Pattern Recognition 72 (2017) 93–107

Contents lists available at ScienceDirect

Pattern Recognition

journal homepage: www.elsevier.com/locate/patcog

# Training Support Vector Machines with privacy-protected data

Francisco-Javier González-Serrano*, Ángel Navia-Vázquez, Adrián Amor-Martín

*Department of Signal Theory and Communications, Universidad Carlos III de Madrid, Avda. de la Universidad, 30, 28911, Leganés, Spain*

ARTICLE INFO

*Article history:*
Received 31 March 2017
Revised 12 May 2017
Accepted 7 June 2017
Available online 28 June 2017

*Keywords:*
Machine learning
Privacy protection
Homomorphic encryption
Support Vector Machines

ABSTRACT

In this paper, we address a machine learning task using encrypted training data. Our basic scenario has three parties: Data Owners, who own private data; an Application, which wants to train and use an arbitrary machine learning model on the Users' data; and an Authorization Server, which provides Data Owners with public and secret keys of a partial homomorphic cryptosystem (that protects the privacy of their data), authorizes the Application to get access to the encrypted data, and assists it in those computations not supported by the partial homomorphism. As machine learning model, we have selected the Support Vector Machine (SVM) due to its excellent performance in supervised classification tasks. We evaluate two well known SVM algorithms, and we also propose a new semiparametric SVM scheme better suited for the privacy-protected scenario. At the end of the paper, a performance analysis regarding the accuracy and the complexity of the developed algorithms and protocols is presented.

## 1. Introduction

Recently, an increasing interest is detected in the machine learning (ML) research community regarding the design of algorithms and applications that take into account both the privacy of the signals/data to be processed and the trustworthiness of the parties involved in the transaction (see [1] and the references therein). One of the catalysts for that interest is the availability of ciphers with a homomorphic property. Privacy homomorphisms [2] are encryption functions that allow a set of operations on encrypted data to be performed by second parties (Service Providers or Applications) without any knowledge of the decryption key.

With Fully Homomorphic Encryption (FHE), it would be possible to evaluate an arbitrary number of additions and multiplications on ciphertexts and, thus, compute any function on encrypted data. However, current implementations of FHEs [3] are not fully usable yet due to the significant size of the encryptions and the need of costly bootstrapping operations on the ciphers to achieve the full homomorphism [4].

In contrast to FHE, partially homomorphic encryption systems are much more practical but at the expense of only supporting either multiplying or adding (but not both operations at the same time) encrypted data.

Unfortunately, the implementation of general ML algorithms in the Encrypted Domain will generally require the implementation of functions/operations other than the supported by the partial homomorphism. In this case, one option could be to implement a secure protocol to get assistance from the Data Owner (who also owns the secret decryption key) in those operations not supported by the partial homomorphism. In order to reduce (or eliminate) the direct involvement of Data Owners in all the intermediate operations, a third-party can provide privacy services to the Data Owners, and the required computing resources to the Applications [5].

### 1.1. Our contribution

In this work, we address the implementation of machine learning algorithms using data encrypted with multiple private keys. Taking as a starting point the scheme presented by Peter et al. [6], we propose a machine learning scenario where multiple mutually distrustful *Data Owners* (or Users) provide encrypted (with multiple keys) real-valued training examples to the *Application*, which processes them, with the assistance and previously granted access, of an *Authorization Server*, in order to learn the *same* parameters that would have been obtained if the training data had been in cleartext[1] (see Fig. 1). The computations not supported by the partially homomorphic encryption are outsourced to the Authorization Server, reducing the Data Owner involvement in the machine learning process. In order to increase the efficiency of the dedicated hardware (at the Authorizarion Server facilities), and protect

* Corresponding author.
*E-mail addresses:* fran@tsc.uc3m.es (F.-J. González-Serrano), navia@tsc.uc3m.es (Á. Navia-Vázquez), aamor@tsc.uc3m.es (A. Amor-Martín).

[1] In this context, "cleartext" means "not encrypted".

http://dx.doi.org/10.1016/j.patcog.2017.06.016
0031-3203/© 2017 Elsevier Ltd. All rights reserved.

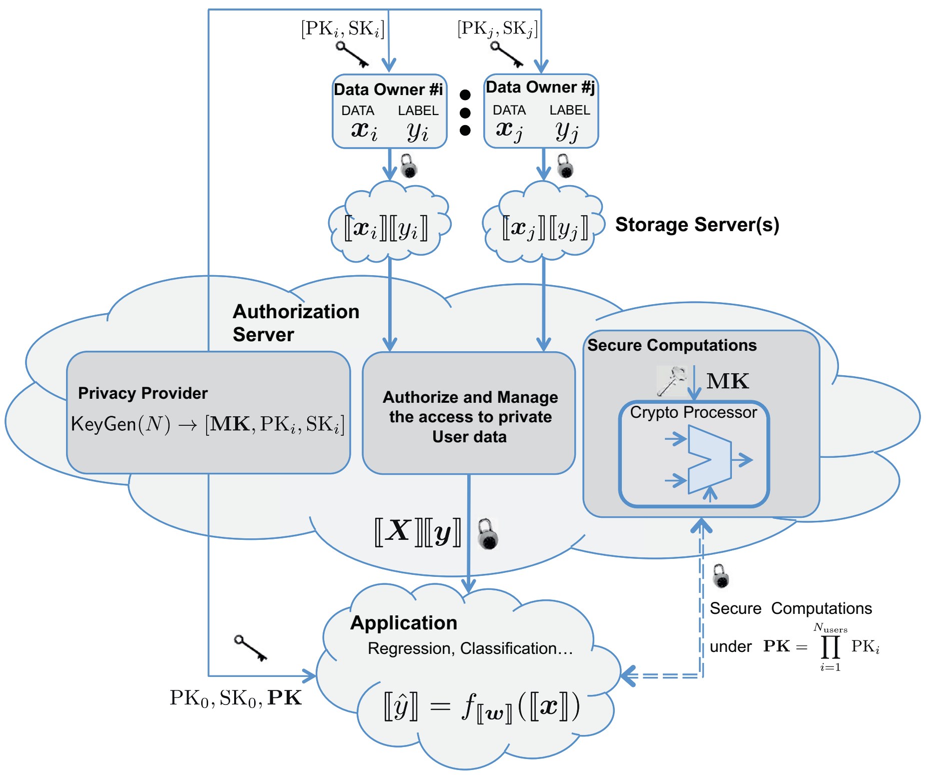
$[\mathrm{PK}_i, \mathrm{SK}_i]$
$[\mathrm{PK}_j, \mathrm{SK}_j]$
Data Owner #i
DATA LABEL
$\boldsymbol{x}_i$ $y_i$
Data Owner #j
DATA LABEL
$\boldsymbol{x}_j$ $y_j$
$[\![\boldsymbol{x}_i]\!][\![y_i]\!]$
$[\![\boldsymbol{x}_j]\!][\![y_j]\!]$
Storage Server(s)
Authorization Server
Privacy Provider
$\mathsf{KeyGen}(N) \rightarrow [\mathbf{MK}, \mathrm{PK}_i, \mathrm{SK}_i]$
Authorize and Manage the access to private User data
Secure Computations
$\mathbf{MK}$
Crypto Processor
$[\![\boldsymbol{X}]\!][\![\boldsymbol{y}]\!]$
Secure Computations under $\mathbf{PK} = \prod_{i=1}^{N_{\text{users}}} \mathrm{PK}_i$
Application
Regression, Classification...
$[\![\hat{y}]\!] = f_{[\![\boldsymbol{w}]\!]}([\![\boldsymbol{x}]\!])$
$\mathrm{PK}_0, \mathrm{SK}_0, \mathbf{PK}$

**Fig. 1.** Scenario.

the privacy of the algorithm itself, we propose to outsource a reduced set of operations including that one that preserves the ring structure of input cleartexts (in this paper, the modular multiplication).

With regard to the Application's model, in this work we will focus on the Support Vector Machine (SVM) due to its strong mathematical foundations and high reliability in many practical applications [7,8], although the proposed scheme can also be easily extended to many other machine learning algorithms.

In the paper we address two important, but not frequently considered, problems in the training and testing of SVMs on encrypted data.

The first one is that encryption requires integer-valued cleartext data, and therefore, the use of finite precision arithmetic both to represent the real-valued training examples and the intermediate results. Many of the traditional optimization algorithms for SVMs are impractical to implement under finite precision conditions [9], and fail when the number of bits that represent the real valued training examples and the intermediate results is insufficient [10,11]. Besides, conventional SVM optimization algorithms make extensive use of comparison routines (like max or *arg max*), which require complex and slow interactive protocols when implemented in the encrypted domain. For all these reasons, we have chosen the Primal Estimated sub-Gradient Solver (PEGASOS) described in [12], because it is a very simple gradient-descent method, only requiring one comparison per iteration, which provides good performance (in terms of rate of convergence and predictive error) under finite word-length effects.

The second problem is that the classifier learned by the SVM contains some intact instances of the training data (the support vectors), which inherently violates the privacy of the training data. In order to avoid this privacy breach, we propose a semiparametric SVM model that internally builds the classifier by means of some data prototypes not included in the training set. The semiparametric SVM models not only keep the size of the machines under control, but they also maintain, and sometimes even improve, the performance and generalization abilities of the machine. The aforementioned prototypes can be obtained by minimizing the error in the approximation of the kernel matrix as in [13,14], or with a clustering procedure as in [15]. In particular, in this paper, we compare two clustering approaches, namely, the standard k-Means clustering algorithm [16] and the Frequency-Sensitive Competitive Learning (FSCL) [17]. As we will see later, the combination of the gradient method PEGASOS and the FSCL clustering provides good results under the restrictions imposed by the considered cryptographic scenario (finite precision arithmetic and minimum use and complexity of the outsourced operations).

### 1.2. Related work

Works in private (or "privacy-preserving") machine learning have followed three major directions: the randomization/perturbation of the data, the use of secure multiparty computation protocols, and the cryptographic approach. In the randomization approach, data is locally perturbed (adding "noise", blocking or swapping attributes), before data processing is done [18]. However, data perturbation may not protect privacy completely [19], and the analytics based on the perturbed data might not be accurate enough. An alternative approach is to use Secure Multiparty Computation (SMC) protocols [20]. In SMC, data owners collaborate, in a distributed way, to develop a global model keeping their sensitive data hidden (for example, by encrypting or sharing among the protocol participants). Specifically, secure methods based on partial homomorphic cryptosystems have been proposed for clustering [21,22], classification [23,24], and regression [25].

Many of the conventional SMC solutions rely on the direct involvement of users in those operations not supported by the partial homomorphism. For this reason, and taking into account the growing importance of the Cloud Computing paradigm, these operations can be outsourced to an untrusted server, which performs the computation on behalf of the users. This 3-party scenario (single secret-key Data Owners, Service Provider, and untrusted server) has been already used successfully to generate private recommendations [5], in face recognition [26], or in privacy-preserving e-Health and multimedia cloud processing [4]. One limitation of the previous solutions is that the Data Owners have no control on the access authorization to their data. And furthermore, they do not consider mutually distrusting Data Owners, each one having its own pair of public and private keys.

As it has been stated before, encryption requires integer-valued cleartext data. Therefore, a machine learning algorithm in the encrypted domain must be designed to operate on finite-wordlength data, keeping all intermediate results within the required precision and range. With respect to this, and focusing in the SVM, there are some works that study and reduce the impact of finite precision effects on the computations. Most of them only consider the *feed-forward* phase of the SVM, assuming that the training was carried out with infinite precission [27,28]. One of the few papers addressing the training under finite precision arithmetic is [29], in which the authors implement a *hardware friendly* version of the Sequential Minimal Optimization (SMO) [30]. Although the robustness against finite precision effects (*hardware friendly*) is a necessary condition for implementing an algorithm in the encrypted domain, there is another condition that makes an algorithm *crypto-friendly*: the protocols and interactions between parties should be secure, simple and efficient. Thus, although the SMO algorithm requires simple arithmetic operations like multiplications and divisions, it also makes an extensive use of comparison routines, which in turn require complex and slow interactive protocols when implemented in the encrypted domain.

Perhaps the biggest problem of SVMs is that its output function uses *intact* instances of the training data (the support vectors), thus violating the privacy-preservation requirements. One of the few works addressing this problem is [31], where the authors approximate the learned SVM decision function with a (low order) Taylor Polynomial. Obviously, the main drawback of this approach, apart from introducing additional errors, is that it fails when the decision function is highly nonlinear. The Semiparametric SVMs provide a way to maintain the performance without compromising the privacy of the data. Thus, the projection of the training data onto an intermediate-dimensional space defined by a set of prototypes has been used to provide a privacy-preserving mechanism in data distributed scenarios [32].

### 1.3. Organization

The paper is structured as follows. We start by presenting the basic learning scenario in Section 2. Then, we introduce our solution in Section 3. Next, in Section 4, we go into further details on the implementation and optimization of a SVM machine using encrypted examples. Section 5 describes the basic secure operations and protocols needed for the implementation of the previous learning algorithms. An analysis of the performance, in terms of predictive ability (Area under ROC Curve, AuC), complexity, storage, and communication costs, of the developed protocols is given in Section 7. Finally, Section 8 summarizes our paper and provides our concluding thoughts.

## 2. Problem statement

As it has been stated before, the focus of this paper is on the learning of parameters of a model (or function) using private encrypted training data.

### 2.1. The encrypted domain

Cryptosystems are designed and used to preserve confidentiality. They are composed of three algorithms: key generation (KeyGen), encryption (Enc) and decryption (Dec). Obviously, their security relies on the secrecy of the decryption key. In contrast to symmetric cryptosystems (where the two parties share a single common key), public-key cryptosystems employ two separate keys: the encryption key, which is public, and the decryption key, which is private [33]. Thus, in a public-key cryptosystem:

1. The key generation algorithm (KeyGen) takes the security parameter $N$ as input and outputs a pair of keys (PK, SK), the public key and the private key or secret key.
2. The encryption algorithm (Enc) takes as inputs a public-key PK, and a plaintext integer $m$ from some underlying message space. It produces an (integer) ciphertext $\mathsf{Enc}(m, \mathrm{PK}) = [\![m]\!]$ from an underlying ciphertext space.
3. The decryption algorithm (Dec) takes a private-key SK and a ciphertext $[\![m]\!]$ as inputs, and produces an output message $m = \mathsf{Dec}([\![m]\!], \mathrm{SK})$.

It is important to note that cryptosystems operate on plaintext messages that take values from a finite alphabet. Additionally, the cryptographic operations are carried out in finite fields involving modular artihmetic operations on integers. Therefore, in order to encrypt signals or data taking real values, it is necessary to previously perform a format conversion, or quantization, into an integer representation.

### 2.2. The homomorphic property

As it was mentioned before, homomorphic cryptosystems perform a set of operations on encrypted data without knowledge of the decryption key. For example, an additive homomorphic operation allows the computation of the linear combination of two plaintexts through ciphertext manipulation:

$$[\![\alpha m_1 + \beta m_2]\!] = [\![m_1]\!]^{\alpha} \cdot [\![m_2]\!]^{\beta} \tag{1}$$

where $\alpha$ and $\beta$ are plaintext signed integer constants ($[\![m]\!]^{-1}$ is the modular multiplicative inverse: $[\![m]\!] \cdot [\![m]\!]^{-1} \mod N = 1$; the group of invertible integers modulo $N$ is denoted by $\mathbb{Z}_N^*$).

### 2.3. The scenario

The most simple machine learning scenario has the following participants:

1. On one side, different and mutually distrusted Data Owners, or Users, each one holding its respective public and private keys of a partial homomorphic cryptosystem. For simplicity, we will assume that users' data are arranged in the form of mean-corrected real-valued vectors of fixed size $d$ ($\boldsymbol{x} \in \mathbb{R}^d$). Thus, the users' data have to be converted into a finite precission representation before being encrypted. In this paper we will use the $Q_{n_i.n_f}$ Fractional Fixed-Point Representation [34], where $n_i$ is the number of bits in the integer part (exclusive of the sign), and $n_f$ the same for the fractional part.[2] After the integer conversion, Data Owners encrypt (with its own public key) the data, ($[\![\boldsymbol{x}]\!], [\![y]\!]$) and, afterwards, upload them to an untrusted *Storage Server*. In order to keep privacy and control, any request for accessing or processing data by a third-party must be previously authorized by Data Owners.

[2] Negative numbers are mapped into the upper half of the interval [0, $N$) using the cyclic (or modular) property of the cryptosystem.

2. On another side, the Application, which wants to adjust the parameters, $\boldsymbol{w}$, of a function $f_{\boldsymbol{w}}(\cdot)$, so that it approximates a given, but unknown, input/output mapping. After being authorized by the Data Owners, the Application is allowed to access the multiple-key encrypted training dataset $\{[\![\boldsymbol{x}_i]\!], [\![y_i]\!]\}_{i=1}^m = ([\![\boldsymbol{X}]\!], [\![\boldsymbol{y}]\!])$. The function parameters have to be adjusted without disclosing any private inputs nor intermediate results. Also, the training has to be carried out without involving neither the Data Owners nor the Storage Server.

Regarding the security model, we consider the semi-honest adversarial model (honest-but-curious) [35], meaning that all participants agree to follow a given protocol/algorithm, but trying to gather information about other parties' inputs, intermediate results, or overall outputs. This model uses blinding and randomization techniques to assure privacy protection and it is widely used in the privacy protection community. Nevertheless, if more aggressive attackers (malicious adversaries) participating in the computation are considered, protocols secure in the semi-honest model can be transformed to be secure against malicious adversaries using commitment schemes and zero-knowledge proofs. These techniques increase the interactions between the participants making slower the protocol (up to an order of ten), [26].

## 3. Our proposed scenario

Our solution, represented in Fig. 1, involves multiple-key encryption, and a third participant: the *Authorization Server*. Basically, the Authorization Server interfaces between the Data Owners and the Application, by managing the access to the Users' data, and assisting the Application in those computations on encrypted data not supported by the cryptosystem.

### 3.1. Multiple-key encryption

Solutions based on traditional Secure Multiparty Computations (SMC) require the whole dataset to be encrypted using a common (and single) public key. In the more realistic case in which the dataset is horizontally partitioned (all the instances have the same attributes) between different and mutually distrusting Data Owners, solutions based on single public key encryption are no longer valid. As it was mentioned in Section 2, Data Owners encrypt data using multiple unrelated public keys, $\mathrm{PK}_i$, so a multiple-key cryptosystem turns out to be mandatory. In this paper, we will use the BCP cryptosystem by Bresson, Catalano and Pointcheval [36], which is additively homomorphic (i.e., it allows the addition of plaintexts in the encrypted domain), and offers two independent decryption mechanisms: the first one depends on the secret key of each Data Owner, $\mathrm{SK}_i$; and the second decryption mechanism depends on a master secret key, $\mathbf{MK}$, that is stored on the Authorization Server.

#### 3.1.1. BCP Cryptosystem

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

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

#### 3.1.2. The authorization server

The Authorization Server (or Auth Server) has two responsibilities.

- Firstly, the Authorization Server is an intermediary between the Users and the Applications that allows the latter to access the Users' data. For this purpose, we propose to use as a basis the OAuth 2.0 authorization framework [37], in which Applications request access grants to the Authorization Server, which issues an access token, with a previous permission of the Users (see Fig. 2).
- The second one is to provide privacy services to the parties involved in the process. In this regard, the Authorization Server sets up the selected cryptosystem and distributes the public encryption parameters $\mathbf{PP}$ to Users and Applications. Users, with their $\mathsf{KeyGen}$ primitives, generate their respective public and secret keys. Finally, the Authorization Server provides the Application with a single public key $\mathbf{PK}$ under which the encryptions of all participating users are reencrypted without disclosing nor changing the underlying cleartexts.

Within the privacy responsibility, the Authorization Server will assist Applications, with no collusion between them, in a limited set of (previously agreed) secure computations on encrypted data not supported by the partially homomorphic cryptosystem. In particular, we propose to use a minimal set that includes the other operation that preserves the ring structure of input plaintexts, which in the case under consideration (BCP cryptosystem) is the modular multiplication ($a \cdot b \bmod N$), a few basic logical (bit shifting) and conditional branching instructions (comparison). This tightly limited set of instructions could be executed directly on the encryption/decryption hardware, or CryptoProcessor [38], of the Authorization Server, without requiring additional computational resources (we assume that the CryptoProcessor has a flexible architecture to implement different cryptosystems [39]). In addition, the encoding of a high-level algorithm into such reduced set of instructions can be seen as a means of protecting the privacy itself of the high-level functions/routines (exponentiations, divisions, matrix inversions) involved in the learning algorithm.

Finally, it is important to mention that the honest-but-curious assumption requires to protect the privacy of the outsourced operators. In this sense, we will use secure protocols based on blinding and the additively homomorphic property of the underlying cryptosystem [26] to hide the exchanged data.

#### 3.1.3. The application

As stated above, the objective of the Application is to learn the *same* parameters as those that would be obtained if the training data were in cleartext. For this purpose, the Application is allowed to run interactive protocols with the Authorization Server on the encrypted Users' Data. Obviously, the protection of the Users' Data privacy is essential, just as it is the protection of the privacy of the propietary machine learning algorithm instructions, and the

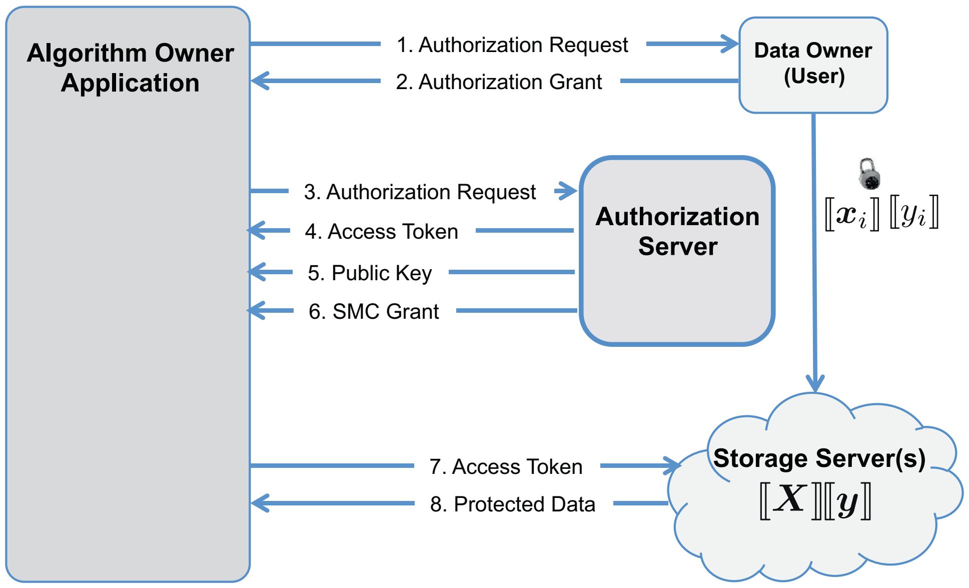

**Fig. 2.** Authorization protocol flow.

parameters of the machine learning algorithm. For this reason, in this work we consider the Application as another user, having its own pair of public and secret keys ($\mathrm{PK}_0$, $\mathrm{SK}_0$). This way, the Application could get access to the cleartext result of comparisons on encrypted data, which is required in some conditional branching instructions.

### 3.2. Operation

The basic idea of our proposal consists of the following steps:

#### 3.2.1. Authorization

Once the Application receives an Authorization Grant from the User, it requests an Access Token from the Auth Server by presenting the authentication of its own identity and the User's Authorization Grant. If the Application identity is authenticated and the Authorization Grant is valid, the Auth Server issues an Access Token and a public-key to reencrypt the encryptions of the involved Users. The Application asks the Storage Servers to get access to the Users' private data. If the Access Token is valid, the Storage Servers send the multi-key encrypted data to the Application based on OAuth 2.0.

#### 3.2.2. Reencryption

After collecting the individually encrypted inputs, the Application runs an SMC protocol with the Auth Server in order to transform the given data (encrypted under the Users' public-keys) into encryptions under a new single public key. In our proposal, we use the approach presented in [6] by Peter et al., in which the reencryption is done using the product of all involved public-keys ($\mathbf{PK} = \prod_i \mathrm{PK}_i \mod N$).

#### 3.2.3. Secure learning

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

#### 3.2.4. Result retrieval

Once the secure learning phase is completed, the output of the function (encrypted under the product of all keys) is ready to be sent back to the Users. And to do this, the Application runs a final SMC protocol with Auth Server in order to transform the output back into encryptions under the Users' respective public keys.

## 4. Training SVMs in the encrypted domain

Support Vector Machines (SVMs) have been extensively used in machine learning and data mining tasks. In binary classification, the SVM learns the decision function $f(\boldsymbol{x}) = \mathrm{sign}(\langle \boldsymbol{w}, \phi(\boldsymbol{x})\rangle + b)$ from a training set $\{\boldsymbol{x}_i, y_i\}_{i=1}^m$, where $\boldsymbol{w}$ and $b$ are the coefficients of the separating hyperplane, and $\phi(\boldsymbol{x})$ maps the input vector $\boldsymbol{x}$ into the feature space.

In the primal form, the SVM optimization problem becomes [7]:

$$\min_{\boldsymbol{w}} J(\boldsymbol{w}) = \min_{\boldsymbol{w}} \left\{ \frac{1}{2}\|\boldsymbol{w}\|^2 + C\sum_{i=1}^{m} L(y_i, \langle \boldsymbol{w}, \phi(\boldsymbol{x}_i)\rangle) \right\} \tag{2}$$

where $L(y, f(x)) = \max\{0, 1 - y \cdot f(x)\}$ is the hinge-loss function, and $C$ is the misclassification penalty. In this formulation we have dropped the constant offset $b$ since it can be computed from the Karush-Kuhn-Tucker (KKT) optimality conditions [8].

The corresponding dual optimization problem is:

$$\max_{\boldsymbol{\alpha}\in\mathbb{R}^m} \left\{ \sum_{i=1}^{m}\alpha_i - \frac{1}{2}\sum_{i,j}^{n}\alpha_i\alpha_j y_i y_j k(\boldsymbol{x}_i, \boldsymbol{x}_j) \right\}$$

$$\text{s.t. } 0 \le \alpha_i \le C, \ \sum_{i=1}^{m}\alpha_i y_i = 0, \qquad i = 1, \ldots, m \tag{3}$$

where $k(\boldsymbol{x}_i, \boldsymbol{x}_j) = \phi(\boldsymbol{x}_i)\phi(\boldsymbol{x}_j)$ is a general nonlinear positive semidefinite symmetric kernel. Usually, after solving the dual problem, only a few dual coefficients $\alpha_i$ turn out to be non-zero (dual sparsity). If we denote as $\mathcal{SV}$ the set of indices of the non-zero dual coefficients, the set $\{\boldsymbol{x}_i\}_{i\in\mathcal{SV}}$ are the support vectors, and the decision function can be represented as:

$$f(\boldsymbol{x}) = \sum_{i\in\mathcal{SV}} \alpha_i y_i k(\boldsymbol{x}_i, \boldsymbol{x}). \tag{4}$$

From a privacy-preserving viewpoint, it is important to note that if the Application wants to evaluate the decision function learned by the SVM, it needs to get access to the cleartext version of the support vectors and their corresponding labels, $\{\boldsymbol{x}_i, y_i\}_{i \in \mathcal{SV}}$, which, in fact, are private.

### 4.1. Sequential minimal optimization algorithm

Sequential Minimal Optimization (SMO) algorithm [30] is a specialized optimization approach for the SVM dual Quadratic Program. Following the heuristic proposed by Keerthi et al. [40], the training points are partitioned into 5 subsets, represented by their indices:

1. (Unbounded SVs) $I_0 = \{i : 0 < \alpha_i < C\}$
2. (Positive NonSVs) $I_1 = \{i : y_i > 0, \alpha_i = 0\}$
3. (Bounded Negative SVs) $I_2 = \{i : y_i < 0, \alpha_i = C\}$
4. (Bounded Positive SVs) $I_3 = \{i : y_i > 0, \alpha_i = C\}$
5. (Negative NonSVs) $I_4 = \{i : y_i < 0, \alpha_i = 0\}$.

From these 5 subsets, two sets of indexes are defined: $\mathcal{S}_{\text{high}} : I_0 \cup I_1 \cup I_2$, and $\mathcal{S}_{\text{low}} : I_0 \cup I_3 \cup I_4$. At each step of the algorithm, the SMO finds the two weights that maximally violate the KKT conditions: $(I_{\text{high}}, e_{\text{high}}) = (arg\,max_{\mathcal{S}_{\text{high}}} \boldsymbol{e}, \max_{\mathcal{S}_{\text{high}}} \boldsymbol{e})$, and $(I_{\text{low}}, e_{\text{low}}) = (arg\,min_{\mathcal{S}_{\text{low}}} \boldsymbol{e}, \min_{\mathcal{S}_{\text{low}}} \boldsymbol{e})$, where the vector $\boldsymbol{e} = \sum_{j=1}^{m} \alpha_j y_j k(x_i, x_j) - y_i,\ i = 1, \ldots m$ tracks the optimality conditions. Finally, the SMO updates the two maximum violating elements of $\boldsymbol{\alpha}$:

$$\alpha'_{I_{\text{low}}} = \alpha_{I_{\text{low}}} + \frac{y_{I_{\text{low}}}(e_{I_{\text{high}}} - e_{I_{\text{low}}})}{\eta} \tag{5}$$

$$\alpha'_{I_{\text{high}}} = \alpha_{I_{\text{high}}} + y_{I_{\text{high}}} y_{I_{\text{high}}} (\alpha_{I_{\text{low}}} - \alpha'_{I_{\text{low}}}), \tag{6}$$

where $\eta = k(\boldsymbol{x}_{I_{\text{high}}}, \boldsymbol{x}_{I_{\text{high}}}) + k(\boldsymbol{x}_{I_{\text{low}}}, \boldsymbol{x}_{I_{\text{low}}}) - 2k(\boldsymbol{x}_{I_{\text{high}}}, \boldsymbol{x}_{I_{\text{low}}})$.

The SMO algorithm is summarized in Procedure 1:

**Procedure 1** Sequential minimal optimization.

```
Input: $\{\boldsymbol{x}_i, y_i\}_{i=1}^m$, $C$, $\tau$.
  Initialize: set $\boldsymbol{\alpha} = 0$, $\boldsymbol{e} = -y_i, \forall i \in \{1, \ldots, m\}$
  Compute: $I_{\text{high}}, e_{\text{high}}, I_{\text{low}}, e_{\text{low}}$
  Update: $\alpha_{I_{\text{high}}}$ and $\alpha_{I_{\text{low}}}$
  repeat
    Update: $\boldsymbol{e}$
    Compute: $I_{\text{high}}, e_{\text{high}}, I_{\text{low}}, e_{\text{low}}$
    Update: $\alpha_{I_{\text{high}}}$ and $\alpha_{I_{\text{low}}}$
  until $e_{\text{low}} \le e_{\text{high}} + 2\tau$
```

#### 4.1.1. Implementation in the encrypted domain

From an arithmetic point of view, the implementation of the SMO algorithm in the encrypted domain requires to precompute the (encrypted) kernel matrix $\boldsymbol{K}$, where $\boldsymbol{K}[i, j] = k(\boldsymbol{x}_i, \boldsymbol{x}_j)$. In order to reduce the computational complexity of the kernel evaluation, we modify the conventional gaussian kernel by using 2 as the base of the exponentiations, which is more *hardware friendly* [27]. This way, the kernel matrix becomes:

$$\boldsymbol{K}[i, j] = k(\boldsymbol{x}_i, \boldsymbol{x}_j) = 2^{-\frac{\|\boldsymbol{x}_i - \boldsymbol{x}_j\|^2}{2\sigma^2}}, \tag{7}$$

which can be obtained by computing $m \cdot (m-1)$ encrypted distances between $d$-dimensional vectors, $m \cdot (m-1)$ products, and $m \cdot (m-1)$ bit-shifting operations.

Besides this operation, and the sums supported by the additive homomorphic cryptosystem, the SMO requires one division and three products over encrypted data (per iteration) when updating $\alpha_{I_{\text{high}}}$ and $\alpha_{I_{\text{low}}}$. Finally, $m^2$ outsourced products are also needed to compute the vector $\boldsymbol{e}$ in each iteration.

Unfortunately, the SMO makes extensive use of comparisons and the derived max, min, $arg\,min$ and $arg\,max$ operations over encrypted data. Just as an illustration, one single $arg\,max$ operation over a vector of dimension $m$ requires $m-1$ comparisons, and $7(m-1)$ outsourced products encrypted integers [24].

In order to reduce the number of comparisons, the algorithm is stopped after a fixed number of iterations, $T_{\text{SMO}}$. To produce an $\epsilon$-accurate solution $\tilde{\boldsymbol{w}}$ ($J(\tilde{\boldsymbol{w}}) \le \min_{\boldsymbol{w}} J(\boldsymbol{w}) + \epsilon$, where $J(\boldsymbol{w})$ is defined in Eq. (2)), the SMO requires $T_{\text{SMO}} \sim \mathcal{O}(m/\epsilon)$ iterations [41].

### 4.2. Primal estimated sub-gradient solver for SVM

The Primal Estimated sub-Gradient Solver for SVM, PEGASOS [12], is a stochastic algorithm that, in each iteration, operates on a single training example chosen at random. Setting $\lambda = 1/(Cm)$ and replacing the objective function in (2) with an approximation based on the training example $(\boldsymbol{x}_{i_t}, y_{i_t})$ yields the objective function:

$$J(\boldsymbol{w}; i_t) = \frac{\lambda}{2}\|\boldsymbol{w}\|^2 + L(y_{i_t}, \langle \boldsymbol{w}, \phi(\boldsymbol{x}_{i_t}) \rangle) \tag{8}$$

If we obtain the sub-gradient of the previous approximate objective, the vector $\boldsymbol{w}$ can be updated with the following equation:

$$\boldsymbol{w}_{t+1} = \frac{1}{\lambda t} \sum_{i=1}^{t} \mathbb{1}[y_{i_t} \langle \boldsymbol{w}, \phi(\boldsymbol{x}_i) \rangle < 1] y_{i_t}, \phi(\boldsymbol{x}_{i_t}), \tag{9}$$

where $\mathbb{1}\left[y_{i_t} \langle \boldsymbol{w}, \phi(\boldsymbol{x}_i) \rangle < 1\right]$ takes value of one if its argument is true (the SVM yields non-zero loss on the example $(\boldsymbol{x}_{i_t}, y_{i_t})$), and zero otherwise Procedure 1.

The PEGASOS algorithm can be implemented using only kernel evaluations, without direct access to the feature vectors $\phi(\boldsymbol{x})$ or explicit access to the weight vector $\boldsymbol{w}$. Thus, the Kernelized PEGASOS Algorithm is shown in Procedure 2:

**Procedure 2** The kernelized PEGASOS algorithm.

```
Input: $\{\boldsymbol{x}_i, y_i\}_{i=1}^m$, $C$,$T_{\text{PEG}}$.
  Initialize: set $\boldsymbol{\alpha}_0 = 0$, $\lambda = 1/(Cm)$.
  for $t = 1, 2 \ldots, T_{\text{PEG}}$ do
    $\boldsymbol{\alpha}_t = \boldsymbol{\alpha}_{t-1}$
    Choose $i_t \in_R \{1, \ldots, m\}$ uniformly atrandom
    if $y_{i_t} \frac{1}{\lambda t} \sum_{j=1}^{m} \alpha_t[j] y_j k(\boldsymbol{x}_{i_t}, \boldsymbol{x}_j) < 1$ then
      set $\alpha_{t+1}[i_t] = \alpha_t[i_t] + 1$
    end if
  end for
Output: $\boldsymbol{\alpha}_{T_{\text{PEG}}+1}$.
```

Since the run-time does not linearly depend on the size of the training set (it requires $T_{\text{PEG}} \sim \mathcal{O}(1/\epsilon)$ iterations to achieve an $\epsilon$-accurate solution [12], the PEGASOS algorithm is especially well suited for learning from large datasets.

#### 4.2.1. Implementation in the encrypted domain

The PEGASOS algorithm is much simpler than the SMO. In terms of outsourced operations, and again assuming that the kernel matrix is precomputed, it requires $m+1$ products; with respect to relational operations, it requires only one comparison of two encrypted integers per iteration.

### 4.3. Semiparametric SVM

As already mentioned, it is important that the resulting model is of moderate size, and even more fundamental, that no training patterns are present in the final model architecture to preserve privacy. Our proposal is to use a small set of prototypes $\boldsymbol{c}_j$ to build the following compact model:

$$f(\boldsymbol{x}) = \langle \boldsymbol{w}, \boldsymbol{r}(\boldsymbol{x})\rangle \tag{10}$$

where

$$\begin{aligned}\boldsymbol{r}(\boldsymbol{x})[0] &= 1\\ \boldsymbol{r}(\boldsymbol{x})[j] &= k(\boldsymbol{c}_j, \boldsymbol{x})\end{aligned} \tag{11}$$

We need to estimate values $\boldsymbol{c}_j$ in (11) beforehand using one of the previously mentioned clustering methods. Finally, we need to solve the following optimization problem to find weights $\boldsymbol{w}$:

$$\min_{\boldsymbol{w}} \left\{ \frac{\lambda}{2}\|\boldsymbol{w}\|^2 + \sum_{i=1}^{m} L(y_i, \langle \boldsymbol{w}, \boldsymbol{r}(\boldsymbol{x}_i)\rangle) \right\} \tag{12}$$

We compute the sub-gradient for (12) which, at every step, is:

$$\nabla_t = \lambda \boldsymbol{w}_t - \mathbb{1}[y_{i_t}\langle \boldsymbol{w}_t, \boldsymbol{r}(\boldsymbol{x}_{i_t})\rangle < 1] y_{i_t} \boldsymbol{r}(\boldsymbol{x}_{i_t}) \tag{13}$$

If we iteratively apply the subgradient step on randomly chosen patterns, we obtain the procedure named as Semiparametric PEGASOS (SP), as summarized in Procedure 3. Depending on which clustering algorithm we use to find the prototypes, we obtain the variants SP-kmeans or SP-FSCL.

**Procedure 3** The semiparametric PEGASOS algorithm.

```
Input: $\{\boldsymbol{x}_i, y_i\}_{i=1}^m$, $C$, $T_{\mathrm{PEG}}$.
  Obtain prototypes $\{\boldsymbol{c}_j\}_{j=1}^R$ using either k-Means or FSCL.
  Initialization: set $\boldsymbol{w}_0 = 0$, $\lambda = 1/(Cm)$.
  for $t = 1, 2 \ldots, T_{\mathrm{PEG}}$ do
    $\eta_t = 1/(\lambda t)$
    Choose $i_t \in_R \{1, \ldots, m\}$ uniformly at random
    Build $\boldsymbol{r}(\boldsymbol{x}_{i_t})$:$\boldsymbol{r}(\boldsymbol{x}_{i_t})[0] = 1$, $\boldsymbol{r}(\boldsymbol{x}_{i_t})[j] = k(\boldsymbol{x}_{i_t}, \boldsymbol{c}_j)$
    if $y_{i_t}\langle \boldsymbol{w}_t, \boldsymbol{r}(\boldsymbol{x}_{i_t})\rangle < 1$ then
      set $\boldsymbol{w}_{t+1} = (1 - \eta_t\lambda)\boldsymbol{w}_t + \eta_t y_{i_t}\langle \boldsymbol{w}_t, \boldsymbol{r}(\boldsymbol{x}_{i_t})\rangle$
    else
      set $\boldsymbol{w}_{t+1} = (1 - \eta_t\lambda)\boldsymbol{w}_t$
    end if
  end for
Output: $\boldsymbol{w}_{T_{\mathrm{PEG}}+1}$.
```

#### 4.3.1. Implementation in the encrypted domain

Apart from the operations needed by the clustering algorithm, the semiparametric version of the PEGASOS algorithm takes, in terms of outsourced operations, one inner product of two encrypted vectors of dimension $R$, which in turn requires $R$ element-wise products, and 2 scaling operations on vectors of dimension $R$, and 3 additional products per iteration. In addition, one comparison between two encrypted integers is also needed in each iteration.

### 4.4. Clustering

One way of implementing a clustering procedure in a $d$-dimensional space is to regard it as the minimization of a cost function

$$J(\boldsymbol{c}_j) = \sum_{j=1}^{R}\sum_{i\in\mathcal{S}_j} D(\boldsymbol{x}_i, \boldsymbol{c}_j) \tag{14}$$

where $R$ is the total number of clusters, $\boldsymbol{c}_j$ are the cluster representatives, $D(\boldsymbol{x}_i, \boldsymbol{c}_j)$ is a distance measurement between the vectors $\boldsymbol{x}_i$ and $\boldsymbol{c}_j$, and $\mathcal{S}_j$ is the set of indices of the vectors $\boldsymbol{x}_i$ which are closer to the cluster representative $\boldsymbol{c}_j$ than the other representatives. The Euclidean distance measure $D_E(\boldsymbol{x}_i, \boldsymbol{c}_j) = \|\boldsymbol{x}_i - \boldsymbol{c}_j\|^2$ is often used in many practical situations.

#### 4.4.1. k-Means

The k-Means clustering algorithm consists of two steps: cluster assignment and cluster updating. In cluster assignment all data points are assigned to one of the clusters by

$$i \in \mathcal{S}_j \Leftrightarrow j = arg\,min_k D(\boldsymbol{x}_i, \boldsymbol{c}_k) \tag{15}$$

In cluster updating the cluster representative is updated by minimizing $J(\boldsymbol{c}_j)$, leading to

$$\boldsymbol{c}_j = \frac{\sum_{i\in\mathcal{S}_j} \boldsymbol{x}_i}{N_j} \tag{16}$$

where $N_j$ is the total number of data points that is assigned to cluster $j$.

The algorithm works as follows. Initially, $R$ cluster representatives are randomly positioned in the feature space. Then, cluster assignment is done by assigning every data point to one cluster using (15). Cluster updating is globally performed by updating every cluster centre using (16). Both cluster assignment and updating stages are performed alternately until the maximum number of iterations, $T_{\mathrm{kM}}$ is reached.

From an operational point of view, the main drawbacks of the k-Means algorithm are its sensitivity to the initial placement of clusters centers, its bad performance when working with unevenly sized clusters (it gives more "weight" to larger clusters), and its easiness to get trapped into a local minimum. In addition, and now from a cryptographic point view, the main drawback is that the cluster assignment (15) needs to evaluate $R \times m$ (Euclidean) distances between $d$-dimensional vectors, $(R-1) \times m$ comparisons and one division over encrypted values per iteration, where $m$ is the data set size. Another important problem is that the theoretical bounds on the maximum number of iterations depends exponentially on the number $R$ of clusters ($T_{\mathrm{kM}} \sim \mathcal{O}(2^{\mathcal{O}(R)} \cdot m \cdot d)$) [42]. As a consequence, for large training data sizes, the implementation in the encrypted domain of the k-Means results in very slow and costly protocols [21,22].

#### 4.4.2. Frequency-Sensitive Competitive Learning

Frequency-Sensitive Competitive Learning (FSCL) [17] performs clustering by means of a gradient descent optimization of (14). At each iteration only one data point is presented and assigned to a cluster using (15). In order to avoid imbalanced clusters, the distance measure is modified in order to favour cluster centers with low update frequencies:

$$D_{FS}(\boldsymbol{x}_{i_t}, \boldsymbol{c}_k) = N_k D_E(\boldsymbol{x}_{i_t}, \boldsymbol{c}_k), \tag{17}$$

where $N_k$ is estimated as the number of times that cluster $k$ has been updated so far:

$$N_k^{(t+1)} = \begin{cases} N_k^{(t)} + 1, & k = arg\,min_j\, D_{FS}(\boldsymbol{x}_{i_t}, \boldsymbol{c}_j) \\ N_k^{(t)} & \text{otherwise.} \end{cases} \tag{18}$$

Finally, the representative of the cluster is updated as:

$$\boldsymbol{c}_j^{(t+1)} = \boldsymbol{c}_j^{(t)} + \mu(\boldsymbol{x}_{i_t} - \boldsymbol{c}_j^{(t)}), \tag{19}$$

where $\mu$ is a learning parameter that decreases with time $t$.

The FSCL algorithm works as follows. During the first iteration steps, the same distance measurement is used for all clusters. This will lead to an overassignment of the smaller clusters. As a result, during the next iterations, the distance measurements for these

**Table 1**
Complexity of SVM training and clustering algorithms per iteration.

| Algorithm | Products | Divisions | Comparisons | Iterations |
|---|---|---|---|---|
| SMO | $m^2+3$ | 1 | $8m-2$ | $\mathcal{O}(m/\epsilon)$ [41] |
| PEGASOS | $3m+3$ | – | 1 | $\mathcal{O}(1/\epsilon)$ [12] |
| k-Means | $R\cdot m\cdot d$ | $R$ | $(R-1)\cdot m$ | $\mathcal{O}(2^{\mathcal{O}(R)}\cdot m\cdot d)$ [42] |
| FSCL | $R\cdot(d+1)$ | – | $R-1$ | $\mathcal{O}(R\cdot m)$ [43] |
| PEGASOS SP | $3R+3$ | – | 1 | $\mathcal{O}(1/\epsilon)$ [12] |

clusters become larger than for the other ones, hereby equalizing again the number of assignments. It is balancing between different weights for the distance measure that finally leads to an equal treatment of the assignment step for all clusters. From a cryptographic point view, the FSCL requires to compute $R$ Euclidean distances, $R-1$ comparisons, and $R$ products per iteration.

### 4.5. Computational complexity

Table 1 summarizes the computational complexity per iteration of the previous SVM training and clustering algorithms. It is important to notice that the number of divisions and comparisons appears for comparison purposes, since both operations have to be implemented using the products and bit-wise shifting and relational operations available at the cryptoprocessor side. In order to ensure a fair comparison, some bounds on the number of iterations required to achieve convergence are also given (for SVM training algorithms, bounds are given in terms of the accuracy of the solution $\epsilon$).

From Table 1 some conclusions can be drawn. First, and focusing on the comparison between PEGASOS and SMO, the PEGASOS algorihtm is much simpler ($\mathcal{O}(m)$ vs. $\mathcal{O}(m^2)$) and faster ($\mathcal{O}(1/\epsilon)$ vs. $\mathcal{O}(m/\epsilon)$) than the SMO. In addition, and now focusing on the suitability for the implementation in the encrypted domain, the PEGASOS algorithm minimizes the number of both outsourced operations and exchanged data. Regarding the Clustering algorithms for the Semiparametric solution, the computational complexity of the k-Means method is $m$-times higher than that of the FSCL solution (being $m$ the number of training instances). In addition, the running time of the FSCL is linear with respect to the number $R$ of centroids (in contrast to the exponential behavior of k-Means) [43].

## 5. Basic operations in the encrypted domain

In this section, we will describe how to implement some basic arithmetic and relational functions required to train and optimize the parameters of the Application's model using encrypted data.

It is assumed that the examples are encrypted by an additively homomorphic cryptosystem, so that additions and substractions (using the multiplicative inverse) can be obtained directly by the Application. Any other computation not supported by the homomorphic cryptosystem has to be implemented with the assistance of the Authorization Server.

It is also assumed that the Authorization Server has an optimized CryptoProcessor [44] with a (limited) set of instructions (including modular multiplications, and relational and logical operations) available to the Application.

In order to provide a clear idea about the complexity of each protocol, we consider three factors: computation (expressed in terms of outsourced operations), communication (describing the amount of bits exchanged between the Application and the Authorization Server), and Storage (counting the number of encrypted values that have to be stored, since the size of plain integers are relatively small).

### 5.1. Secure multiplication: $[\![x]\!]\odot[\![y]\!]$

Our interactive secure multiplication protocol is based on the blinding solution presented in [26].

The Application homomorphically blinds $[\![x]\!]$, and $[\![y]\!]$, with random values $r_x$ and $r_y$, respectively. The Auth Server decrypts $[\![z_x]\!]=[\![x+r_x]\!]$, and $[\![z_y]\!]=[\![y+r_y]\!]$ and computes the product $z_x\cdot z_y$. Then, Auth Server applies a normalization procedure (or truncation) in order to maintain the intermediate results within the required precision ($Q_{n_i.n_f}[z_x\cdot z_y]$) and re-encrypts the result. Note that even though the Auth Server decrypts $z_x$ and $z_y$, it does not learn $x$ and $y$ since these have been blinded by the random values $r_x$ and $r_y$, respectively. Finally, the Application calculates the final result of the secure multiplication as:

$$\begin{aligned}[\![x]\!]\odot[\![y]\!] &= [\![z_xz_y-(xr_y+yr_x+r_xr_y)]\!]\\ &= [\![z_xz_y]\!]\cdot[\![x]\!]^{-r_y}\cdot[\![y]\!]^{-r_x}\cdot[\![r_xr_y]\!]^{-1}\end{aligned}$$

In terms of complexity, each product requires the exchange of 3 encrypted messages ($3\cdot\ell_N$ bits), and the storage of $6\cdot\ell_N$ bits.

#### 5.1.1. Secure inner and matrix product protocols

The product of two encrypted matrices can be obtained by iterating the previous secure multiplication protocol and using the additive homomorphic property. One particular case is the Secure Inner Product Protocol: $[\![\boldsymbol{x}]\!]^T\odot[\![\boldsymbol{y}]\!]\equiv\langle[\![\boldsymbol{x}]\!],[\![\boldsymbol{y}]\!]\rangle$.

#### 5.1.2. Secure Euclidean distance $\pi_{DE}([\![x]\!],[\![c]\!])$

The Secure Inner Product is the basis for the Protocol 1 that computes the (square) Euclidean distance between two encrypted vectors.

### 5.2. Secure division: $\pi_{Div}([\![a]\!],[\![d]\!])$

One of the most popular quadratically converging algorithms for the division is the Newton–Raphson method. This method iteratively calculates a single zero of a function $f(x)$ given an initial guess $x_0$:

$$x_{i+1}=x_i-\frac{f(x_i)}{f'(x_i)} \tag{20}$$

In our case, $x=1/d$, and $f(x)=\frac{1}{x}-d$ (notice that $f(x)=0$). Thus, the Newton–Raphson iteration becomes:

$$x_{i+1}=\left(2x_i-dx_i^2\right), \tag{21}$$

which can be easily implemented in the encrypted domain using the multiplication protocol described in Section 5.1. The final result is obtained after a secure multiplication between $[\![x]\!]$ and $[\![a]\!]$.

#### 5.2.1. Complexity

Our implementation of the Secure Division algorithm is based on the one proposed in [45]. It requires $2\theta+8$ outsourced products, where $\theta$ is the number of Newton–Raphson iterations (typ. $\theta=5$ is usually enough for a resolution of $n_f=32$ bits in the fractional part [46]).

### 5.3. Secure Bit-Shifting Protocol $\pi_{BS}([\![n_b]\!])$

In some arithmetic operations, as the secure multiplication protocol described above, a scaling factor in the form $2^{n_b}$ has to be applied in order to maintain results within the required precision. In this work, we propose the protocol shown in Fig. 3, which takes encryption $[\![n_b]\!]$ as input and returns $[\![2^{n_b}]\!]$.

**Protocol:** Square Euclidean Distance: $[\![s]\!] = \pi_{\mathrm{DE}}([\![\boldsymbol{x}]\!], [\![\boldsymbol{c}]\!])$

Application:

Input: encrypted private data vectors $[\![\boldsymbol{x}]\!]$ and $[\![\boldsymbol{c}]\!]$

Output: encrypted distance $[\![s]\!] = \pi_{\mathrm{DE}}([\![\boldsymbol{x}]\!], [\![\boldsymbol{c}]\!])$

**1.** Obtain vector $\boldsymbol{d}_{\mathrm{PK}} = [\![\boldsymbol{x} - \boldsymbol{c}]\!] = [\![\boldsymbol{x}]\!] \cdot [\![\boldsymbol{c}]\!]^{-1}$

**2.** Secure Inner Product $[\![s]\!] = \boldsymbol{d}_{\mathrm{PK}}^T \odot \boldsymbol{d}_{\mathrm{PK}}$

**Protocol 1.** Euclidean distance protocol.

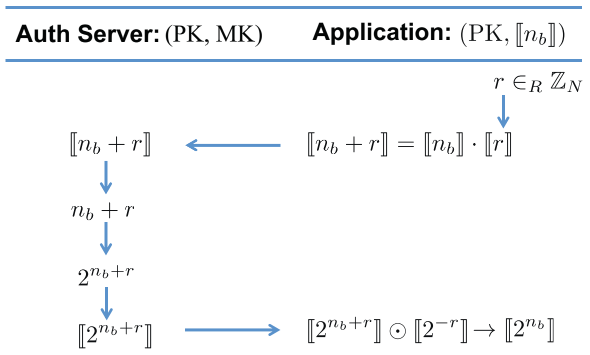
Auth Server: (PK, MK)
Application: (PK, $[\![n_b]\!]$)
$r \in_R \mathbb{Z}_N$
$[\![n_b + r]\!]$
$[\![n_b + r]\!] = [\![n_b]\!] \cdot [\![r]\!]$
$n_b + r$
$2^{n_b+r}$
$[\![2^{n_b+r}]\!]$
$[\![2^{n_b+r}]\!] \odot [\![2^{-r}]\!] \rightarrow [\![2^{n_b}]\!]$

**Fig. 3.** Secure Bit-Shifting Protocol. $\pi_{\mathrm{BS}}([\![n_b]\!])$.

*5.3.1. Complexity*

Each invocation of the Secure Bit-shifting protocol requires 2 outsourced operations (the bit-shifting itself and one extra product), the exchange of $5\ell_N$ bits, and the storage of $7\ell_N$ bits. Thus, the computation of the encrypted SVM Kernel Matrix corresponding to $m$ training instances of dimension $d$, and $R$ centroids ($R = m - 1$ for conventional SVM solutions) requires $(m \cdot R \cdot d)$ operations, and the exchange of $\mathcal{O}(m \cdot R \cdot d \cdot \ell_N)$ bits and the storage of $\mathcal{O}(m \cdot R \cdot \ell_N)$ bits.

### 5.4. *Secure comparison* $\pi_{a<b}([\![a]\!], [\![b]\!])$

Now, the problem is to compare two encrypted (and unsigned) numbers, $[\![a]\!]$ and $[\![b]\!]$, each one of $\ell$ bits in cleartext ($0 \le a, b < 2^\ell$, and $\ell < <\log_2(N)$).

We have used as a basis the protocol proposed in [47] and [48], which produces an encryption of the Most Significative Bit (MSB) of $z = 2^\ell + a - b$, because it indicates whether $a < b$ (i.e. $z_\ell \equiv [\![a < b]\!]$).

In terms of complexity, one comparison of two encrypted integers of $\ell = n_i + n_f + 1$ bits requires $7(\ell - 1) + 2$ multiplications over encrypted data, and the exchange of $3(\ell - 2)$ messages of $\ell_N$ bits between the Application and the Authorization Server [48].

### 5.5. arg max (arg min) *Over encrypted data*

The maximum (or minimum) of a vector can be obtained using a binary comparison search tree [49]. At the top level, secure binary comparisons between pairs of elements of the given vector are obtained as follows:

$$\begin{aligned} \max\,([\![a]\!], [\![b]\!]) &= [\![(a < b) \cdot (a - b) + b]\!] \\ &= \left(\pi_{a<b}([\![a]\!], [\![b]\!]) \odot [\![a]\!][\![b]\!]^{-1}\right)[\![b]\!] \end{aligned} \tag{22}$$

At the next level, the number of elements is reduced by half; secure binary comparisons between pairs of the reduced-size vector are performed again. Assuming that the size of the vector $p$ is a power of two (if this is not the case, dummy values have to be added), after $\log_2(p)$ levels, the minimum (or maximum) is obtained.

In order to obtain the index of the maximum value we have used the protocol proposed in [50]. Assuming a vector of $d$ components, the protocol performs $(d - 1)$ encrypted comparisons of $\ell$-bit integers, and $7(d - 1)$ secure multiplications. In terms of round trips, the protocol adds $(d - 1)$ roundtrips to the basic comparison protocol.

## 6. Semiparametric SVM in the encrypted domain

In this section we describe how to encode the high-level SVM algorithm into a sequence of basic operations in the encrypted domain. In particular, we are going to implement the training of a Semiparametric SVM using the PEGASOS algorithm.

### 6.1. *General initialization*

The initial step to all encrypted machine learning algorithms (under the considered scenario) is the generation and distribution of public and secret keys (see Protocol 2).

### 6.2. *Computation of kernel matrix*

Once the keys are distributed among the parties, the Application computes the encrypted kernel matrix $\boldsymbol{K}_{\mathrm{PK}}$. If a semiparametric SVM scheme is considered, a clustering algorithm is applied before in order to obtain a small set of $R$ centroids $\boldsymbol{c}_j$ (see Protocol 3).

### 6.3. *PEGASOS Loop*

And finally, once the encrypted kernel matrix is computed, the Application trains the SVM model using the PEGASOS algorithm (see Protocol 4).

## 7. Performance

We have considered the scenario depicted in Fig. 1 (see Section 3). In particular, we have considered the double trapdoor public-key BCP cryptosystem by Bresson, Catalano and Pointcheval [36] with cipher modulus $N$ of length $\ell_N = 1024$ bits.

We have taken several datasets from the UCI repository [51] to benchmark our private-data trained classifier implementations: Ripley ($m = 250,\ d = 2$), Kwok ($m = 500,\ d = 2$), Twonorm ($m = 5920,\ d = 20$), and Waveform ($m = 4000,\ d = 21$). We have preprocessed the patterns to have zero mean and unit standard deviation, but we have respected the original train/test partition.

```
Protocol: General Initialization
AUTH SERVER does:
  Generate private and public key pairs (PK_i,SK_i).
USERS do:
  Encrypt $\boldsymbol{x}_i$ and $y_i$ using PK_i
  Upload $[\![\boldsymbol{x}_i]\!], [\![y_i]\!]$ to the Storage Server.
APPLICATION does:
  Ask for permission of the Authorization Server to access
  encrypted Users' Data.
  If granted, reencrypt $\{[\![\boldsymbol{x}_i]\!], [\![y_i]\!]\}_{i=1}^{m}$ with $\mathbf{PK} = \prod_{i=1}^{m} \mathrm{PK}_i$
```

**Protocol 2.** General setup protocol.

```
Protocol: Kernel Matrix, $\boldsymbol{K}_{\mathrm{PK}} = \pi_{\mathrm{SVM\ ini}}([\![\boldsymbol{X}]\!], \sigma, R)$
STORAGE SERVER:
  Input: encrypted private data $[\![\boldsymbol{X}]\!]$
APPLICATION:
  Input: plaintext SVM hyperparameter $\sigma$ and $R$
  Output: encrypted Kernel Matrix $\boldsymbol{K}_{\mathrm{PK}}$
1.  Clustering Algorithm
      Obtain centroids $\{[\![\boldsymbol{c}_j]\!]\} = \pi_{\mathrm{FSCL\ or\ k\text{-}Means}}([\![\boldsymbol{X}]\!], R)$
      Compute distances $[\![\boldsymbol{S}(i,j)]\!] = \pi_{\mathrm{DE}}([\![\boldsymbol{x}_i]\!], [\![\boldsymbol{c}_j]\!])$
2.  Kernel Matrix
      Define $\gamma = -\frac{1}{2\sigma^2}$
      Obtain the Kernel Matrix $\boldsymbol{K}_{\mathrm{PK}}[i,j] = [\![k(\boldsymbol{x}_i, \boldsymbol{c}_j)]\!]$
        for $i \in \{1,\ldots,m\}$, $j \in \{1,\ldots,R\}$
          Secure Product: $[\![\gamma]\!] \odot [\![\boldsymbol{S}[i,j]]\!]$
          Secure Bit Shifting (see Section 5.3)
          $\boldsymbol{K}_{\mathrm{PK}}[i,j] = [\![2^{\gamma \boldsymbol{S}(i,j)}]\!] = \pi_{\mathrm{BS}}\left([\![\gamma]\!] \odot [\![\boldsymbol{S}[i,j]]\!]\right)$
```

**Protocol 3.** Secure semiparametric SVM initialization protocol for the computation of the kernel matrix.

### 7.1. Complexity

Using the FSCL as the Clustering algorithm, the computation of the $R$ centroids requires $\mathcal{O}(R \cdot m)$ iterations, resulting $\mathcal{O}(R^2 \cdot m \cdot d)$ outsourced operations. Since the clustering algorithm already includes the evaluation of Euclidean distances between centroids and training instances, the computation of the Kernel Matrix does not significantly contribute to the total computational complexity. Assuming that the accuracy, $\epsilon$, of the SVM training method is directly related to the number of fractional bits, $n_f$, the main PEGASOS loop is repeated a maximum of $\mathcal{O}(2^{n_f})$ times, resulting in a computational complexity of $\mathcal{O}(R \cdot 2^{n_f})$. In contrast, the conventional SMO training algorithm, requires $\mathcal{O}(m^2 \cdot d)$ operations to compute the Full Kernel Matrix, and $\mathcal{O}(m \cdot 2^{n_f})$ iterations, each one of complexity $\mathcal{O}(m^2)$. Just for illustration purposes, for a training set of $m = 1000$ training instances of dimension $d = 10$, and a resolution of $n_f$=16 bits, the encrypted SMO has a computational complexity of $\mathcal{O}(10^{13})$, while the complexity of the semiparametric PEGASOS+FSCL, with $R = 0.2 \cdot m$, is $\mathcal{O}(10^8)$. The complexities of the SMO and PEGASOS SVM training algorithms, and of the k-Means and FSCL clustering methods are summarized in Table 2.

### 7.2. Accuracy and AuC

Since the accuracy of the different methods may vary depending on the selected thresholds, we have decided to compare the

```
Protocol: PEGASOS Loop, $[\![\boldsymbol{w}]\!] = \pi_{\mathrm{PEG}}(\boldsymbol{K}_{\mathrm{PK}}, [\![\boldsymbol{y}]\!])$
STORAGE SERVER:
  Input: encrypted private data $[\![\boldsymbol{y}]\!]$
APPLICATION:
  Input: encrypted Kernel Matrix $\boldsymbol{K}_{\mathrm{PK}}$
         private SVM hyperparameters $C$, $T_{\mathrm{PEG}}$
1. Initialization
     Set $\boldsymbol{w} = \boldsymbol{0}$, $\lambda = 1/(C \cdot m)$
2. for $t = 1 \dots T_{\mathrm{PEG}}$
     $\eta_t = 1/(\lambda t)$
     Choose $i_t \in_R \{1, \dots, m\}$ uniformly at random
     Build $[\![\boldsymbol{r}(\boldsymbol{x}_{i_t})]\!]$
         $[\![\boldsymbol{r}(\boldsymbol{x}_{i_t})]\!][0] = [\![1]\!]$
         $[\![\boldsymbol{r}(\boldsymbol{x}_{i_t})]\!][j] = \boldsymbol{K}_{\mathrm{PK}}[i_t, j]$, $j = 1, \dots, R$.
     Secure Inner Product $g_{\mathrm{PK}} = [\![\boldsymbol{w}]\!]^T \odot [\![\boldsymbol{r}(\boldsymbol{x}_{i_t})]\!]$
     Secure Product $g_{\mathrm{PK}} = [\![y_{i_t}]\!] \odot g_{\mathrm{PK}}$
     Update $[\![\boldsymbol{w}]\!] = [\![(1 - 1/t)]\!] \odot [\![\boldsymbol{w}]\!]$
     Secure Comparison $[\![b]\!] = \pi_{a<b}(g_{\mathrm{PK}}, [\![1]\!])$
     Update $[\![\boldsymbol{w}]\!] = [\![\boldsymbol{w}]\!] \cdot [\![b]\!] \odot ([\![\eta_t]\!] \odot g_{\mathrm{PK}})$
```

**Protocol 4.** Semiparametric PEGASOS loop protocol.

**Table 2**
Complexity of different secure SVMs.

| Complexity | | | |
|---|---|---|---|
| Algorithm | Computation | Storage | Communication |
| SMO | $\mathcal{O}(m^2(m \cdot 2^{n_f} + d))$ | $\mathcal{O}(m^2 \cdot \ell_N)$ | $\mathcal{O}(m^3 \cdot 2^{n_f} \cdot \ell_N)$ |
| PEGASOS | $\mathcal{O}(m(2^{n_f} + m \cdot d)))$ | $\mathcal{O}(m^2 \cdot \ell_N)$ | $\mathcal{O}(m \cdot 2^{n_f} \cdot \ell_N)$ |
| k-Means | $\mathcal{O}(2^{\mathcal{O}(R)} R \cdot m^2 \cdot d^2)$ | $\mathcal{O}(R \cdot m \cdot \ell_N)$ | $\mathcal{O}(2^{\mathcal{O}(R)} \cdot R \cdot m^2 \cdot d^2 \cdot \ell_N)$ |
| FSCL | $\mathcal{O}(R^2 \cdot m \cdot d)$ | $\mathcal{O}(R \cdot m \cdot \ell_N)$ | $\mathcal{O}(R^2 \cdot m \cdot d \cdot \ell_N)$ |
| PEGASOS+FSCL | $\mathcal{O}(R(2^{n_f} + R \cdot m \cdot d))$ | $\mathcal{O}(R \cdot m \cdot \ell_N)$ | $\mathcal{O}(R(2^{n_f} + R \cdot m \cdot d) \cdot \ell_N)$ |

results using a threshold independent measurement, which is the Area Under the Receiver Operation Curve (AuC).

The $C$ and $\gamma$ parameters have been optimized for every algorithm using a grid search, on a set of positive and negative integer powers of 2, nested with a 5-fold cross-validation.

In Fig. 4 we show the AuC values for all four methods under study (namely, SMO, PEGASOS, SP-kmeans, SP-FSCL) and the UCI datasets mentioned above. It can be observed how the performance is similar in all cases, therefore the use of the reduced complexity and privacy preserving versions does not imply a reduction in performance.

Concerning the size of the resulting machines (number of kernel evaluations needed to produce a prediction on a new sample), we have depicted in Fig. 5 the results. SMO and PEGASOS produce much larger machines than the semiparametric versions (roughly, the latter produces machines 10 times smaller than the former). This a very relevant factor, since when operating in the cryptographic scenario, every operation has a non negligible cost.

Under fixed-point arithmetic conditions, there is an special interest in determining the minimum (cleartext) data wordlength. And one of the reasons, apart from simplifying the involved processing hardware, is the reduction of the communication complexity by using data packing [52]. The idea is to take advantage from the fact that in practical situations, the cipher modulus $N$ (typically of 1024 bits) is much larger than the data wordlength ($\ell \ll \log_2(N)$). As a consequence, multiple $\ell$-bit data samples can be packed into a single encrypted message. This way, the amount of data to be transmitted between parties and the number of operations on the encrypted data, and thus the run time, will decrease linearly in relation to the amount of packing allowed.

We have evaluated the performance of the algorithms when they are operated with a reduced number of bits as well as the solution when cleartext is used, for comparison purposes. As it can be observed in Figs. 6–9 most methods produce good results when working with $\ell$ =24 bits ($n_i = 7$, $n_f = 16$ bits), and sometimes with less. Furthermore, the solution with 128 bits is prac-

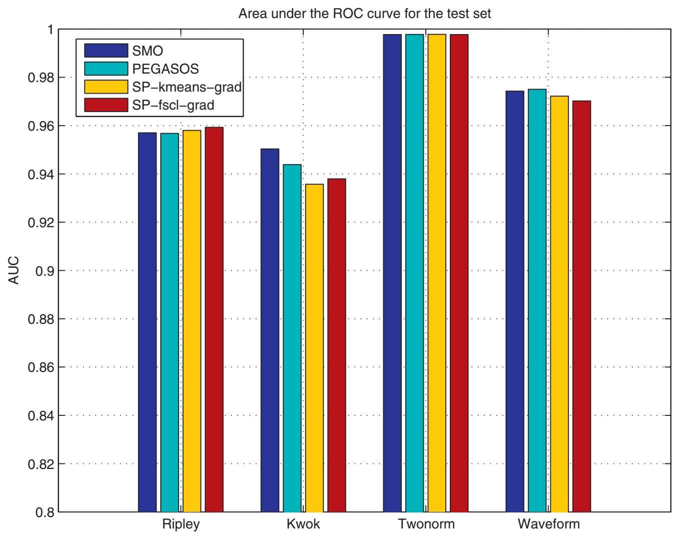

**Fig. 4.** Performance comparison of the implemented methods.

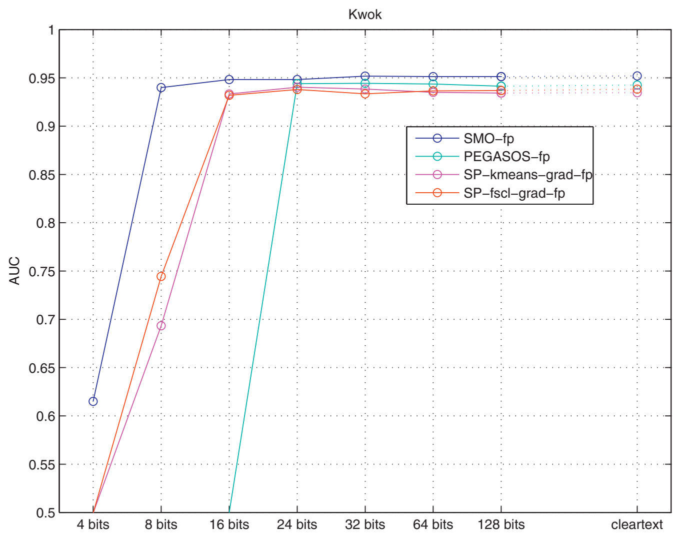

**Fig. 7.** Evolution of performance for an increasing number of bits, for the Kwok case.

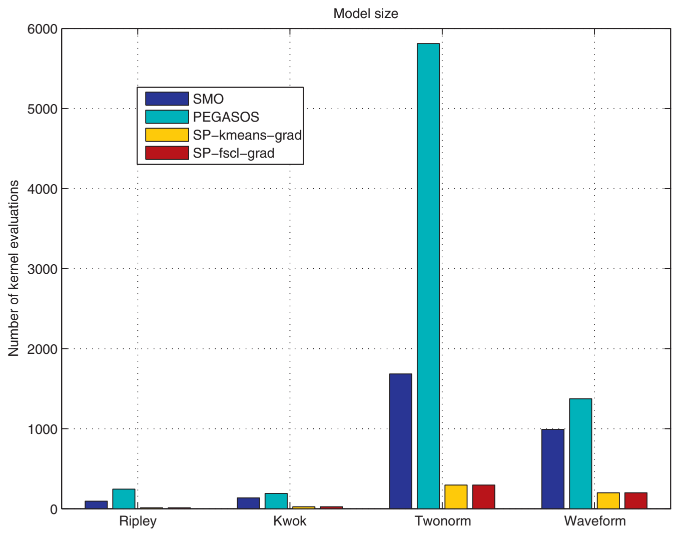

**Fig. 5.** Size comparison of the implemented methods.

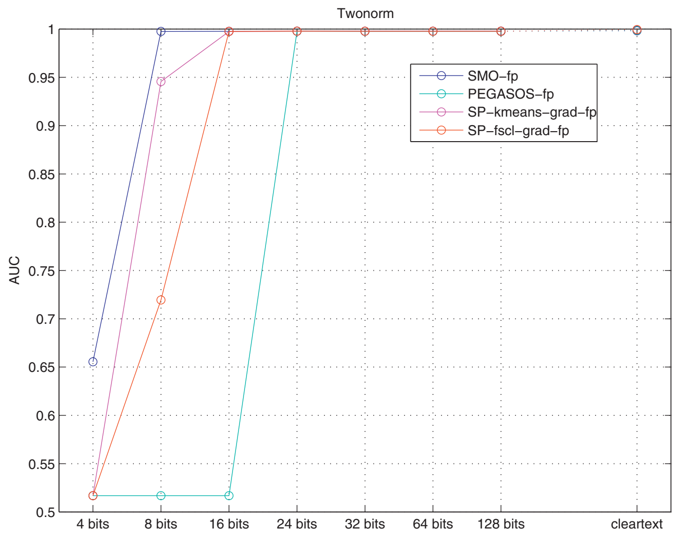

**Fig. 8.** Evolution of performance for an increasing number of bits, for the Twonorm case.

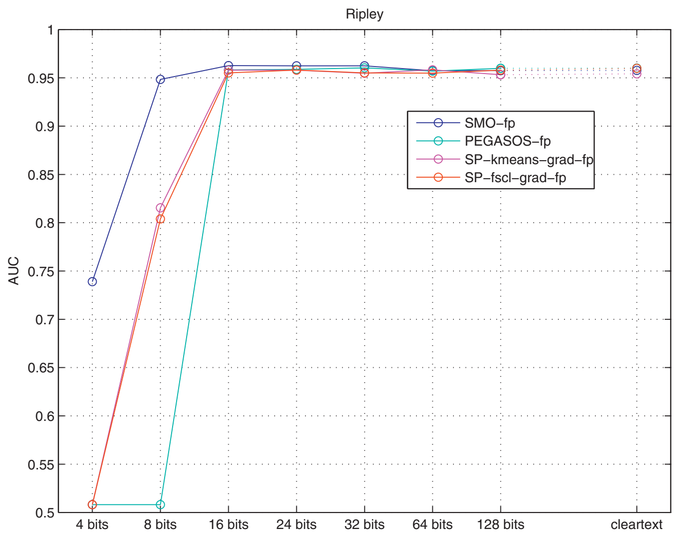

**Fig. 6.** Evolution of performance for an increasing number of bits, for the Ripley case.

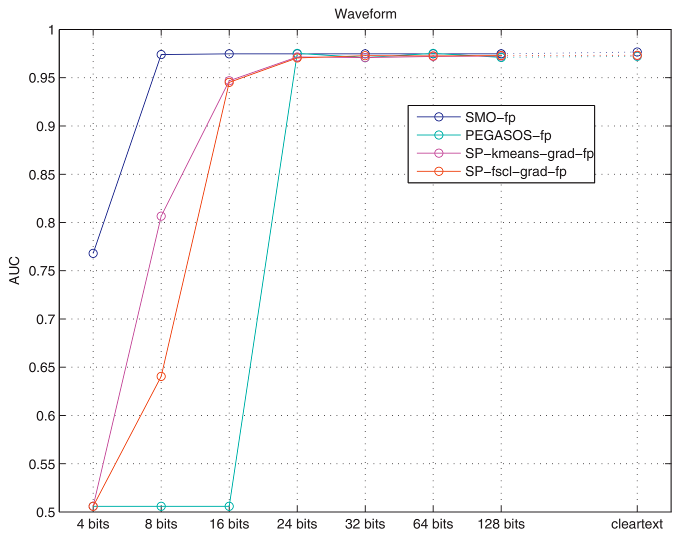

**Fig. 9.** Evolution of performance for an increasing number of bits, for the Waveform case.

tically equivalent to the cleartext solution. The most sensitive algorithm to the number of bits is PEGASOS, which fails to produce results with $\ell = 8$ bits ($n_i = 3,\ n_f = 4$), and sometimes also fails with 16 bits ($n_i = 4,\ n_f = 11$). The most robust algorithm in this sense is SMO, which always produces good results with 8 bits. The proposed Semiparametric methods lay in-between, producing good results in general with 16 bits. In any case, the Semiparametric versions are the unique fully privacy-preserving methods.

## 8. Conclusions

We have presented a scenario for machine learning tasks using privacy-protected data (with multiple encryption keys) training data. The scenario includes an Authorization Server that provides privacy services to Users (Data Owners), authorizes the third-party Applications to access the Users' encrypted data. In our scenario, the Authorization Server also assists the Application in some operations on encrypted data not supported by the cryptosystem. For benchmarking purposes, we have implemented two well known SVM training algorithms (SMO and PEGASOS) in the encrypted domain. We have also proposed a new semiparametric SVM scheme, which includes an unsupervised clustering stage, that avoids the use of *intact* training instances as a part of the SVM model. After analyzing the complexity, in terms of computation, storage, and communication costs, and the impact of finite precision effects on the SVM performance, we conclude that the semiparametric SVM based on the FSCL (in the clustering stage) and the PEGASOS training algorithm is a good candidate for the considered scenario.

## 9. Acknowledgments

This work is partially funded by the Projects CASI-CAM P2013/ICE2845, and CAM P2013ICE-2933 of the Regional Government of Madrid, and TIN2015-70308-REDT, TEC2014-52289-R, TEC2016-80386-P of the Government of Spain. A. Amor-Martín is supported by a FPU predoctoral fellowship of the Government of Spain.

## References

[1] Inspiring new research in the field of signal processing in the encrypted domain, IEEE Signal Process. Mag. (2013), doi:10.1109/MSP.2012.2236771.
[2] R.L. Rivest, L. Adleman, M.L. Dertouzos, On data banks and privacy homomorphisms, Found. Secure Comput. 32 (4) (1978) 169–178.
[3] C. Gentry, Computing arbitrary functions of encrypted data, Commun. ACM 53 (3) (2010) 97–105, doi:10.1145/1666420.1666444.
[4] J. Troncoso-Pastoriza, F. Pérez-González, Secure signal processing in the cloud: enabling technologies for privacy-preserving multimedia cloud processing, Signal Process. Mag. IEEE 30 (2) (2013) 29–41, doi:10.1109/MSP.2012.2228533.
[5] Z. Erkin, T. Veugen, T. Toft, R. Lagendijk, Generating private recommendations efficiently using homomorphic encryption and data packing, Inf. Forensics Secur. IEEE Trans. 7 (3) (2012) 1053–1066, doi:10.1109/TIFS.2012.2190726.
[6] A. Peter, E. Tews, S. Katzenbeisser, Efficiently outsourcing multiparty computation under multiple keys, Inf. Forensics Secur. IEEE Trans. 8 (12) (2013) 2046–2058, doi:10.1109/TIFS.2013.2288131.
[7] C. Cortes, V. Vapnik, Support-vector networks, Mach. Learn. 20 (3) (1995) 273–297.
[8] V.N. Vapnik, Statistical Learning Theory, one ed., Wiley-Interscience, 1998. (September 30, 1998)
[9] D. Anguita, A. Boni, S. Ridella, Learning algorithm for nonlinear support vector machines suited for digital VLSI, Electron. Lett. 35 (16) (1999) 1349–1350, doi:10.1049/el:19990950.
[10] D. Anguita, A. Boni, Digital least squares support vector machines, Neural Process. Lett. 18 (1) (2003) 65–72, doi:10.1023/A:1026249319477.
[11] T.W. Kuan, J.F. Wang, J.C. Wang, P.C. Lin, G.H. Gu, VLSI Design of an SVM learning core on sequential minimal optimization algorithm, IEEE Trans. Very Large Scale Integr. (VLSI) Syst. 20 (4) (2012) 673–683, doi:10.1109/TVLSI.2011.2107533.
[12] S. Shalev-Shwartz, Y. Singer, N. Srebro, A. Cotter, Pegasos: primal estimated sub-gradient solver for SVM, Math. Program. 127 (1) (2011) 3–30.
[13] E. Parrado-Hernández, J. Arenas-García, I. Mora-Jiménez, A. Figueiras-Vidal, A. Navia-Vázquez, Growing support vector classifiers with controlled complexity, Pattern Recognit. 36 (2003) 1479–1488.
[14] A. Navia-Vázquez, Compact multiclass support vector machines, Neurocomputing 71 (2001) 400–405.
[15] A. Navia-Vázquez, F. Pérez-Cruz, A. Artés-Rodríguez, A. Figueiras-Vidal, Weighted least squares training of support vector classifiers leading to compact and adaptive schemes, IEEE Trans. Neural Netw. 12 (2001) 1047–1059.
[16] S.P. Lloyd, Least squares quantization in PCM, Inf. Theory IEEE Trans. 28 (2) (1982) 129–137.
[17] S.C. Ahalt, A.K. Krishnamurthy, P. Chen, D.E. Melton, Competitive learning algorithms for vector quantization, Neural Netw. 3 (3) (1990) 277–290.
[18] R. Agrawal, R. Srikant, Privacy-preserving data mining, ACM SIGMOD Rec. 29 (2) (2000) 439–450, doi:10.1145/335191.335438.
[19] Z. Huang, W. Du, B. Chen, Deriving private information from randomized data, in: Proceedings of the 2005 ACM SIGMOD International Conference on Management of Data, ACM, 2005, pp. 37–48.
[20] A.C. Yao, Protocols for secure computations, in: Foundations of Computer Science, 1982. SFCS '08. 23rd Annual Symposium on, 1982, pp. 160–164, doi:10.1109/SFCS.1982.38.
[21] J. Vaidya, C. Clifton, Privacy-preserving k-Means clustering over vertically partitioned data, in: Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, ACM, 2003, pp. 206–215.
[22] M. Beye, Z. Erkin, R. Lagendijk, Efficient privacy preserving k-Means clustering in a three-party setting, in: Information Forensics and Security (WIFS), 2011 IEEE International Workshop on, 2011, pp. 1–6, doi:10.1109/WIFS.2011.6123148.
[23] J. Vaidya, H. Yu, X. Jiang, Privacy-preserving SVM classification, Knowl. Inf. Syst. 14 (2) (2008) 161–178.
[24] R. Bost, R.A. Popa, S. Tu, S. Goldwasser, Machine learning classification over encrypted data., IACR Cryptol 2014 (2014) 331. ePrint Archive
[25] V. Nikolaenko, U. Weinsberg, S. Ioannidis, M. Joye, D. Boneh, N. Taft, Privacy-preserving ridge regression on hundreds of millions of records, in: Security and Privacy (SP), 2013 IEEE Symposium on, 2013, pp. 334–348, doi:10.1109/SP.2013.30.
[26] R. Lagendijk, Z. Erkin, M. Barni, Encrypted signal processing for privacy protection: conveying the utility of homomorphic encryption and multiparty computation, Signal Process. Mag. IEEE 30 (1) (2013) 82–105, doi:10.1109/MSP.2012.2219653.
[27] D. Anguita, S. Pischiutta, S. Ridella, D. Sterpi, Feed-forward support vector machine without multipliers, Neural Netw. IEEE Trans. 17 (5) (2006) 1328–1331, doi:10.1109/TNN.2006.877537.
[28] B. Lesser, M. Mücke, W.N. Gansterer, Effects of reduced precision on floating–point SVM classification accuracy, Procedia Comput. Sci. 4 (2011) 508–517.
[29] D. Anguita, A. Ghio, S. Pischiutta, S. Ridella, A support vector machine with integer parameters, Neurocomputing 72 (1) (2008) 480–489.
[30] J. Platt, et al., Sequential Minimal Optimization: A Fast Algorithm for Training Support Vector Machines, Technical Report MSR-TR-98-14, Microsoft Research, 1998.
[31] K.-P. Lin, M.-S. Chen, On the design and analysis of the privacy-preserving SVM classifier, Knowl. Data Eng. IEEE Trans. 23 (11) (2011) 1704–1717.
[32] A. Navia-Vázquez, E. Parrado-Hernández, Distributed support vector machines, Neural Netw. IEEE Trans. 17 (4) (2006) 1091–1097.
[33] W. Diffie, M.E. Hellman, New directions in cryptography, Inf. Theory IEEE Trans. 22 (6) (1976) 644–654.
[34] T. Instruments, The TMS320 Family of Digital Signal Processors, 1997. Literature number spra396. http://www.ti.com/lit/an/spra396/spra396.pdf
[35] O. Goldreich, Foundations of Cryptography II, Cambridge University Press, Cambridge, UK, 2004.
[36] E. Bresson, D. Catalano, D. Pointcheval, A simple public-key cryptosystem with a double trapdoor decryption mechanism and its applications, in: Advances in Cryptology-ASIACRYPT 2003, Springer, 2003, pp. 37–54.
[37] D. Hardt, The OAuth 2.0 Authorization Framework, RFC 6749, RFC Editor, 2012. http://www.rfc-editor.org/rfc/rfc6749.txt.
[38] L. Bossuet, M. Grand, L. Gaspar, V. Fischer, G. Gogniat, Architectures of flexible symmetric key crypto engines survey: from hardware coprocessor to multi--crypto-processor system on chip, ACM Comput. Surv. (CSUR) 45 (4) (2013) 41.
[39] K. Sakiyama, L. Batina, B. Preneel, I. Verbauwhede, Multicore curve-based cryptoprocessor with reconfigurable modular arithmetic logic units over GF (2∧ n), IEEE Trans. Comput. (9) (2007) 1269–1282.
[40] S.S. Keerthi, S.K. Shevade, C. Bhattacharyya, K.R.K. Murthy, Improvements to Platt's SMO algorithm for SVM classifier design, Neural Comput. 13 (3) (2001) 637–649.
[41] D. Hush, P. Kelly, C. Scovel, I. Steinwart, QP Algorithms with guaranteed accuracy and run time for support vector machines, J. Mach. Learn. Res. 7 (2006) 733–769.
[42] R. Ostrovsky, Y. Rabani, L.J. Schulman, C. Swamy, The effectiveness of Lloyd–type methods for the k-Means problem, J. ACM (JACM) 59 (6) (2012) 28.
[43] A.S. Galanopoulos, R.L. Moses, S.C. Ahalt, Diffusion approximation of frequency sensitive competitive learning, Neural Netw. IEEE Trans. 8 (5) (1997) 1026–1030.
[44] T.-W. Kwon, C.-S. You, W.-S. Heo, Y.-K. Kang, J.-R. Choi, Two implementation methods of a 1024-bit RSA cryptoprocessor based on modified montgomery algorithm, in: Circuits and Systems, 2001. ISCAS 2001. The 2001 IEEE International Symposium on, 4, IEEE, 2001, pp. 650–653.
[45] O. Catrina, A. Saxena, Secure computation with fixed-point numbers, in: Financial Cryptography and Data Security, Springer, 2010, pp. 35–50.
[46] F. González-Serrano, A. Amor-Martín, J. Casamayón-Antón, State estimation using an extended Kalman filter with privacy-protected observed inputs, in: GlobalSIP14-Workshop on Information Forensics and Security 2014. Proceedings of the, 2014, pp. 1647–1652.
[47] Z. Erkin, M. Franz, J. Guajardo, S. Katzenbeisser, I. Lagendijk, T. Toft, Privacy-p

reserving face recognition, in: I. Goldberg, M. Atallah (Eds.), 9th International Symposium, PETS 2009, volume 5672 of Privacy Enhancing Technologies, Seattle, WA, USA, 2009, pp. 235–253.

[48] T. Veugen, Comparing Encrypted Data, Technical Report, Multimedia Signal Processing Group, Delft University of Technology, The Netherlands, and TNO Information and Communication Technology, Delft, The Netherlands, 2011.

[49] Z. Erkin, T. Veugen, T. Toft, R. Lagendijk, Privacy-preserving user clustering in a social network, in: First IEEE International Workshop on Information Forensics and Security, WIFS 2009, London, 2009, pp. 96–100, doi:10.1109/WIFS.2009.5386476.

[50] R. Bost, R.A. Popa, S. Tu, S. Goldwasser, Machine learning classification over encrypted data, in: Proceedings of 2015 Network and Distributed System Security (NDSS) Symposium, 2015.

[51] K. Bache, M. Lichman, UCI Machine Learning Repository, 2013. http://archive.ics.uci.edu/ml.

[52] T. Bianchi, A. Piva, M. Barni, Efficient linear filtering of encrypted signals via composite representation, in: Digital Signal Processing, 2009 16th International Conference on, 2009, pp. 1–6, doi:10.1109/ICDSP.2009.5201116.

**Francisco J. González-Serrano** received the Ph.D. degree in telecommunications engineering from the Universidad de Vigo, Vigo, Spain, in 1997. Since 2000, he has been a Professor with the Department of Signal Theory and Communications, Universidad Carlos III de Madrid, Madrid, Spain. His current research interests include wireless sensor networks (particularly localization and tracking tasks) and the application of signal processing to communication systems.

**Ángel Navia-Vázquez** received the Degree in telecommunications engineering in 1992 from Universidad de Vigo, Spain, and the Ph.D. degree, also in telecommunications engineering in 1997 from the Universidad Politécnica de Madrid, Spain. He is now an Associate Professor at the Department of Signal Theory and Communications, Universidad Carlos III de Madrid, Spain. His research interests are focused on new architectures and algorithms for nonlinear/adaptive/distributed processing, as well as their application to multimedia, communications, signal processing and data analysis. He has (co)authored about 30 international refereed journal papers in these areas, several book chapters, more than 50 conference communications, and participated in more than 20 research projects. Prof. Navia-Vázquez has been an Associate Editor of IEEE Transactions on Neural Networks (Jan 2004 to Dec 2008).

**Adrián Amor-Martín** received the Telecommunications Engineering and the Research Masters Degree in Multimedia and Communications from the Universidad Carlos III de Madrid, Spain, in 2012 and 2014 respectively, where he is currently pursuing the Ph.D. degree in communications. His research interests are focused on the application of numerical methods to high-performance computational electromagnetics including finite elements, and hp adaptivity. Mr. Amor Martín received the best Master Thesis Disser- tation Award from the COIT/AEIT (Official Association of Spanish Telecom- munication Engineers) in 2012.