circuit of each node in a DT.

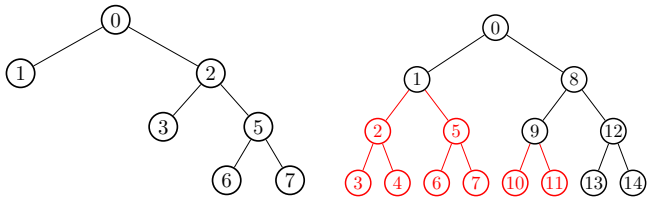
very slow for those operations. So, for some of the ML algorithms, we used a HE system, since it offers arithmetic operations as a core operations. The following sections describe the chosen combinations.

GC + DT. The process of evaluating a DT in a privacy-preserving context is similar to evaluating it in the usual manner, as described in 3. 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 caused by observation of the computations.

Figure 1 shows the computations done inside each node of the DT. Each node contains the ID of the feature to be selected from the sample to be classified, and the threshold that the value of the feature is compared against. The first MUX gate selects from the sample the feature to be compared. The GREATERTHAN gate compares the selected feature with the threshold. The second MUX chooses the next ID and threshold for the next node in the tree.

One other aspect to mention is that the trees are always complete, i.e., the number of nodes n is always the maximum possible, and can be defined as $n = 2^{h+1} - 1$, where h is the *height* of the tree. Figure ref shows the implications of this expansion.

GC + k-Means. The process of evaluating the k-Means algorithm in a privacy-preserving manner is similar to evaluating in the usual manner. The operations in the prediction step of the algorithm were transformed into Boolean Circuits, with logic gates representing operations. In Figure 3 we show the circuit we designed to represent the k-Means prediction.



(a) Original DT. (b) Complete DT.

Figure 2: Expansion of binary trees.

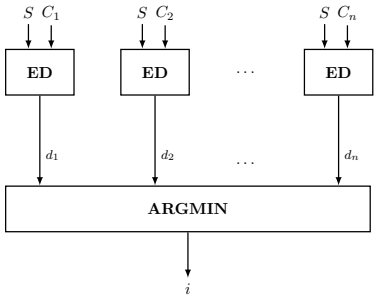


Figure 3: Boolean circuit of the k-Means prediction.

HE + LR. In order to use a FHE system, the prediction function for LR described in 3 must be converted to:

$$f_{FHE}(x) = D_k\left(E_k(\beta_0) + \sum_{i=0}^{m} E_k(\beta_i) \cdot E_k(x_i)\right) \tag{4}$$

where E_k represents the encryption operation and D_k represents the decryption operation using the key k.

Converting Equation 4 to be computed using a PHE system is straightforward, but this can only be done under 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 additive PHE system becomes:

$$f_{PHE}(x) = D_k\left(E_k(\beta_0) \cdot \prod_{i=1}^{m} E_k(x_i)^{\beta_i}\right) \tag{5}$$

HE + SVM. For the SVM algorithm, we only considered the linear kernel, as it simplifies the scoring function. The Equation 2 is then simplified to the following:

$$f(x) = \sum_{i=1}^{m} \alpha_i x_{SV}^{(i)} x + b = \sum_{i=1}^{m} \alpha_i \sum_{j=1}^{n} x_j x_{SV}^{(i,j)} + b \tag{6}$$

To compute this function using a FHE system, we must convert it to:

$$f_{FHE}(x) = D_k\left(\sum_{i=1}^{m} E_k(\alpha_i).\sum_{j=i}^{n} E_k(x_j).E_k(x_{SV}^{(i,j)})+E_k(b)\right) \quad (7)$$

where E_k represents the encryption operation and D_k represents the decryption operation using the key k.

Like before, converting it to be computed using a PHE system is equally straightforward, but this must be done under the same two assumptions: 1) the data to be evaluated (x) and the model parameters (α_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:

$$f_{FHE}(x) = D_k\left(\prod_{i=1}^{m}\left(\prod_{j=1}^{n} E_k(x_i)^{x_{SV}^{(i,j)}}\right)^{\alpha_i}.E_k(b)\right) \quad (8)$$

4 EXPERIMENTAL EVALUATION

This section presents the evaluation results. The objective of the experimental evaluation is to answer two important questions: 1) How accurate is the prediction versus the baseline system? (Section 4) 2) what is the overhead in using privacy-preserving techniques in terms of execution time and communication costs? (Sections 4 and 4)

All the experiments were performed using a machine with an Intel Core i5-4300M CPU @2.60Gz with 3MB L3 cache memory and 12 GB RAM memory. 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. The training step of the algorithms was done using publicly available ML toolkit for Python, *scikit-learn*[4]. The GC results were obtained using the toolkit developed by VIPP group from the University of Siena[5]. The results using FHE were obtained using the HElib toolkit [4].

Comparison with the Baseline

It is important to mention that, after analyzing the results obtained using the VIPP toolkit to implement GC, 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 2 and 3, for the experiments for DT and k-Means respectively. It is to be noted that this error is computed versus the baseline prediction results, not the prediction labels from the dataset.

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-Means only output an integer representing the label, and not a real number, the visible effect of changing the number of bits is minimal.

Table 2: GC+DT. Average label prediction error VS the baseline.

Bits	Pima	Breast Cancer	Credit	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 3: GC+k-Means. Average label prediction error VS the baseline.

Bits	Pima	Breast Cancer	Credit	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%

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 the 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.

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 separately. We do not present total execution times for the whole datasets because execution times per sample are independent of the dataset size, and execution times 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.

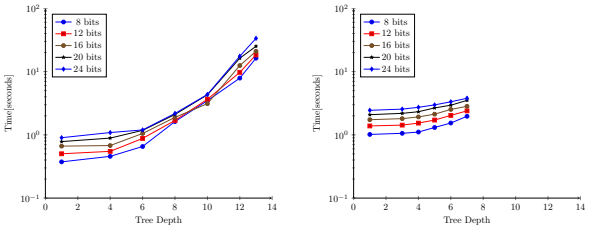
GC+DT. We present here the execution times obtained by using the VIPP toolkit to build a GC implementation of

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

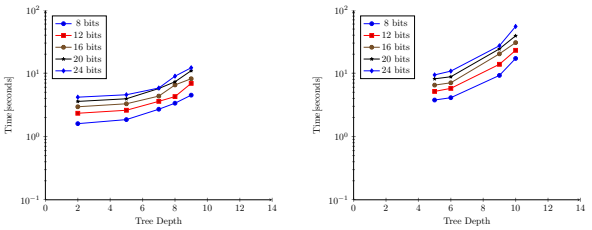
[5]http://clem.dii.unisi.it/ vipp/index.php/home

Table 4: GC+DT. Average pre-computation times/data sample, in seconds.

Bits	Pima	Breast Cancer	Credit	Adult Income
8	0.219	0.205	0.224	0.233
12	0.285	0.240	0.253	0.271
16	0.310	0.281	0.271	0.313
20	0.344	0.325	0.290	0.355
24	0.356	0.356	0.315	0.373



(a) Pima Indians Diabetes Dataset. (b) Breast Cancer Wisconsin Diagnostic Dataset.



(c) Credit Approval Dataset. (d) Adult Income Dataset.

Figure 4: GC+DT. Runtime per data sample, in seconds.

DT for all datasets. The results are presented in terms of average pre-computation times per data sample (Table 4) and Runtimes per data sample (Figure 4).

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 number of bits used. This means that pre-computation poses no restrictions regarding the scalability of our approach.

Regarding the runtimes per data sample, we can observe that the results obtained eclipse the pre-computation times. Although they 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.

Table 5: GC+*k*-Means. Average pre-computation times/data sample, in seconds.

Bits	Pima	Breast Cancer	Credit	Adult Income
8	0.260	0.225	0.226	0.214
12	0.283	0.251	0.249	0.214
16	0.307	0.274	0.253	0.232
20	0.331	0.289	0.265	0.266
24	0.339	0.301	0.272	0.262

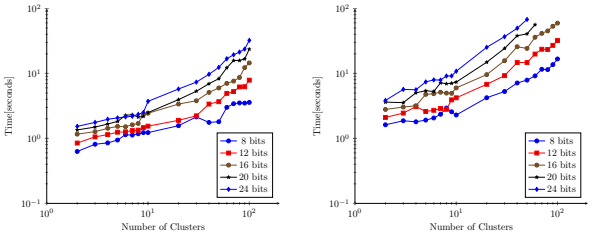
GC+k-Means. We present here the execution times obtained by using the VIPP toolkit to build a GC implementation of *k*-Means for all datasets. The results are presented in terms of average pre-computation times per data sample (Table 5) and Runtimes per data sample (Figure 5).

We can see in Table 5 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.

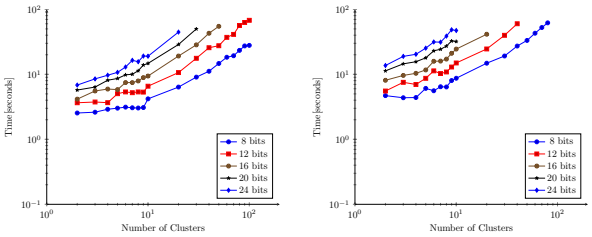
Regarding the runtimes per data sample, we observe again that the results are much larger that the pre-computation times. They scale linearly with the number of features and slightly super-linearly with the number of clusters, none of which compromises the scalability of our approach. However, runtimes 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 3 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 number 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 number of clusters considered in order to obtain much faster runtimes.

HE+LR. We present here the execution times obtained by using the PHE and FHE systems with LR. For the FHE system, we present the times for methods 1 (M1) and 2 (M2) together, for ease of comparison. The results are presented in terms of execution time per data sample.

When observing the execution times obtained using PHE (Figure 6), 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 increases. This can cause a problem of scalability, but it can

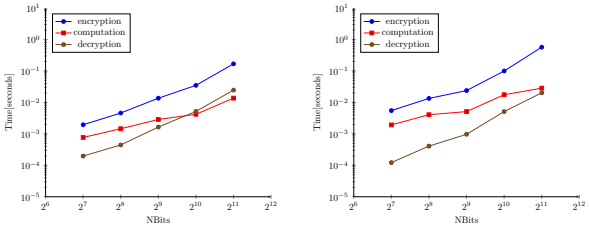


(a) Pima Indians Diabetes Dataset. (b) Breast Cancer Wisconsin Diagnostic Dataset.

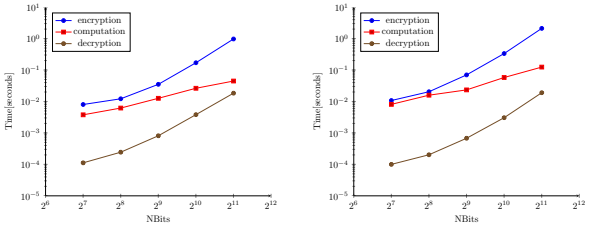


(c) Credit Approval Dataset. (d) Adult Income Dataset.

Figure 5: GC+*k*-Means. Runtime per data sample, in seconds.



(a) Pima Indians Diabetes Dataset. (b) Breast Cancer Wisconsin Diagnostic Dataset.



(c) Credit Approval Dataset. (d) Adult Income Dataset.

Figure 6: PHE+LR. Execution time per data sample, in seconds.

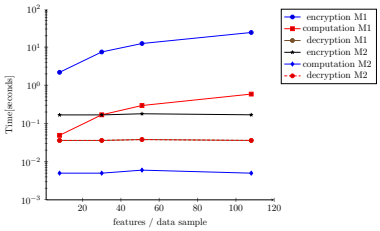


Figure 7: FHE+LR. Execution time per data sample, in seconds.

be safely ignored since the execution times per sample are still very small.

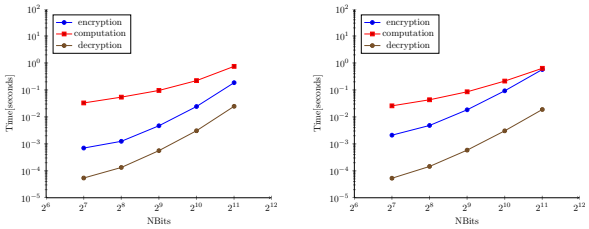
When analyzing the results obtained using FHE (Figure 7), we can observe that the packing used by method M2 greatly decreases the encryption and computation times, when compared to method M1. We also observe that method M2 makes the encryption and computation times independent from the number of features. Overall, we see that method M2 is much more efficient that method M1, showing the obvious advantage of packing 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 caused by feature packing, especially due to the gains in encryption time.

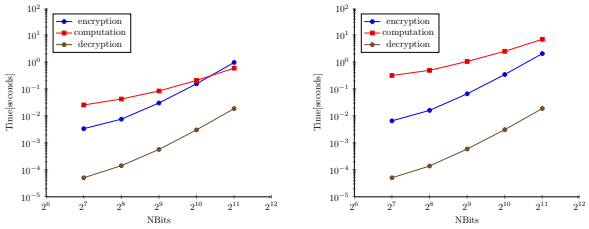
HE+SVM. We present here the execution times obtained by using the PHE and FHE systems with SVM. For the FHE system, we present the times for methods 1 (M1) and 2 (M2) in the same cell, for ease of comparison. The results are presented in terms of execution time per data sample for the PHE case, and execution time per sample and support vector for the FHE case.

When observing the execution times obtained using PHE (Figure 8), 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 NBits. Finally, we can see that decryption times are constant with increasing number of features and number of Support Vectors, but we verify that there 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 on one side of the protocol (due to the way the Paillier cruptosystem works). Also, we can observe that decryption times are comparable in both cases, as decryption only occurs once, when all operations have been performed.



(a) Pima Indians Diabetes Dataset. (b) Breast Cancer Wisconsin Diagnostic Dataset.



(c) Credit Approval Dataset. (d) Adult Income Dataset.

Figure 8: PHE+SVM. Execution time per data sample, in seconds.

When observing the execution times obtained using FHE (Figure 9), 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 of 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 many multiplications have to be made (one for each Support Vector) overwhelms FHE when compared with PHE for evaluating an SVM.

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

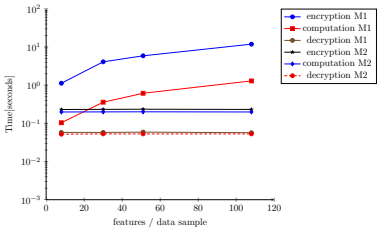
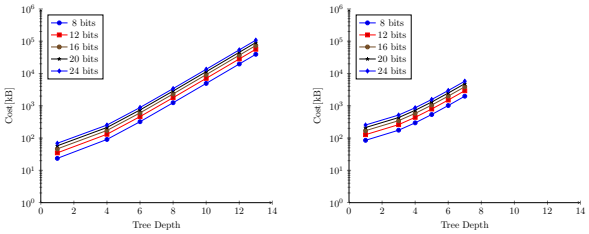
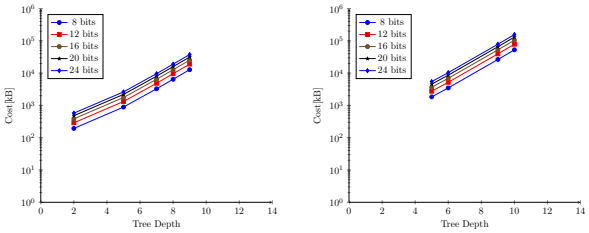


Figure 9: FHE+SVM. Execution time per data sample, in seconds.



(a) Pima Indians Diabetes Dataset. (b) Breast Cancer Wisconsin Diagnostic Dataset.

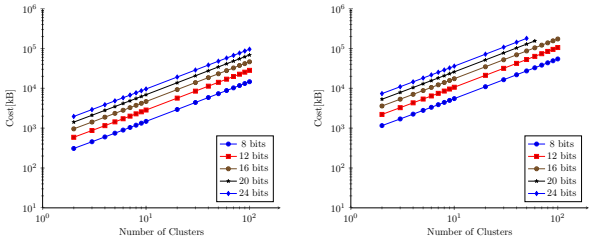


(c) Credit Approval Dataset. (d) Adult Income Dataset.

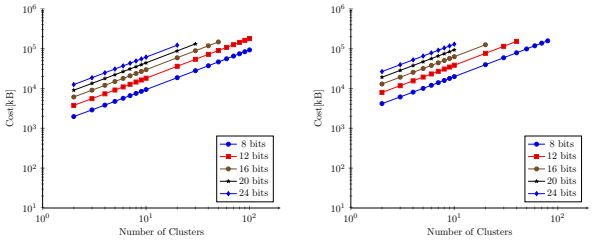
Figure 10: GC+DT. Amount of bytes per data sample (in kB) received during runtime by the GC evaluator.

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. We also do not present total communication costs for the whole datasets because communication costs per sample are independent of 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.

GC+DT. We present here the communication costs obtained by using a GC implementation of DT for all datasets. We present results in terms of the number of bytes per data sample received during runtime by the GC evaluator.



(a) Pima Indians Diabetes Dataset. (b) Breast Cancer Wisconsin Diagnostic Dataset.



(c) Credit Approval Dataset. (d) Adult Income Dataset.

Figure 11: GC+k-Means. Amount of bytes per data sample (in kB) received during runtime by the GC evaluator.

Regarding the amount of bytes per data sample received during runtime (Figure 10), they depend linearly on the numeric precision, on 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 considerably large, but it can be easily minimized by using the original DT instead of the fully expanded ones.

GC+k-Means. We present here the communication costs obtained by using a GC implementation of k-Means for all datasets. We present results in terms of the number of bytes per data sample received during runtime by the GC evaluator.

Regarding the amount of bytes per data sample received during runtime (Figure 11), 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 3 showed that the loss of accuracy is negligible even when only 12 bits are considered, meaning we can easily minimize its effects.

PHE. 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 $2 * \text{NBits}$ numbers and any ciphertext is a $2 * \text{NBits}$ 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 of 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:

Table 6: PHE. Communication costs per sample, in kilobytes (kB).

NBits	Pima	Breast Cancer	Credit	Adult Income
128	0.352	1.056	1.728	3.552
256	0.704	2.112	3.456	7.104
512	1.408	4.224	6.912	14.208
1024	2.816	8.448	13.824	28.416
2048	5.632	16.896	27.648	56.832

$$cost_{comm} = \underbrace{2(2\text{NBits})}_{\text{public key}} + \underbrace{Nn(2\text{NBits})}_{\text{ciphered data}} + \underbrace{2\text{NBits}}_{\text{ciphered result}} \quad (9)$$

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 6.

As expected, the communication costs increase linearly with increasing 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.

FHE. 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:$w_{key} \approx 3200\text{kB} = 3.2\text{MB}$) and each ciphertext is composed of approximately 100,000 64-bit values (total:$w_{ciphertext} \approx 800\text{kB} = 0.8\text{MB}$).

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 of the ML algorithm considered.

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

Table 7: FHE. Communication costs per sample, in Megabytes (MB).

Pima	Breast Cancer	Credit Approval	Adult Income
10.4	28.0	44.8	90.4

therefore given by:

$$cost_{comm} = \underbrace{w_{key}}_{publickey} + \underbrace{Nnw_{ciphertext}}_{ciphereddata} + \underbrace{w_{ciphertext}}_{cipheredresult} \quad (10)$$

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 7.

Once again, the communication costs increase linearly with increasing 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.

Final Remarks

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 adequate for ML algorithms that rely on arithmetic operations, and the GC approach is 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 GC+k-Means 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, 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.

Another important conclusion with 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 and 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 CONCLUSIONS

This paper presented a platform to perform privacy-preserving ML computations, with the goal of providing means to apply the privacy-preserving paradigm in Big Data operations. We discussed the existing techniques that provide a level of privacy compliance with the laws in force and matched those techniques with the most commonly used ML algorithms. We evaluated the solution by comparing two privacy-preserving techniques: Garbled Circuits and Homomorphic Encryption. We were able to observe the overhead of these techniques when compared to operations in the clear.

REFERENCES

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

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

[3] Giuseppe D'Acquisto, Josep Domingo-Ferrer, Panayiotis Kikiras, Vicenç Torra, Yves-Alexandre de Montjoye, and Athena Bourka. 2015. 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).

[4] Shai Halevi and Victor Shoup. 2014. HElib-An Implementation of homomorphic encryption. *Cryptology ePrint Archive, Report 2014/039* (2014).

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

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

[7] Lei Xu, Chunxiao Jiang, Jian Wang, Jian Yuan, Yong Ren, Lei Xu, Chunxiao Jiang, Jian Wang, Jian Yuan, and Yong Ren. 2014. Information Security in Big Data: Privacy and Data Mining. *IEEE Access* 2 (2014), 1149–1176. arXiv:arXiv:1011.1669v3

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

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

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

[11] Rüdiger Wirth and Jochen Hipp. 2000. 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*. Citeseer, 29–39.

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