Privacy-preserving Distributed Support Vector Machines

Simone Bottoni[1], Stefano Braghin[2],
Theodora Brisimi[2], and Alberto Trombetta[1]

[1] University of Insubria Varese, Italy
{sbottoni,alberto.trombetta}@uninsubria.it
[2] IBM Research Europe, Dublin, Ireland
{stefanob@ie.,theodora.brisimi@,}ibm.com

Abstract. Federated machine learning is a promising paradigm allowing organizations to collaborate toward the training of a joint model without the need to explicitly share sensitive or business-critical datasets. Previous works demonstrated that such paradigm is not sufficient to preserve confidentiality of the training data, even to honest participants. In this work, we extend a well-known framework for training sparse Support Vector Machines in a distributed setting, while preserving data confidentiality by means of a novel non-interactive secure multiparty computation engine, that preserves data confidentiality. We formally demonstrate the security properties of the engine and provide, by means of extensive empirical evaluation, the performance of the extended framework both in terms of accuracy and execution time.

Keywords: Distributed Support Vector Machines, Privacy-Preserving Machine Learning, Secure Federated Learning

1 Introduction

In an era of "big data", computationally efficient and privacy-aware solutions for large-scale machine learning problems become crucial. This becomes very relevant in the healthcare domain where large amounts of data are stored in different locations and owned by different entities. Past research has been focused on centralized algorithms, which assume the existence of a central data repository (database) that stores and processes the data from all participants. Such an architecture, however, can be impractical when data are not centrally located, it does not scale well to very large datasets, and introduces single-point of failure risks which could compromise the integrity and privacy of the data. Given the large amount of data that is widely spread across hospitals/individuals, a decentralized and computationally scalable methodology is very much in need.

Motivating scenario Hospitals want to create reliable and accurate models on the data that they own but for regulatory and compliant constraints they cannot

share data with other institutes. Within national boundaries, it might be possible to leverage governmental aggregators, but it is not always feasible to share data across borders, or even between organizations. Legislations like General Data Protection Regulation (GDPR)[3], California Consumer Privacy Act (CCPA)[4] and Safe Shield[5] provide definitions and levels of sensitivity in personal data and instruct all the entities involved in the data process pipeline how such categories of data should be managed. These data handling instructions further specify under which conditions data can be exchanged, with the general assumption that clear text data should not be openly shared with other entities. This constraint conflicts with the basic requirement of sharing data to train a joint model. Hence, a mechanism to securely compute models in a federated fashion is required.

Federated learning allows the hospitals to train a machine learning model without sharing their data. However, this mechanism is vulnerable to attacks, for example, in the inference problem [15] where an attacker can leak sensitive information about the participants' private data from the machine learning algorithm parameters [19].

To preserve privacy, federated learning can use various methods. Specifically, Secure Multi-party Computation (SMC), Differential Privacy, Homomorphic Encryption [25], or a combination of them, as in [23, 26]. Each of these methods has its advantages and disadvantages. SMC provides a complete Zero-Knowledge property, where each party involved knows nothing except its input and output, but this is difficult to achieve efficiently [8]. Differential Privacy prevents a third party from being able to associate data with an individual and protects their privacy, at the cost of losing a certain amount of the model's accuracy [17]. Finally, the Homomorphic Encryption guarantees privacy protection through parameter exchange under the encryption mechanism. The model's accuracy is preserved but this technique adds a non-negligible communication cost [27].

In this work we present a method based on the Homomorphic Encryption technique, and in particular, using the Pailler Homomorphic Encryption scheme [21], which is a method widely used in the previously described scenario. We choose to use a Homomorphic Encryption approach given the fact that it can preserve the privacy of data from any external adversary, it does not have any model precision loss and, finally, it is possible to apply this security method over an existing federated learning model. This is obtained without modification to the original algorithm, beside the encryption and decryption of the coefficients [27]. A similar approach to our work was considered in [12]. The authors propose a secure weighted aggregation scheme based on the Paillier Homomorphic Encryption where they try to protect the privacy of the clients' data from information leakage. The architecture of their protocol includes multiple clients that solve a machine learning problem with the coordination of a central unit that has the purpose to aggregate the values received from the clients. In their protocol, they evaluate the data disparity, i.e., the different amounts of data that

[3] https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=OJ:L:2016:119:TOC
[4] https://oag.ca.gov/privacy/ccpa
[5] https://eur-lex.europa.eu/eli/dec_impl/2016/1250/oj

the federated learning participants own. For the security of that protocol, the authors take into account both the privacy problems in data disparity evaluation and the privacy-preserving aggregation security issue. They also consider the possibility that both the central server and/or the clients send fraudulent messages to the others clients.

In [16] the authors present the xMK-CKKS, a multi-key homomorphic encryption protocol for a privacy-preserving federated learning method that is an improvement of the state-of-art MK-CKKS protocol [4]. They apply this scheme to FedAvg [18], a federated learning method based on iterative model averaging. The authors apply their solution to a smart healthcare scenario with 10 IoT devices and a central unit that computes the model aggregates. They compare their solution with a federated learning method based on the Paillier encryption scheme assuming that this last solution is not secure because all the devices share the same secret key and public key. They show that their solution loses less than 0.5% accuracy compared to the federated learning scheme and reduces the communication cost, the computational cost, and the energy consumption compared to the Pailler solution.

More recent work was presented in [10], where the author proposes a multi-party privacy-preserving federated machine learning framework, called PFMLP. This framework is based on partially homomorphic encryption, in particular, they used an improved Paillier algorithm, that adds a random integer in the encryption, which can speed up the training by 25–28%. They consider a Federated Network that includes a server that aggregates the encrypted model gradients and sends back the results to the clients. The security of their solution is based on the fact that the server does not have any key, so it can not see any plaintext avoiding the inference problem.

All previous works take into account a central unit with the purpose to aggregate the model coefficients shared from the clients. Our work does not involve the use of a single central unit, in fact, all the clients act both as a node and an aggregator. Our protocol also avoids the security problem related to the fact that all the devices share the same secret key and public key, generating different compatible keys in a particular way.

In this paper, we present Secure Homomorphic Sum protocol (SHSP), a novel SMC protocol that can be applied to any federated learning method. Encrypted messages are exchanged between agents containing the parameters of a model that they all jointly want to learn. To illustrate it in practice, we apply the proposed encryption scheme on top of a federated learning method, where agents jointly solve the sparse Support Vector Machine (sSVM) in a distributed privacy-preserving manner without exchanging their raw data. As a case study to illustrate our algorithm, we apply it in a healthcare scenario, where we study the heart disease problem.

The paper is organized as follows. Section 2 presents an overview of the distributed sSVM algorithm based on a framework called cluster Primal Dual Splitting (cPDS). In Section 3 we explain the protocol and its variation that adapts cPDS to be secure. In Section 4 we present the datasets, how we executed

the tests, and the setting that we used. In section 5 we show the results that we have obtained and we finally discuss them in Section 6. We conclude in Section 7 depicting some future work.

2 Federated Primal Dual Split Method

We will be extending the work presented in [2], that describes a federated learning of predictive models from federated data. Specifically they apply the cPDS framework to the sSVM.

Let us briefly summarize the work presented in [2]. Suppose a set $\mathcal{P}$ of agents that want to collaborate to train a sSVM global model to separate two classes. Each participant $p \in \mathcal{P}$ owns a subset $\mathcal{D}_p \subseteq \mathcal{D}$ of the entire dataset and maintains a copy $(\boldsymbol{\beta}_j, \beta_{j0})$ of the classifier parameters to be estimated. In each iteration of the method the parameters $(\boldsymbol{\beta}_j, \beta_{j0})$ are updated, using data locally stored in the agent and the coefficients that the agent receives from its neighbors. In every iteration each agent updates $\mathbf{x}_j = (\boldsymbol{\beta}_j, \beta_{j0}) \in \mathbb{R}^{d+1}$, $\mathbf{y}_j \in \mathbb{R}^{n_j}$, $\mathbf{q}_j \in \mathbb{R}^{n_j}$ and $\boldsymbol{\lambda}_j \in \mathbb{R}^{d+1}$.

We illustrate in Algorithm 1 the cPDS updates that each agent j is performing. The effectiveness of this approach has been demonstrated against state-of-the-art of local and distributed methods in [2].

3 Method

From a high-level point of view, the protocol presented in Section 2 can be viewed as a sequence of iterations in which, at every stage, a set of nodes sends its own locally computed weights to an aggregator that sums them, and afterward uses the aggregated value as input for the next stage. Note that this is an abstraction of the actual protocol. In cPDS each node is at the same time aggregator and worker, as each nodes receives weights from its neighbours and sends the weights it computed on its own data to the neighbours as well.

In what follows, we describe how an aggregator may sum the coefficients sent by the nodes in an oblivious way, obtaining the aggregated value of their sum without knowing their actual values. The proposed approach is based on a simple deployment of the Paillier encryption scheme, known to be additively homomorphic [21]. This, in turn, allows us to define a secure version of the protocol presented in Section 2 in which the weights collected by an aggregator are not directly accessible neither to the aggregator nor to the other nodes (except – of course – the weights a node has sent itself). In this way, it is not possible for a party acting in the protocol to infer the weights computed by other parties and, in turn, no party is able to perform an inference attack on the data that parties exchange during the protocol's stages. In this work, we do not address attacks different from inference attacks or – as an example – collusion attacks [24]. Other attack scenarios can be addressed by combining works previously presented, such as [17] and [15], to the framework presented in remainder this section. This is

Algorithm 1: cPDS method

Function `main()`:
  $x_j^0, y_j^0, q_j^{-1}, q_j^0, \lambda_j^{-1}, \lambda_j^0 = initialize()$
  for $k \leftarrow 0$ **to** max_iter **do**
    $x_j^{k+1} = compute\_local(\lambda_j^k)$
    $\lambda_j^{k+1} = \lambda\_update(x_j^{k+1})$ `// requires information exchange`
  end

Function `initialize()`:
  $x_j^0 \in \mathbb{R}^{d+1}$
  $y_j^0 \in \mathbb{R}^{n_j}$
  $q_j^{-1} = 0$
  $q_j^0 = \Gamma_j(A_j^\intercal x_j^0 - y_j^0)$
  $\lambda_j^{-1} = 0$
  $\lambda_j^0 = x_j^0 - \sum_{i \in N_j \cup \{j\}} w_{ji} x_j^0$
  return $x_j^0, y_j^0, q_j^{-1}, q_j^0, \lambda_j^{-1}, \lambda_j^0$

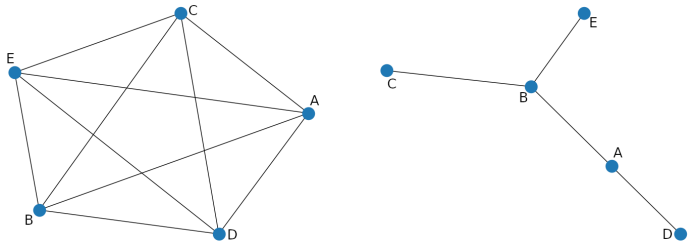
Function `compute_local`(λ_j^k):
  $x_j^{k+1} = \underset{x_j}{\operatorname{argmin}}\{\langle 2q_j^k - q_j^{k-1}, \Gamma_j A_j x_j\rangle + g_j(x_j) + \langle 2\lambda_j^k - \lambda_j^{k-1}, x_j\rangle + 0.5\|x_j - x_j^k\|^2_{\Theta_j}\}$
  $y_j^{k+1} = \underset{y_j}{\operatorname{argmin}}\{f_j(y_j) + \langle q_j^k, -\Gamma_j y_j\rangle + 0.5\|y_j - A_j x_j^{k+1}\|^2_{\Gamma_j^\intercal \Gamma_j}\}$
  $q_j^{k+1} = q_j^k + \Gamma_j(A_j x_j^{k+1} - y_j^{k+1})$
  return x_j^{k+1}

Function λ_update`()`:
  $\lambda_j^{k+1} = \lambda_j^k + x_j^{k+1} - \sum_{i \in N_j \cup \{j\}} w_{ji} x_j^{k+1}$
  return λ_j^{k+1}

possible because the framework here presented preserves confidentiality of the weights computed locally by each node, but does not make any assumption on how the weights have been computed.

We begin by presenting the Paillier-based secure sum protocol. Consider a set $\mathcal{P}$ of nodes, or *participants*, $\{p_1, \ldots, p_m\}$ and a dataset $\mathcal{D}$. Each participant $p \in \mathcal{P}$ owns a subset $\mathcal{D}_p \subseteq \mathcal{D}$ of the entire dataset.

According to the federated machine learning protocol described in [2], each participant of the federated machine learning task acts as an aggregator for the weights computed by its neighbors. Such neighbors are predefined according to the network topology connecting the various agents. Departing from traditional approaches, we remark that our protocol does not require multiple interaction steps among the nodes and an aggregator. There are two possible scenarios about network topology, as shown in Figure 1a and Figure 1b. The first scenario implies



(a) Case 1 (b) Case 2

Fig. 1: Simplified depiction of possible topological scenarios

that all nodes that participate in the computation have at least two neighbors. On the other side, in the second scenario is it possible to find some nodes that have only one neighbor. In this case, it is necessary to modify the protocol to compute the sum. In the following, we present the protocol, its variation, and how we provide security to the machine learning model.

3.1 Keys Generation and Encryption

We assume the presence of a trusted *keys generator* that computes the cryptographic material (keys and random values) and distributes it to the appropriate parties participating in the protocol execution. We can safely assume that after the key generation and distribution phase the key generator may go offline.

The keys generator computes, for each node, a *master private key* msk and a set of *secret encryption keys* $\{\mathsf{pk}_1, \ldots, \mathsf{pk}_m\}$, where m is the number of node's neighbors. The detailed procedure for the generation of the keys, based on the Paillier keys generation scheme, is summarized in Algorithm 2.

We note that the parameters for key generation are the same as in the Paillier scheme, and follow the same rules. Such parameters are used to generate a *master private key* msk that is used by a node to decrypt the aggregated value that it obtains after the sum of all the values that it has received from its neighbors (the associated public key is not used in our protocol, and so it is not generated). We detail how the nodes use these keys and which values (and how many times) are exchanged among the nodes in the next section.

Referring to the Algorithm 2 we note that the set of *secret encryption keys* $\{\mathsf{pk}_1, \ldots, \mathsf{pk}_m\}$ are generated following the Paillier public key rules, but in addition, a random value r_i is added, different for each key of the set. Such keys will be used by the node's neighbors to encrypt the values before exchanging them. The decryption of one encrypted value can only happen after having summed up all the other ones, using the associated *master private key* msk. If a node tries to decrypt an encrypted value before having summed it with the other values, it obtains an obfuscated value with the secret r_i. We detail the security of the message in Section 3.4.

Note that the association between node's neighbors and keys $\{\mathsf{pk}_1, \ldots, \mathsf{pk}_m\}$ could be further randomized to reduce collusion risks.

Algorithm 2: Secure Homomorphic Sum protocol

Function `parameters_generation()`:
  Choose p, q random primes such that $\gcd(pq, (p-1)(q-1)) = 1$
  Compute $n = pq$ and $\lambda = \text{lcm}(p-1, q-1)$
  Select random integer g where $g \in \mathbb{Z}_{n^2}^*$
  Check the existence of $\mu = (L(g^\lambda \bmod n^2))^{-1} \bmod n$, where $L(x) = \frac{x-1}{n}$
  return n, g, L, μ

Function `master_private_key(`L, μ`)`:
  $msk = (\lambda, \mu)$
  return msk

Function `secret_encryption_keys(`n, g`)`:
  Select $m-1$ random numbers, $\{r_1, \ldots, r_{m-1}\}$ from Z_n
  Set $r_m = -\sum r_i$
  Let $p_i = (n, g, r_i)$ be the secret key associated to agent i
  return $\mathcal{P} = \{p_1, \ldots, p_m\}$

Function `encryption(`p_i, m_i`)`:
  Check $m_i < n$
  Select $r < n$ such that $r \in \mathbb{Z}_n^*$
  Ensure that $\gcd(r, n) = 1$
  $c_i = g^{m_i} * g^{r_i} * r^n \bmod n^2$
  return c_i

Function `decryption(`msk, c`)`:
  $m = L(c^\lambda \bmod n^2)\mu \bmod n$
  return m

3.2 Protocol

The protocol assumes that a node p has $m \geq 2$ neighbors. All the nodes work both as a simple node and as an aggregator. In the following, we present how the protocol works taking into account only one node as an aggregator; these operations must be applied to all nodes in parallel, considering each node as an aggregator.

The trusted *keys generator* sends to the aggregator node a *master private key* and a secret encryption key p_m taken from the set of keys $\mathcal{P}$; then it sends to each neighbor of the aggregator a different key always taken from the set of *keys* $\mathcal{P}$. The secret encryption keys are distributed to the nodes so that all the nodes

that participate in the computation group have only one key present in the set and all the keys must be allocated.

At each iteration of the algorithm presented in Section 2, each participant i computes its algorithm parameters based on its data. Then everyone encrypts its coefficients with the secret encryption key p_i, and sends the encrypted value to the node designated as the aggregator. When the aggregator has received all the encrypted data from all its neighbors, it encrypts its algorithm's coefficients with its secret encryption key p_m and aggregates the values by summing all encrypted values that it has received, obtaining a value $E = \sum e_i$. Because of Paillier Encryption schema construction, this results in $E = g^{(v_1+r_1)+(v_2+r_2)+\ldots+(v_m+r_m)}$. Knowing that $\sum r_i = 0$ this is equivalent to $E = g^{v_1+v_2+\ldots+v_m}$, the aggregator can decrypt the aggregated value E with its *master private key* msk to obtain the plaintext.

Because of the presence of r_i in the keys, without first adding all the values, it is not possible to recover with enough confidence the values from the decryption of individual cyphertexts.

3.3 A variation of the protocol

The protocol presented in the previous section implies that a node p has $m \geq 2$ neighbors. In fact, if we were deploying the previously described protocol in a scenario like the one in Figure 1b, where there are some nodes as C, D and E with exactly one neighbor, when these will be aggregators they could leverage their secret encryption keys to extract the values of the coefficients of the nodes A and B. For example, if we consider C as the aggregator, it could encrypt its algorithm's coefficients with its secret encryption key, sum with the encrypted values that it has received from B and then it can decrypt with its *master private key*. This way, C obtains a plaintext that is an aggregated value composed by its value and the value of B. To obtain the B's plaintext the aggregator must only subtract its coefficients. Thus, it would be able to infer some information about the nature of the data that the neighbor node possesses.

This variation adapts the protocol to work correctly with the presence of nodes with only one neighbor in the graph's topology.

The solution that we propose to solve this problem uses dummy links. The keys generator, which knows the graph topology, when seeing a node that has only one neighbor, instead of generating only two secret encryption keys p1, p2, generates a third key p3. This last key is sent to a random node that is a neighbor of the only neighbor node that has the aggregator. In this way, the node that is the only neighbor of the aggregator sends to the aggregator two different values, its value and the other one that has received from one of its neighbors. Applying this solution, the protocol can be considered as the standard one defined in the previous paragraph. The security of this protocol's variation is the same that the standard protocol has, because, using the third values encrypted with the third key, the aggregator can't recover with enough confidence the right values and it can't infer any information about the data.

3.4 Security considerations

We now present the proof that our protocol is secure in an honest-but-curious scenario. In this scenario, we consider a passive adversary that listens to the network and it can obtain all the messages that the federated learning parties exchange during a computation. These messages, which are the model coefficients, were encrypted by a node with its secret encryption key. Following the protocol presented in this section, the only way that the adversary has to decrypt these ciphertexts is to get all the others messages from the others nodes of the group, sum them and decrypt the aggregated value with the aggregator's private key.

Now, we assume that our protocol is insecure. Then we can assume that there is an adversary that may decrypt each message it intercepts in the network (without knowing the secret key). So, the adversary can decrypt every message exchanged during the machine learning computation. We start from the fact that an encrypted message with our protocol is equal to $c = E(p_i, m) = E((n, g, r_i), m) = g^m * g^{r_i} * r^n \bmod n^2$, and, if we set $ri = 0$ we obtain $c = E(p_i, m) = E((n, g), m) = g^m * r^n \bmod n^2$ that is equal to a ciphertext encrypted with the Paillier standard protocol. Given this fact and taking into account an adversary with the previously described power, we can conclude that an attacker that can easily decrypt a message encrypted with our protocol can easily decrypt a message encrypted with the standard Pallier protocol. Thus, an adversary that uses the procedure for breaking our protocol as a block-box for breaking the Paillier scheme.

In conclusion, if we consider [3], where the authors prove that Paillier's encryption scheme is semantically secure, we can assume that our protocol is also semantically secure.

4 Experimental evaluation

In this section, we describe how we modified the algorithm presented in Section 2 to make it more secure, i.e., to avoid that a node can infer sensible information from data shared by other participants in the computation. We also present how we evaluated the impact of our protocol on the algorithm, considering both the time used to train the model and the performance of classification obtained in tests.

To develop the secure version of cPDS we use the *phe* Python3 library [5] that implements the Paillier partially homomorphic encryption. This library is composed of some functions that allow us to generate a different set of keys, encrypt some values, and decrypt the ciphertexts. It also contains a function to sum two ciphertexts and one to multiply a ciphertext with a scalar. We modified this library to adapt it to our requirements; in particular, we added a new function that allows us to generate a new set of keys following the protocol that we previously described. We also adapted the encryption and decryption functions to work correctly with the new version of the keys. Algorithm 2 illustrates the details about these functions.

Finally, we apply to the cPDS algorithm, the adapted functions of the *phe* library, to encrypt the computed coefficients, sum them and decrypt the aggregated values that the nodes composing the system exchange during algorithm execution.

4.1 Settings

To test the cPDS algorithm in a distributed way we considered different environment setups. In particular, we took into account the number of nodes that compose the system and the topology of the network that includes all these nodes.

To test the algorithm in a realistic scenario we vary the number of nodes that want to participate in the computation. In the healthcare scenario, this number represents the number of hospitals that own part of the data used to train the model. To emulate a real environment we range this number from 5 to 30. We observed these different numbers of clients in some business settings where solutions like [14].

In a real scenario, it is not sure that all nodes are communicating with each other. To take into account all the different possibilities we choose to test the secure version of the cPDS algorithm considering different network topologies. We run different tests, and before each test, we generate a graph that represents the topology of the network using the *Erdős - Rényi* model. To generate the graph we use three different levels of connection between nodes, in this way we have (i) a low connected graph where the edges of the nodes are as lower as possible, (ii) a partially connected graph where the nodes are on average connected and (iii) a fully connected graph where all the nodes are connected. To define more formally these three types we take into account the degree property of a network [13], that is the number of edges that a node has. This way, define n as the number of nodes present in the network, the expected number of a node's edges is equal to $n * p$ [9, 20]. Given this property, we define (i) a low connected graph as a network where nodes have an average degree equal to $n * 0.2$, (ii) a partial connected graph as a network with an average degree of $n * 0.5$ and finally (iii) a fully connected graph where the average degree is $n * 1$.

When varying the topology of the network, and/or the number of nodes, the number of the edges that connect the nodes between them also varies, and, as a consequence, the number of messages circulating on the network increases or decreases. This allows us to study - if and how much - the different number of messages and the different number of nodes impact the times and the accuracy of the model.

4.2 Datasource

To evaluate our protocol we train the cPDS algorithm considering three different datasets. The first dataset is a synthetic dataset generated in three different versions that vary in the number of features: the first one has $15,000$ instances and 5 features, the second one has $15,000$ instances and 10 features, and the

Fig. 2: Synthetic test dataset's classes.

third one has 15,000 instances and 20 features. All of these synthetic datasets were generated randomly following a Multivariate Gaussian Distribution [11]. We choose to vary these datasets based on the number of features because, in this way, we can study how the time to encrypt the coefficients of the cPDS algorithm increases as the number of features increases. We show an example of this synthetic dataset in Figure 2.

To evaluate our protocol in a real-world scenario with a dataset based on real data, we use the "*Heart Disease*" dataset [7]. This set is composed of different subsets of data, where each of these sets describes a heart disease study on a different location. In particular, we choose to use the dataset describing the Cleveland population, which is the only one that has been used by machine learning researchers and practitioners. This because it is the only dataset that has a very small percentage of *NaN* values and it is the one that has the highest number of instances. This dataset describes different health situations of patients, and the purpose is to find the presence, or not, of heart disease in a patient. The database has 76 raw features, but we used the *pre-processed* version, which is the version used by all the published experiments that use this dataset where the unimportant attributes are dropped and only the important ones are considered [22], that are only 14; the set has 303 instances. Finally, given the low number of data of the previous set, we choose to test the cPDS algorithm with another medical database, based on real data, that has more instances. The dataset that we choose to use is a subset of the Framingham database that is available on Kaggle [1, 6]. This set of data includes 15 features and more than 4,000 instances. This set is a study of cardiovascular disease conducted in the town of Framingham, Massachusetts, on over 5000 citizens. The data, as the previous one, describes different health situations of patients, and the purpose is to find the presence, or not, of cardiovascular disease in a patient.

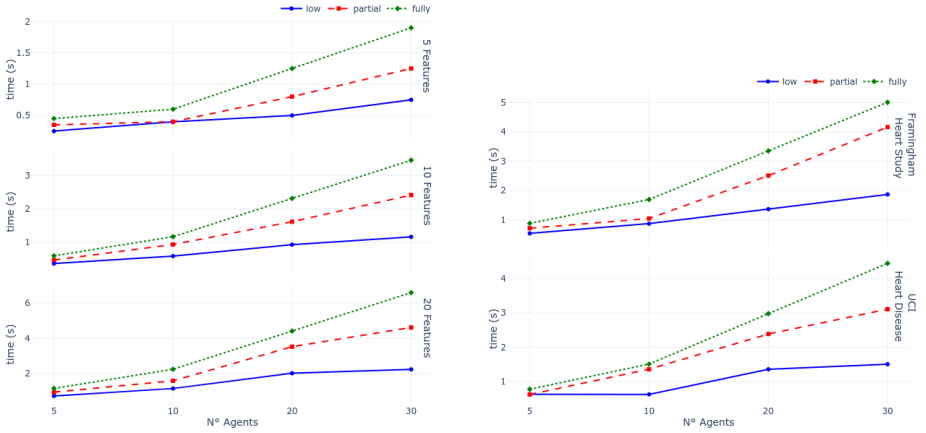
4.3 Metrics

To evaluate the performance of the secure version of cPDS we took into account the additional time compared to the classic version of the algorithm. To estimate the impact of the adapted *phe* secure functions on the execution time of the algorithm we recorded three different times during the execution of the algorithm. These recorded values are the times that were added to execute (i) the operation of encryption of the algorithm coefficients by the nodes, (ii) the operation of sum computed by the aggregators, and (iii) the operation of decryption of the aggregated values. For the encryption's operation, the time recorded was calculated by summing, for each node, the time used to encrypt the algorithm coefficients for every neighbor of the node. From these values, for each iteration, only the time of the node that has the greatest value was recorded, because, given the fact that the execution is in parallel the aggregator can't start the sum's operation without the values of all the nodes. The sum time is the max time that a node used to sum all the values that it receives from all its neighbors. The aggregator starts the operation only when it has received all the values from all its neighbors. Taking into account the fact that each node is both a simple node and an aggregator, when a node finishes the sum's operation it can start to decrypt the aggregated values. The decryption time was taken from each node and is the time that a node used to decrypt its aggregated coefficients. Then a new iteration can start immediately with the new values used to compute a new set of coefficients. Finally, to evaluate the performance of classification, we took into account, the Receiver Operating Characteristic Curve (AUC), obtained in the test of the model. We investigated - if and how much - our secure version of the algorithm impacted the precision of the model compared with the one that is not secure.

5 Results

In this section, we present the conducted tests and the results that we obtained, for both the added time to the computation and the performance of the trained models. We executed different tests to evaluate the secure version of the cPDS algorithm. These tests vary based on the dataset considered, the number of nodes, and the topology of the network that connected the nodes.

For each dataset, we run three different tests, one for each graph topology, and for each of these, we run four other tests, with a different number of nodes. We executed these tests one time for the secure version of cPDS algorithm and one time for the standard version. The results for the synthetic datasets are presented in Figure 3a. These three graphs differ based on the number of features that are present in the dataset. Each of these graphs represents the total time per iteration that we added to the computation of the secure version of cPDS compared to the non-secure version, as a function of the number of nodes that participate in the machine learning computation. The total time is the sum of the time used, by each node, to execute the three secure functions, the encryption, the sum, and the decryption ones, presented in section 4.3. We consider the



(a) Synthetic datasets times.

(b) *UCI - Heart Disease* and *Framingham Heart Study datasets times.*

Fig. 3: Max time per iteration.

maximum time of each of these functions because, taking into account the fact that they are executed in parallel, a node must wait for the values from other nodes before start a new iteration. In each of these graphs, we present the total time based also on the topology of the network taking into account the number of edges that connect the nodes. In Figure 3b we show the results obtained from the executed tests for the other two datasets, the "*UCI - Heart Disease*" and the "*Framingham Heart Study*". These two datasets are based on real data and regards the study of heart diseases. They have 13 and 15 features respectively. As in the previous case, we present the total time as a function of the number of nodes. We consider also the topology of the network.

We present in Figure 4 the AUC that we obtained from the tests that we executed with "*UCI - Heart Disease*" and the "*Framingham Heart Study*" datasets. In this graph, we inserted only one value per test, given the fact that the values obtained from the executed tests with the secure version of cPDS are exactly the same as the values obtained from the non-secure version. To be able to compare the AUC obtained in tests between the two cPDS versions, we used the same parameters and the same network configuration between the nodes. In this way, we could see - if and how much - the added functions to make the algorithm secure impacted the performance of the trained model. In this graph is possible to see the AUC of the model in the setting where the protocol variation is applicable. These cases are the ones where we have 5 and 10 nodes with a lower connected graph, and so we have the highest probability to find a node with only one neighbor. To have a better understanding of this fact, we also show in this graph the AUC obtained from tests executed removing the protocol variation. These tests are executed with the same setting, but given the fact that some

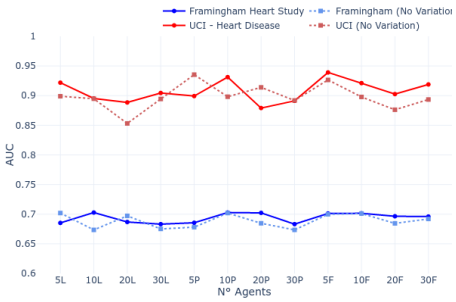


Fig. 4: AUC of tested datasets.

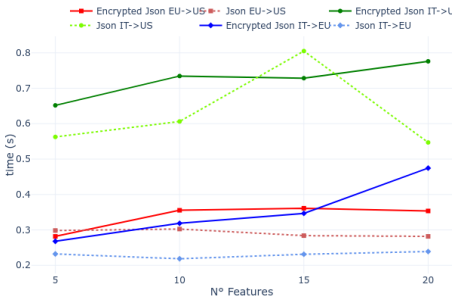


Fig. 5: Communication Cost.

Table 1: JSON size in KB.

# Features	Encrypted JSON(KB)	JSON(KB)
5	7768	122
10	14243	227
15	20715	332
20	27194	435

cPDS parameters are randomly initialized, the obtained AUCs are not perfectly the same.

Finally, we tested the communication costs, taking into account the time that two nodes use to exchange a message and the size of this message. We consider the different number of features, given the fact that a message corresponds to a serialized version of the coefficients computed by a node, and these coefficients are equal in number to the feature of the data, as required by cPDS. We present the communication time based on different server locations in Figure 5 and based on the message sizes in Table 1. To test the time we have created two different servers on the *Heroku* platform, the first one is located in *EU* region, and the second one in the *US* region and we exchange messages with our server located in *Italy*. This time includes the time to serialize a message in JSON format, send it from a server to another, and deserialize the JSON object. We recorded the times with both messages that include encrypted values and messages with clear values.

6 Discussion

Below, we discuss the results presented in the previous section. We also highlight the limits that our solution presents.

In Figure 3a we present the total time that we obtain from the tests with the synthetic datasets. From these three graphs, we show how the time that we add to the execution, to make it secure, depends on the number of features.

This is given by the fact that the coefficient vector of cPDS is as long as the number of features that are present in the dataset. This implies that each node has to encrypt more values, so the time increase. In fact, from this Figure, we can notice that the time increases linearly based on the number of features. We observe the same dependence considering not only the number of features but also the number of nodes and how much these nodes are connected between them. As we show in Figure 3a we note that, taking into account the three graph topologies independently, how the time slightly increases as the number of nodes increases. This is given by the fact that the more nodes we have, and the more they are connected, the more encrypted values each node has to sum. If we also consider the number of edges that connect the nodes we notice how the time increases as the connections between the nodes increase. This is motivated by the fact that each node must encrypt its coefficient several times, with different keys, that strictly depends on the number of nodes, as presented in section 3.2. We also note that the number of instances of a dataset does not impact the added time to the secure cPDS. We observe the same dependencies in Figure 3b where we show the results obtained from the tests with "*UCI - Heart Disease*" and the "*Framingham Heart Study*" datasets.

In Figure 6 we split the total time that we add to an iteration of the tests with the "*Framingham Heart Study*" dataset in its three components, the encryption, the sum, and the decryption time. We present, how the encryption time, that is the time that a node used to encrypt all the values in a single iteration, is around $96 - 98\%$ of the total time that we added to make the cPDS secure. We observe the same relation in all the others sets. From the above analysis, we can conclude that our solution performs well in presence of datasets with a restricted number of features, regardless of the number of instances. We also notice that our solution could be considered too expensive in the presence of a large number of participants in the computation that are all connected because this could fall into an expensive time-consuming.

Regarding Figure 4, one can observe that, for the two datasets considered, the obtained AUC is the same for both the secure version of cPDS and the non-secure version. This implies that our solution has no impact on the classification performance of the machine learning model. Moreover, we observe that the error introduced by the conversion from floating-point coefficients to integer during the encryption process is negligible in all the configurations, as shown in Figure 7. The AUC is also constant for all the settings that we have presented in Section 4.1. We also note how the obtained values do not vary too much in the settings where the protocol variation is applicable compared to the others. We conclude that the protocol variation does not impact the performance of the model, where applicable.

Finally, in Figure 5, where we show the communication cost, we observe that the time that two nodes used to serialize, send and deserialize an encrypted JSON is slightly higher than the not encrypted version, for all the servers considered. This is given by the fact that in a JSON file of the encrypted version are stored the encrypted coefficients, where each coefficient has a length of 2048 bits. In

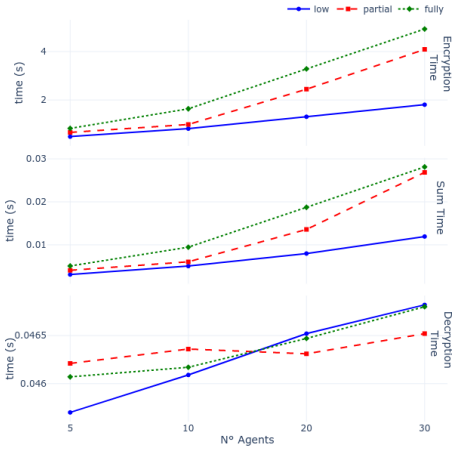


Fig. 6: Framingham Components.

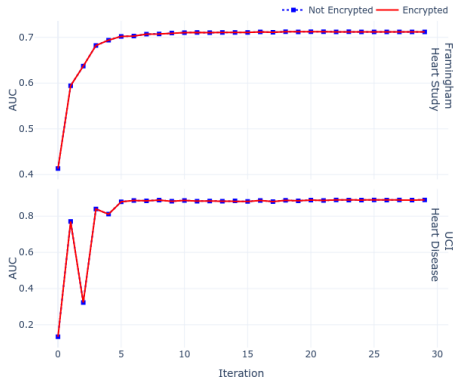


Fig. 7: Training AUC in a setting with 10 nodes partially connected.

the not encrypted version, these coefficients have a length equal to a double-precision floating-point length, which is 64 bits in our Python implementation. This has an impact also in the size of the JSON as showed in Table 1.

7 Conclusions and Future Work

In this work, we presented an extension of the cPDS algorithm based on the Paillier Homomorphic encryption scheme. Our solution allows hospitals to compute a machine learning model in a distributed way without sharing their private data, preserving their patients' privacy from inference attacks. Our work added a secure layer to protect the cPDS model's coefficient from Honest-but-Curious adversaries and we proved that SHSP is semantically secure assuming that Paillier's encryption scheme is semantically secure.

We evaluated our solution with different datasets, with both synthetic data and medical data, that vary according to the number of features and the number of instances. We showed how the time that we added to preserve the privacy of the data depends on the number of features that are present in the data and it does not on the number of instances. We considered also the topology of the network that connects the participants to the computation and we showed that this time depends also on the number of clients that participate in the computation and the number of edges that connect these nodes. We also demonstrated that our solution preserves the accuracy of the model, compared to the non-secure version, without any loss. However, we noted how our solution does not work well in all the settings that we considered. In fact, we noticed how the time that we added could be considered too high in the presence of a higher number of features and also in the settings where are present a lot of nodes with a high degree value.

Further analysis can be conducted to improve the time of our solution, for example reducing the number of encryptions that a node must do before sharing its coefficients with the other nodes. This would greatly reduce the time each node uses to encrypt the values, making our solution much more efficient. A potential future research direction could also be to investigate how we can protect the cPDS from other types of attacks, like membership inference attack.

Acknowledgement. Stefano Braghin and Theodora Brisimi are partially funded from the European Union's Horizon 2020 research and innovation programme under grant agreement No. 824988. `https://musketeer.eu/`

References

1. Ajmera, A.: Framingham heart study dataset (2018), data retrieved from Kaggle, `https://www.kaggle.com/amanajmera1/framingham-heart-study-dataset`
2. Brisimi, T.S., Chen, R., Mela, T., Olshevsky, A., Paschalidis, I.C., Shi, W.: Federated learning of predictive models from federated electronic health records. I. J. Medical Informatics **112**, 59–67 (2018)
3. Catalano, D., Gennaro, R., Howgrave-Graham, N.: The bit security of paillier's encryption scheme and its applications. In: Pfitzmann, B. (ed.) Advances in Cryptology - EUROCRYPT 2001, International Conference on the Theory and Application of Cryptographic Techniques, Innsbruck, Austria, May 6-10, 2001, Proceeding. Lecture Notes in Computer Science, vol. 2045, pp. 229–243. Springer (2001)
4. Chen, H., Dai, W., Kim, M., Song, Y.: Efficient multi-key homomorphic encryption with packed ciphertexts with application to oblivious neural network inference. Cryptology ePrint Archive, Report 2019/524 (2019), `https://eprint.iacr.org/2019/524`
5. Data61, C.: Python paillier library. `https://github.com/data61/python-paillier` (2013)
6. Dawber, T.R., Meadors, G.F., Moore, F.E.: Epidemiological approaches to heart disease: The framingham study. American Journal of Public Health and the Nations Health **41**(3), 279–286 (1951)
7. Detrano, R., Janosi, A., Steinbrunn, W., Pfisterer, M., Schmid, J.J., Sandhu, S., Guppy, K.H., Lee, S., Froelicher, V.: International application of a new probability algorithm for the diagnosis of coronary artery disease. The American Journal of Cardiology **64**(5), 304–310 (1989)
8. Du, W., Han, Y.S., Chen, S.: Privacy-preserving multivariate statistical analysis: Linear regression and classification. In: Berry, M.W., Dayal, U., Kamath, C., Skillicorn, D.B. (eds.) Proceedings of the Fourth SIAM International Conference on Data Mining, Lake Buena Vista, Florida, USA, April 22-24, 2004. pp. 222–233. SIAM (2004)
9. Erdös, P., Rényi, A.: On Random Graphs I. Publicationes Mathematicae Debrecen **6**, 290 (1959)
10. Fang, H., Qian, Q.: Privacy preserving machine learning with homomorphic encryption and federated learning. Future Internet **13**(4) (2021)
11. Flury, B.: The Multivariate Normal Distribution, pp. 171–207. Springer New York (1997)
12. Guo, J., Liu, Z., Lam, K., Zhao, J., Chen, Y., Xing, C.: Secure weighted aggregation in federated learning. CoRR **abs/2010.08730** (2020)

13. Kantarci, B., Labatut, V.: Classification of complex networks based on topological properties. In: 2013 International Conference on Cloud and Green Computing, Karlsruhe, Germany, September 30 - October 2, 2013. pp. 297–304. IEEE Computer Society (2013)
14. Ludwig, H., Baracaldo, N., Thomas, G., Zhou, Y., Anwar, A., Rajamoni, S., Ong, Y., Radhakrishnan, J., Verma, A., Sinn, M., Purcell, M., Rawat, A., Minh, T., Holohan, N., Chakraborty, S., Whitherspoon, S., Steuer, D., Wynter, L., Hassan, H., Laguna, S., Yurochkin, M., Agarwal, M., Chuba, E., Abay, A.: IBM Federated Learning: an Enterprise Framework White Paper V0.1 (2020)
15. Lyu, L., Yu, H., Ma, X., Sun, L., Zhao, J., Yang, Q., Yu, P.S.: Privacy and robustness in federated learning: Attacks and defenses. CoRR **abs/2012.06337** (2020)
16. Ma, J., Naas, S.A., Sigg, S., Lyu, X.: Privacy-preserving federated learning based on multi-key homomorphic encryption (2021)
17. Mammen, P.M.: Federated learning: Opportunities and challenges. CoRR **abs/2101.05428** (2021)
18. McMahan, H.B., Moore, E., Ramage, D., Hampson, S., y Arcas, B.A.: Communication-efficient learning of deep networks from decentralized data (2017)
19. Melis, L., Song, C., Cristofaro, E.D., Shmatikov, V.: Exploiting unintended feature leakage in collaborative learning. In: 2019 IEEE Symposium on Security and Privacy, SP 2019, San Francisco, CA, USA, May 19-23, 2019. pp. 691–706. IEEE (2019)
20. Newman, M.E.J.: Random graphs as models of networks, chap. 2, pp. 35–68. John Wiley & Sons, Ltd (2002)
21. Paillier, P.: Public-key cryptosystems based on composite degree residuosity classes. In: Stern, J. (ed.) Advances in Cryptology - EUROCRYPT '99, International Conference on the Theory and Application of Cryptographic Techniques, Prague, Czech Republic, May 2-6, 1999, Proceeding. Lecture Notes in Computer Science, vol. 1592, pp. 223–238. Springer (1999)
22. Patel, J., Tejalupadhyay, S., Patel, S.: Heart disease prediction using machine learning and data mining technique (03 2016)
23. Truex, S., Baracaldo, N., Anwar, A., Steinke, T., Ludwig, H., Zhang, R., Zhou, Y.: A hybrid approach to privacy-preserving federated learning. In: Cavallaro, L., Kinder, J., Afroz, S., Biggio, B., Carlini, N., Elovici, Y., Shabtai, A. (eds.) Proceedings of the 12th ACM Workshop on Artificial Intelligence and Security, AISec@CCS 2019, London, UK, November 15, 2019. pp. 1–11. ACM (2019)
24. Xu, R., Baracaldo, N., Zhou, Y., Anwar, A., Ludwig, H.: Hybridalpha: An efficient approach for privacy-preserving federated learning. In: Cavallaro, L., Kinder, J., Afroz, S., Biggio, B., Carlini, N., Elovici, Y., Shabtai, A. (eds.) Proceedings of the 12th ACM Workshop on Artificial Intelligence and Security, AISec@CCS 2019, London, UK, November 15, 2019. pp. 13–23. ACM (2019)
25. Yang, Q., Liu, Y., Chen, T., Tong, Y.: Federated machine learning: Concept and applications. ACM Trans. Intell. Syst. Technol. **10**(2), 12:1–12:19 (2019)
26. Yin, L., Feng, J., Xun, H., Sun, Z., Cheng, X.: A privacy-preserving federated learning for multiparty data sharing in social iots. IEEE Transactions on Network Science and Engineering pp. 1–1 (2021)
27. Zhang, C., Li, S., Xia, J., Wang, W., Yan, F., Liu, Y.: Batchcrypt: Efficient homomorphic encryption for cross-silo federated learning. In: Gavrilovska, A., Zadok, E. (eds.) 2020 USENIX Annual Technical Conference, USENIX ATC 2020, July 15-17, 2020. pp. 493–506. USENIX Association (2020)