Secure Computation of Inner Product of Vectors with Distributed Entries & its Applications to SVM

No Institute Given

Abstract. Nowadays organizations and individuals outsource computation and storage to cloud. This poses a threat to the privacy of users. Different users encrypt their private data with (possibly) different keys to prevent any kind of outside attack on their privacy. In this outsourced model of computation where the data owners have already encrypted and uploaded private data, to enable the users for collaborative data mining a scheme is needed that can process encrypted data under multiple keys. Privacy preserving inner product computation is an essential tool on which many data mining algorithms are based. There are few works which address this outsourced privacy preserving inner product computation but none of them deals with the scenario when the entire database is arbitrarily partitioned among the users. We propose two outsourced privacy preserving protocols for computation of inner product of vectors when the underlying database is arbitrarily partitioned. We provide an SVM training model that preserves the privacy of the user's data-vectors. Our scheme is based on an integer vector encryption scheme.

Keywords: Privacy preserving, support vector machine, inner product, homomorphic encryption.

1 Introduction

Machine learning tasks are typically classified into two broad categories- *supervised* learning and *unsupervised* learning. In the former, machine is given example inputs and corresponding outputs by a "supervisor" and the goal of the machine is to *learn* a general rule that maps inputs to outputs. In unsupervised learning, the existence of supervisor is not assumed. *Classification* is one of the most important tasks found in today's world. The goal of classification is to divide the input space into two or more classes and the learning algorithm must produce a model that will be able to map unseen/new inputs to one of these classes. For example, looking at the blood pressure, blood sugar etc. of patients and their history of heart-disease, a doctor may want to classify patients vulnerable to heart attacks.

A *support vector machine* (SVM) [4] is one of the most commonly used classifiers that divides its input space into two regions, separated by a hyperplane.

One side is known as the side of positive class and the other side as negative class. When a new input comes then the classifier has to decide, based on its training, which side of the hyperplane does this new input belong to and classifies it accordingly as positive or negative. To build an SVM, the basic building block is the computation of inner products (a.k.a dot products) of the data vectors (i.e. input vectors). These inner products of the available data vectors help to construct *kernel matrix*. Based on this kernel matrix, a minimization problem is solved to obtain the required hyperplane. More discussion on SVM follows in Section 2.3.

In standard setting, it is assumed that while building the classification algorithms, one has direct access to all available data points. However, in certain areas e.g. medical records, the data points encode sensitive information owned by different entities or organizations. To build a strong classification algorithm, it is beneficial to train the model on a large, complete data set which may not be readily available in the clear. In a sense, this requires distributed knowledge discovery without disclosure of data. Thus began the journey of privacy-preserving computation on distributed private data. The two pioneering works [2, 7] paved the path for future research in this direction.

Broadly speaking, there are two models of collaborative computation in a privacy-preserving manner - (*i*) the *distributed* model of computation and (*ii*) the *outsourced* model of computation. In the distributed model of computation, data owners hold their private data with themselves and may take help of a third party to build a classification model/classifier without revealing thei private data to anyone. This third party is however limited in its capacity for storing data or bearing heavy overheads of computation. On the other hand, in the outsourced model of computation, data owners do not hold their private data. They have already uploaded (i.e. outsourced) their data (in an encoded form) to a cloud and with their permission the cloud initiates a classification algorithm on the stored data and learns the model. In this model the cloud provides the storage space and bears the heavy burden of computation, whereas the data owners can be very limited in terms of computational power.

In the present day the outsourced model is very relevant as organizations are finding it extremely difficult to store huge amount of personal data. More and more organizations are becoming motivated to outsource data to a cloud system. However, while enjoying the benefits of outsourced storage and management, organizations face the risk of private information disclosure. Once, the private data is outsourced to the cloud, it is no longer under the control of the owner(s). To prevent cloud from any wrong doing, data owners generally encrypt their sensitive private data before sending them to the cloud. This motivates the problem of how the cloud executes data mining process on encrypted data if need be.

As mentioned earlier, the main building block for SVM is the computation of the inner products of the available data vectors. If the inner products can be computed in a privacy preserving manner then an SVM can be modeled securely based on that.

Some private inner product protocols were proposed in the literature of distributed computation model [5, 14]. Later, Goethals et al.[6] demonstrated attacks on these protocols and proposed a generic secure inner product protocol based on homomorphic encryption. Vaidya et al. [15] and Yu et al. [18, 19] provided privacy preserving inner product computation protocols and constructed SVM for horizontally and vertically partitioned data sets.

In the outsourced model of computation, Liu et al. [8] first proposed the construction using BGN cryptosystem [3] which supports homomorphic additions and one homomorphic multiplication under a single secret key to compute the inner product. Lopez et al. [11] proposed a fully homomorphic encryption scheme for multiple secret keys. However, using FHE is not efficient for practical purpose. Several research work have been carried out which support multiple secret keys [13, 16]. However, these works assume the existence of non colluding servers and are applicable to construct linear mean classifier.

Recently, Zhang et al. [20] proposed secure inner product protocol and its application to SVM for horizontally and vertically partitioned data. They used an Integer Vector encryption scheme proposed by Zhou and Wornell [22]. Zheng et al.'s construction supports outsourced model of computation with multiple keys and does not require existence of two non colluding servers. However, they have given constructions of SVM for horizontally and vertically partitioned database.

Consider a scenario where two laboratories can perform same set of expensive medical tests. No patient wants to take the same test twice. But it may very well happen that at one point of time, a patient took some medical tests with one laboratory and later, took the rest of medical tests with another laboratory. Varying over different patients, we see that the database of the test results of the patients is arbitrarily partitioned between the two laboratories. Due to privacy agreement, the laboratories cannot disclose any information to one another. On the other hand, they store their encrypted data to the same cloud system. Later, the cloud system may help the laboratories for data mining from database. An additional problem, in this scenario is that the laboratories may use different keys to encrypt their data before uploading the encrypted data to the cloud.

A recent work [12] considered privacy preserving analytics on arbitrarily partitioned data. But, they have build their scheme on secure gradient descent method and left open the problem of privacy preserving SVM classifier.

1.1 Our Contribution

We propose a method for secure inner product computation of vectors whose entries are arbitrarily distributed between the parties. We propose a solution on the basis of Integer Vector Encryption. We first build a secure protocol based on Zhang et al.'s construction and later propose a modified algorithm for SVM training for arbitrarily partitioned database. The modification in the algorithm reduces the communication rounds between the parties and the cloud. There is only one round of key-agreement between the parties. Our protocol is very efficient and supports outsourced model of computation. We achieve sufficient

security when there is only one server and our scheme is fully secure if the existence of non-colluding servers is assumed.

2 Preliminaries

To start with we recall some basic terminologies and building blocks and give a short overview of system model and the threat model.

2.1 System Model

We consider the outsourced model of computation where there are multiple users and one cloud. Different users encrypt their private data using different keys and then upload the encrypted data to the cloud. The cloud stores and manages the data. It runs data mining algorithms on the encrypted data when requested by the users and computes the encrypted classifier.

2.2 Threat Model

We assume that the cloud and the users are honest but curious. They will follow the protocol honestly but will try to infer sensitive information. We further assume that there is no collusion between the parties and the cloud. More details is given in Section 4.

2.3 Support Vector Machine

In a binary classification problem (also called *two class* problem) the aim is to classify an unknown "feature" vector ($\boldsymbol{x}$) as belonging to which class of the two. Given a training data set $(\boldsymbol{x}_i, y_i)$ for $i = 1, \ldots, n$ where $\boldsymbol{x}_i \in \mathbb{R}^m$ are the training samples and $y_i \in \{-1, +1\}$ are the corresponding *labels*, a Support vector machine (SVM) finds a best separating hyperplane ($\boldsymbol{w}^T\boldsymbol{x} - b = 0$) somewhere in the middle so that the distance of the positive samples from the plane and the negative samples from the plane are kind of maximized. $\boldsymbol{w} \in \mathbb{R}^m$ is called the *weight* vector and b is called the *bias* item. $\frac{1}{||\boldsymbol{w}||}$ is the *margin*. The standard SVM maximizes the margin while minimizing the error:

$$min_{\boldsymbol{w},\gamma_i} \ \frac{1}{2}\boldsymbol{w}^T\boldsymbol{w} \ + \ C\sum_{i=1}^{n}\gamma_i$$

such that $y_i(\boldsymbol{w}^T\boldsymbol{x_i} - b_i) \geq 1 - \gamma_i$ and $\gamma_i \geq 0$ for all i.

Here C balances the margin size and the error and γ_is are slack variables that allow error. These slack variables are used to cope with the noise. The above *primal* form of SVM is often solved in its *dual* form:

$$min_{\boldsymbol{\alpha}} \ \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}\boldsymbol{\alpha}^T y_i y_j K(\boldsymbol{x_i}, \boldsymbol{x_j})\boldsymbol{\alpha} \ - \ \sum_{i=1}^{n}\alpha_i$$

such that $0 \leq \alpha_i \leq C$ for all i and $\sum_{i=1}^{n}\alpha_i y_i = 0$.

Here $\boldsymbol{\alpha} \in \mathbb{R}^n$ where n denotes the number of training samples and $K(\boldsymbol{x_i}, \boldsymbol{x_j})$ denotes the (i,j)th entry of the so called *Kernel* matrix $K \in \mathbb{R}^{n \times n}$. Kernel matrix contains the "information" for every pair of training samples. For *linear* SVM, $K(\boldsymbol{x_i}, \boldsymbol{x_j}) = \boldsymbol{x_i}.\boldsymbol{x_j}$ the inner/scalar product of the vectors. The α_is are solved from the dual problem and those $\boldsymbol{x}_i$s are *support* vectors for which α_i are *strictly* greater than zero. Once the support vectors are found, the weight vector can be computed: $\boldsymbol{w} = \sum_{\alpha_i > 0} \alpha_i y_i \boldsymbol{x}_i$. From this we can find the bias item b by solving $\boldsymbol{w}^T \boldsymbol{x}_i - y_i$ for any randomly chosen support vector sample $\boldsymbol{x}_i$ and its class label y_i. To get a moderate value of b, we can find b_i for every support vector $\boldsymbol{x_i}$ and then average them out.

2.4 Cryptographic Building Blocks

In this section we discuss some building blocks that will be needed in due course.

Integer Vector Encryption Our protocols rely on the integer vector (homomorphic) encryption scheme of Zhou and Wornell [22]. For the sake of completeness we give a brief description of the scheme and its useful properties. Integer vector encryption encrypts a vector as a whole with a single key, rather than encrypting every element of the vector one by one.

- **Encryption:** Let $\boldsymbol{x} \in \mathbb{Z}_N^m$ be a plaintext vector, where N denotes the alphabet size, m denotes the vector-length. Let $S \in \mathbb{Z}^{m \times n}$ be the secret-key. Let t be a large integer and $\boldsymbol{e} \in \mathbb{Z}^m$ be an *error* vector such that $||\boldsymbol{e}||_\infty < \frac{t}{2}$. The ciphertext is a vector $\mathcal{E}(\boldsymbol{x}) = \boldsymbol{c} \in \mathbb{Z}^n$ which satisfies: $S\boldsymbol{c} = t\boldsymbol{x} + \boldsymbol{e}$.
- **Decryption:** With the help of secret key S, given a ciphertext vector $\boldsymbol{c}$ the decryption algorithm works as $\mathcal{D}(\boldsymbol{c}) = \lceil \frac{S\boldsymbol{c}}{t} \rfloor$.

Most important properties of the cryptosystem are:

- **Key-switching:** From a ciphertext vector $\boldsymbol{c}$ under the private-key S it is possible to convert it to another ciphertext $\boldsymbol{c}_1 \in \mathbb{Z}^{n'}$ under a *different* key $S_1 \in \mathbb{Z}^{m \times n'}$ such that $S\boldsymbol{c} = S_1\boldsymbol{c}_1$.
- **Addition:** Given two ciphertext vectors $\boldsymbol{c}_1$ and $\boldsymbol{c}_2$ corresponding to two plaintext vectors $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$ under the same key S, the addition $\boldsymbol{c}_1 + \boldsymbol{c}_2$ corresponds to the plaintext vector $\boldsymbol{x}_1 + \boldsymbol{x}_2$. It is to be noted that if the two plaintext vectors are encrypted using two different keys S_1 and S_2 then using the *key-switching* technique the ciphertext vectors can be converted to ciphertexts under a common key S. Now they can be added.

$$S(\boldsymbol{c}_1 + \boldsymbol{c}_2) = t(\boldsymbol{x}_1 + \boldsymbol{x}_2) + (\boldsymbol{e}_1 + \boldsymbol{e}_2) \tag{1}$$

- **Inner Product:** Given a pair $(\boldsymbol{x}_1, \boldsymbol{c}_1)$ under key S_1 and another pair $(\boldsymbol{x}_2, \boldsymbol{c}_2)$ under S_2,

$$S_1\boldsymbol{c}_1 = t\boldsymbol{x}_1 + \boldsymbol{e}_1; S_2\boldsymbol{c}_2 = t\boldsymbol{x}_2 + \boldsymbol{e}_2 \tag{2}$$

the encrypted inner product of plaintext vectors can be obtained as $\mathcal{E}(\boldsymbol{x}_1^T.\boldsymbol{x}_2) = \lceil \frac{vec(\boldsymbol{c}_1\boldsymbol{c}_2^T)}{t} \rfloor$. The secret key which decrypts this cipertext is $vec(S_1^T S_2)^T$.

3 Protocol for secure inner product computation on outsourced vectors

Suppose there are two m-dimensional vectors $\boldsymbol{x} = (x_1, x_2, \ldots, x_m)$ and $\boldsymbol{y} = (y_1, y_2, \ldots, y_m)$. Our target is to find the scalar product $\boldsymbol{x}.\boldsymbol{y} = \sum_{i=1}^{m} x_i y_i$. Also suppose party A owns the entries $(x_i)_{i\in I}$ and $(y_j)_{j\in J}$ where $I, J \subset [1, m]$. Party B owns the rest namely, $(x_i)_{i\in I'}$ and $(y_j)_{j\in J'}$ where I' and J' are complements of I and J respectively. We will denote a missing entry by $*$. More specifically, if the ith entry is missing then we will denote it by $*_i$.

3.1 A non-private solution for scalar product computation on outsourced vectors

We consider the model of where the encrypted data are stored in the Cloud by both the parties and the cloud is responsible for majority part of the storage and computation.
Main question for the parties as to how to upload their vectors in the cloud so that it facilitates the inner product computation on the encrypted vectors. Since the parties upload their vectors in the cloud therefore one natural privacy issue for an individual party is he does not want the cloud to know which entries he is missing. One possible non-private solution can be formulated as follows.
A first finds the intersections $I \cap J$ and $I' \cap J'$. Suppose $I \cap J = \{i_1, i_2, \ldots, i_s\}$ and $I' \cap J' = \{j_1, j_2, \ldots, j_t\}$. Thus he sees (after a suitable rearrangement) the following matrix where the first row contains the entries corresponding to $\boldsymbol{x}$ and the second row contains the entries corresponding to $\boldsymbol{y}$:

$$\begin{bmatrix} \boldsymbol{x}_A \\ \boldsymbol{y}_A \end{bmatrix} = \left[\begin{array}{c|c|c} x_{i_1} x_{i_2} .. x_{i_s} & *_{j_1} *_{j_2} .. *_{j_t} & x_\alpha \ \ldots \ *_\beta \\ y_{i_1} y_{i_2} .. y_{i_s} & *_{j_1} *_{j_2} .. *_{j_t} & *_\alpha \ \ldots \ y_\beta \end{array}\right].$$

The first block of the matrix corresponds to the indices $I \cap J$, second block corresponds to $I' \cap J'$. The third block corresponds to the rest of the entries where if A holds x_α then the corresponding y_α is held by B and vice-versa.
Hence B observes the "complement" of the above matrix:

$$\begin{bmatrix} \boldsymbol{x}_B \\ \boldsymbol{y}_B \end{bmatrix} = \left[\begin{array}{c|c|c} *_{i_1} *_{i_2} .. *_{i_s} & x_{j_1} x_{j_2} .. x_{j_t} & *_\alpha \ \ldots \ x_\beta \\ *_{i_1} *_{i_2} .. *_{i_s} & y_{j_1} y_{j_2} .. y_{j_t} & y_\alpha \ \ldots \ *_\beta \end{array}\right].$$

In this scenario, both the parties replace their missing entries $*$ by 0 and upload the modified vectors in the cloud. We denote the modified vectors by $\boldsymbol{x}_A^*$, $\boldsymbol{y}_A^*$, $\boldsymbol{y}_B^*$, $\boldsymbol{x}_B^*$. So, the parties A and B respectively has the following modified vectors:

$$\begin{bmatrix} \boldsymbol{x}_A^* \\ \boldsymbol{y}_A^* \end{bmatrix} = \left[\begin{array}{c|c|c} x_{i_1} x_{i_2} .. x_{i_s} & 0_{j_1} 0_{j_2} .. 0_{j_t} & x_\alpha \ ... \ 0_\beta \\ y_{i_1} y_{i_2} .. y_{i_s} & 0_{j_1} 0_{j_2} .. 0_{j_t} & 0_\alpha \ ... \ y_\beta \end{array}\right]$$

$$\begin{bmatrix} \boldsymbol{x}_B^* \\ \boldsymbol{y}_B^* \end{bmatrix} = \left[\begin{array}{c|c|c} 0_{i_1} 0_{i_2} .. 0_{i_s} & x_{j_1} x_{j_2} .. x_{j_t} & 0_\alpha \ ... \ x_\beta \\ 0_{i_1} 0_{i_2} .. 0_{i_s} & y_{j_1} y_{j_2} .. y_{j_t} & y_\alpha \ ... \ 0_\beta \end{array}\right].$$

Once these are uploaded to the cloud, the cloud computes $\boldsymbol{x}_A^*.\boldsymbol{y}_A^* + \boldsymbol{x}_A^*.\boldsymbol{y}_B^* + \boldsymbol{y}_A^*.\boldsymbol{x}_B^* + \boldsymbol{x}_B^*.\boldsymbol{y}_B^*$ and outputs the value. The correctness of the protocol can easily be verified.

3.2 Privacy preserving computation

In this section we give a privacy preserving protocol for the secure dot product computation. We assume that the original data vectors consist of non-zero entries so that when a party downloads some data vectors from the cloud and decrypt it to see some zero entries, he can readily conclude that the corresponding entries were missing. This assumption although restrictive, is quite realistic. For example, in medical domain most test results have non zero values.

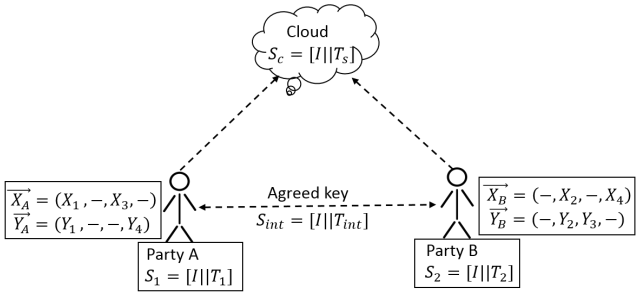


Fig. 1: Private inputs of the parties

Party A randomly chooses a secret key $S_1 = [I||T_1]$, the concatenation of the identity matrix of size $m \times m$ and a "*thin*" (possibly a column) matrix T_1. A encrypts his vectors $\boldsymbol{x}_A^*$ and $\boldsymbol{y}_A^*$ with the help of the secret matrix S_1 to output ciphertext vectors $\boldsymbol{c}_{x,A}$ and $\boldsymbol{c}_{y,A}$ and uploads to the *cloud*. Similarly, B chooses a secret matrix $S_2 = [I||T_2]$, outputs ciphertext vectors $\boldsymbol{c}_{x,B}$ and $\boldsymbol{c}_{y,B}$ and uploads to the cloud (*see* Fig. 1 & Fig. 2).

Recall that the cloud needs to compute $\boldsymbol{x}_A^*.\boldsymbol{y}_A^* + \boldsymbol{x}_A^*.\boldsymbol{y}_B^* + \boldsymbol{y}_A^*.\boldsymbol{x}_B^* + \boldsymbol{x}_B^*.\boldsymbol{y}_B^*$ in order to get the inner product of $\boldsymbol{x}$ and $\boldsymbol{y}$. So there are inner product of vectors from the same user as well as inner product of vectors from different users. To compute the inner products of the plaintext vectors from the ciphertext vectors the cloud requires corresponding secret keys. For example, the encryption of $\boldsymbol{x}_A^*.\boldsymbol{y}_A^*$ is obtained from $\lceil \frac{vec(\boldsymbol{c}_{x,A} \ \boldsymbol{c}_{y,A}^T)}{t} \rfloor$ and the corresponding secret key is $vec(S_1^T S_1)^T$. Similarly, the other secret keys are $vec(S_1^T S_2)^T$ (for encrypted inner

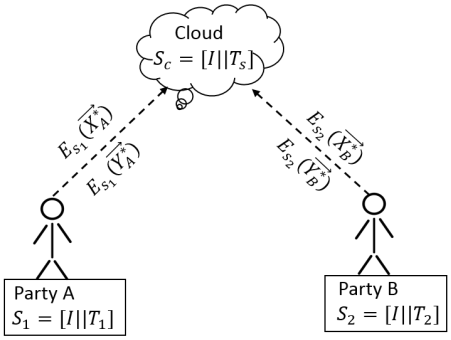


Fig. 2: Outsourcing encrypted data to Cloud

products of vectors from both users A and B) and $vec(S_2^T S_2)^T$ (for encrypted inner product of vectors from B).

In order to transform the encryption of $\boldsymbol{x}_A^*.\boldsymbol{y}_A^*$ from $vec(S_1^T S_1)^T$ to secret key $[I||T_s]$ of the cloud, A computes the *key-switching* matrix M_1 as follows:

$$\begin{bmatrix} (vec(S_1^T S_1)^T)^* - T_s A_a + E_a \\ A_a \end{bmatrix},$$

where A_a is a random matrix and E_a is a random noise matrix chosen by A.

Similarly, for computing the encrypted inner product $\boldsymbol{x}_B^*.\boldsymbol{y}_B^*$, B computes the *key-switching* matrix M_2 as

$$\begin{bmatrix} (vec(S_2^T S_2)^T)^* - T_s A_b + E_b \\ A_b \end{bmatrix},$$

where A_b is a random matrix and E_b is a random noise matrix chosen by B.

However, the secret key corresponding to the encrypted inner products of the vectors from the users A and B is given by $vec(S_1^T S_2)^T$ and hence it requires a joint computation. From the form of S_1 and S_2 it follows that:

$$S_1^T S_2 = \begin{bmatrix} I & T_2 \\ T_1^T & T_1^T T_2 \end{bmatrix}$$

$$= \begin{bmatrix} I & \mathbf{0} \\ T_1^T & \mathbf{0} \end{bmatrix} + \begin{bmatrix} \mathbf{0} & T_2 \\ \mathbf{0} & \mathbf{0} \end{bmatrix} + \begin{bmatrix} \mathbf{0} & \mathbf{0} \\ \mathbf{0} & T_1^T T_2 \end{bmatrix}.$$

A now generates a random invertible matrix P_1, computes $P_1 T_1^T$ and sends it to B. B then generates a random invertible matrix P_2 and sends $P_1 T_1^T T_2 P_2$ to S. A and B separately send P_1^{-1} and P_2^{-1} to S who then recovers $T_1^T T_2$. Therefore at this point of time A, B and S separately hold $\begin{bmatrix} I & \mathbf{0} \\ T_1^T & \mathbf{0} \end{bmatrix}$, $\begin{bmatrix} \mathbf{0} & T_2 \\ \mathbf{0} & \mathbf{0} \end{bmatrix}$ and $\begin{bmatrix} \mathbf{0} & \mathbf{0} \\ \mathbf{0} & T_1^T T_2 \end{bmatrix}$. They can now compute corresponding key-switching matrices M_A,

M_B and M_S separately. Notice that the ultimate key-switching matrices will be the sum $M_A + M_B + M_S = M_{final}$, say. Once these three final key-switching matrices viz. M_1, M_2 and M_{final} are obtained, S can transform the encrypted inner product $\boldsymbol{x}_A^*.\boldsymbol{y}_A^* + \boldsymbol{x}_A^*.\boldsymbol{y}_B^* + \boldsymbol{y}_A^*.\boldsymbol{x}_B^* + \boldsymbol{x}_B^*.\boldsymbol{y}_B^*$ into an encryption with respect to his private key $[I||T_s]$. With the help of his secret key S can now construct the Kernel matrix and run the SVM algorithm.

We now describe an SVM training algorithm and classification protocol in the following Algorithm 1 and Algorithm 2.

4 Security Analysis

We first describe our *adversarial* model.

Adversarial Model: We assume that A, B and the *cloud* all are semi-honest (also known as honest-but-curious). Each of them follows the protocols correctly but tries to gather or infer extra information than they are supposed to know. We also assume that none of the parties and the cloud collude with each other.

In order to carry out the security analysis, first we prove that the "key-switching" technique is secure. That is, the *cloud* cannot deduce the private keys of the parties from the key-switching matrices. To prove the result we need the following hardness assumption of *learning with errors* (LWE) problem.

Learning with Errors problem: Given arbitrary many samples $(\boldsymbol{a}_i, b_i) \in \mathbb{Z}_q^m \times \mathbb{Z}_q$, it is infeasible to recover (with non-negligible probability) $\boldsymbol{v} \in \mathbb{Z}_q^m$ from $b_i = \boldsymbol{v}.\boldsymbol{a}_i + \epsilon_i$, where ϵ_i denotes random noise chosen suitably from a distribution over $\mathbb{Z}_q$.

The above hardness problem essentially says that it is very hard to find a solution to a system of "approximate" linear equations even if arbitrarily many such equations are provided.

Theorem 1. *Let S denote the private-key (matrix), S^* its intermediate representation, M the key-switching matrix from S to S'. Thus, $S'M = S^* + E$ where E is a random noise matrix.*
It is infeasible for the cloud to recover, with non-negligible probability, S^* although it has access to both S' and M.

Proof. Choose a prime modulus $q >> max.\{||S||_\infty, ||M||_\infty, ||S'||_\infty, ||E||_\infty\}$ so that every element in $\mathbb{Z}$ for the protocol can be treated as elements in $\mathbb{Z}_q$. When the cloud knows S' and M, it can compute the ith row of $S'M$. Let $\boldsymbol{b}_i$ denote the ith row vector of $S'M$. Then $\boldsymbol{b}_i = \boldsymbol{v} + \boldsymbol{\epsilon}_i$, where $\boldsymbol{v}$, $\boldsymbol{\epsilon}_i$ respectively denote the ith rows of S^*, E. Hence, the cloud has a set of approximate linear equations $b_{ij} = v_j + \epsilon_{ij}$ modulo the prime modulus q. It is now not very hard to see that if the cloud can solve this set of approximate linear equations then it essentially solves the LWE problem. □

We now have the following theorem which follows from Theorem 1.

Algorithm 1 SVM Training

1: **procedure** KEY-AGREEMENT
2: A, B secretly agree on an internal private key $S_{int} = [I||T_{int}]$ completely hidden from the cloud.
3: **procedure** KEY-SWITCHING
4: A uploads key-switching matrix $M_1 = \begin{bmatrix} (vec(S_1^T S_1)^T)^* - T_s A_a + E_a \\ A_a \end{bmatrix}$.
5: B uploads key-switching matrix $M_2 = \begin{bmatrix} (vec(S_2^T S_2)^T)^* - T_s A_b + E_b \\ A_b \end{bmatrix}$.
6: $A, B, Cloud$ jointly compute $M_{final} = \begin{bmatrix} (vec(S_1^T S_2)^T)^* - T_s A_{ab} + E_{ab} \\ A_{ab} \end{bmatrix}$.
7: **procedure** COMPUTATION OF ENCRYPTED DOT PRODUCT BY THE CLOUD
8: Using M_1, the *cloud* transforms the encryption of $\boldsymbol{x}_A^*.\boldsymbol{y}_A^*$ to an encryption with underlying secret key $S_c = [I||T_s]$.
9: Using M_2, the *cloud* transforms the encryption of $\boldsymbol{x}_B^*.\boldsymbol{y}_B^*$ to an encryption with underlying secret key $S_c = [I||T_s]$.
10: Using M_{final}, the *cloud* transforms encryptions of $\boldsymbol{x}_A^*.\boldsymbol{y}_B^*$ and $\boldsymbol{x}_B^*.\boldsymbol{y}_A^*$ to encryptions with underlying secret key $S_c = [I||T_s]$.
11: Adding these the *cloud* gets the encryption of $\boldsymbol{x}.\boldsymbol{y}$ under the secret key S_c.
12: **procedure** COMPUTATION OF GRAM MATRIX BY THE CLOUD
13: Repeating steps $6-9$ the *cloud* can compute the encrypted Gram matrix $\mathcal{E}_{S_c}(K)$ which can be decrypted by using S_c.
14: Using K, the *cloud* can now run the SVM algorithm to obtain α_is which are greater than zero. Suppose, without loss of generality, first k coefficients $\alpha_1, \ldots, \alpha_k > 0$.
15: **procedure** COMPUTATION OF ENCRYPTED WEIGHT-VECTOR BY THE CLOUD
16: Cloud computes $y_1\alpha_1 \boldsymbol{c}_{1,A}^* + \cdots + y_k\alpha_k \boldsymbol{c}_{k,A}^* = \mathcal{E}_{S_1}(\boldsymbol{w}_A)$.
17: Cloud computes $y_1\alpha_1 \boldsymbol{c}_{1,B}^* + \cdots + y_k\alpha_k \boldsymbol{c}_{k,B}^* = \mathcal{E}_{S_2}(\boldsymbol{w}_B)$.
18: A and B generate key-switching matrices separately to transform the secret keys of $\mathcal{E}_{S_1}(\boldsymbol{w}_A)$ and $\mathcal{E}_{S_2}(\boldsymbol{w}_B)$ to S_{int}. Uploading the key-switching matrices to the *cloud* will enable it to find $\mathcal{E}_{S_{int}}(\boldsymbol{w}) = \mathcal{E}_{S_{int}}(\boldsymbol{w}_A) + \mathcal{E}_{S_{int}}(\boldsymbol{w}_B)$, where S_{int} is completely hidden from the *cloud*.
19: **procedure** COMPUTATION OF THE BIAS ITEM b
20: Cloud takes an $\alpha_i > 0$ and selects the corresponding $\boldsymbol{c}_{i,A}^*$ and $\boldsymbol{c}_{i,B}^*$.
21: Cloud computes the dot products $\mathcal{E}_{S_{int}}(\boldsymbol{w})^T \boldsymbol{c}_{i,A}^*$ and $\mathcal{E}_{S_{int}}(\boldsymbol{w})^T \boldsymbol{c}_{i,B}^*$ and sends them to A and B respectively.
22: A uses $vec(S_1^T S_{int})^T$ and B uses $vec(S_2^T S_{int})^T$ for decryption to find $\boldsymbol{w}^T \boldsymbol{x}_{i,A}^*$ and $\boldsymbol{w}^T \boldsymbol{x}_{i,B}^*$ respectively.
23: Using the symmetric key encryption A and B send each other these scalars and they individually compute $b = \boldsymbol{w}^T \boldsymbol{x}_{i,A}^* + \boldsymbol{w}^T \boldsymbol{x}_{i,B}^* - y_i$

Algorithm 2 SVM Classification

1: **procedure** PRIVATE-INPUTS
2: A, B possess secret-key $S_{int} = [I||T_{int}]$, bias b and $\boldsymbol{z}_A, \boldsymbol{z}_B$ which are arbitrary partitions of a vector $\boldsymbol{z}$.
3: Cloud possesses $\mathcal{E}_{S_{int}}(\boldsymbol{w})$.
4: **procedure** PRE-PROCESSING THE DATA VECTOR
5: A fills the missing entries in $\boldsymbol{z}_A$ by 0 to prepare $\boldsymbol{z}_A^*$.
6: B fills the missing entries in $\boldsymbol{z}_B$ by 0 to prepare $\boldsymbol{z}_B^*$.
7: A and B separately compute $\mathcal{E}_{S_{int}}(\boldsymbol{z}_A^*)$ and $\mathcal{E}_{S_{int}}(\boldsymbol{z}_B^*)$ and send them to the *cloud*.
8: **procedure** COMPUTATION OF ENCRYPTED DOT PRODUCT BY THE CLOUD
9: Cloud first adds $\mathcal{E}_{S_{int}}(\boldsymbol{z}_A^*)$ and $\mathcal{E}_{S_{int}}(\boldsymbol{z}_A^*)$ to get $\mathcal{E}_{S_{int}}(\boldsymbol{z})$.
10: Cloud finds the encrypted dot product of $\mathcal{E}_{S_{int}}(\boldsymbol{w})$ with $\mathcal{E}_{S_{int}}(\boldsymbol{z})$ and sends to both A and B.
11: **procedure** CLASSIFICATION
12: A and B decrypt the encrypted dot product $\mathcal{E}_{S_{int}}(\boldsymbol{w}^T\boldsymbol{z})$ and compare the value with b to output the class label.

Theorem 2. *Computation and uploading the key-switching matrices in Algorithm1 (lines 4,5,6 & 18) do not reveal any private information of parties A and B.*

Theorem 3. *With the assumption that the integer vector encryption is semantically secure, it is infeasible for the cloud to infer about the original data vectors of the parties with non-negligible probability during the SVM training phase given in Algorithm 1.*

Proof. First we observe that the parties replace the missing entries by zero and then encrypt therefore the cloud cannot infer about the positions of the missing entries. This can be proved using the indistinguishability of the real ideal paradigm model. Let $\mathcal{A}_C$ denote the semi-honest adversary corrupting the cloud in the real world. The view of $\mathcal{A}_C$ includes as inputs the encrypted vectors $\{\mathcal{E}_{S_1}(\boldsymbol{x}_A^*), \mathcal{E}_{S_2}(\boldsymbol{x}_B^*)\}$ with identical outputs. We can build, in the ideal world setting, a *simulator* $\mathcal{A}_{ideal}$ which computes $\{\mathcal{E}_{S_1}(\mathbf{1}), \mathcal{E}_{S_2}(\mathbf{1})\}$, outputs the same and returns to $\mathcal{A}_C$. Since $\mathcal{A}_C$ has no knowledge of the private keys S_1 and S_2, therefore if it can distinguish the views from the ideal and real worlds then it leads to an attack on the semantic security of the Integer Vector Encryption scheme. This contradiction proves that it is computationally infeasible for the *cloud* to decide how a vector $\boldsymbol{x}$ is partitioned between the two parties A and B.

We have already shown that the key-switching procedure is secure and based on it the security of transforming the ciphertexts under one key to another follows easily.

Now during the procedure of computation of encrypted dot product by the cloud, since the cloud does not know the positions of missing entries therefore it is infeasible for the cloud to infer about the original data vectors of the parties. However, from the computation of inner product some partial information is leaked e.g. the cloud can compute $\boldsymbol{x}_A^*.\boldsymbol{y}_A^*$, $\boldsymbol{x}_A^*.\boldsymbol{y}_B^*$, $\boldsymbol{x}_B^*.\boldsymbol{y}_A^*$ and $\boldsymbol{x}_B^*.\boldsymbol{y}_B^*$. We

observe that all these do not divulge any information about the individual entries of the vectors as the cloud do not have any information about how the vectors are partitioned between the two parties. There are more unknowns than the number of equations. This leakage of partial information is inevitable as the parties have already uploaded their encrypted data in the cloud and so whatever computation/transformation is needed to compute the unencrypted $\boldsymbol{x}.\boldsymbol{y}$, the same transformations can be performed on $\boldsymbol{x}_A^*$, $\boldsymbol{x}_B^*$, $\boldsymbol{y}_A^*$ and $\boldsymbol{y}_B^*$ which will ultimately reveal some partial information about the inner product.

During the Gram matrix computation procedure since the cloud solves a quadratic optimization problem with no sensitive private information, the procedure is secure.

The computation of encrypted weight vectors $\mathcal{E}_{S_1}(\boldsymbol{w}_A)$ and $\mathcal{E}_{S_2}(\boldsymbol{w}_B)$ is secure as the underlying integer vector scheme is semantically secure. Moreover, we have already proved that switching the keys of $\mathcal{E}_{S_1}(\boldsymbol{w}_A)$ and $\mathcal{E}_{S_2}(\boldsymbol{w}_B)$ to S_{int} is secure. Hence, the computation of $\mathcal{E}_{S_{int}}(\boldsymbol{w})$ is secure.

Computation of bias item b involve decryptions that are done at the user ends. Hence, it is secure.

Theorem 4. *The SVM classification phase given in Algorithm 2 is secure in the sense that the cloud cannot deduce the class label of the testing sample, given the semantic security of the integer vector encryption.*

Proof. The cloud stores $\mathcal{E}_{S_{int}}(\boldsymbol{w})$. The security of the procedure of computing the encrypted dot product of the test vector follows directly from the semantic security of the underlying integer vector encryption scheme. Lastly, the classification procedure can only be performed by the parties A and B. Thus the cloud has no information about the classification result.

4.1 Modified protocol

In order to reduce the communication rounds of the parties we now give a protocol for SVM training. We note that with this modification the cloud has no partial information about the secret keys of the parties (in Algorithm 1 the cloud knows the product $T_1^T T_2$ with no randomization). Moreover, there is only one round of communication between the parties during the training period. The computational cost is also reduced for the parties as they do not have to jointly collaborate to compute the key-switching matrix. The *classification* protocol remains the same as *Algorithm 2*.

Discussions on the security of the modified protocol : The main difference between the modified protocol and the original protocol is in the key-switching procedure. Parties A and B upload the key-switching matrices to convert their ciphertexts (under their personal keys) to ciphertexts under a common agreed key S_{int}. Once the *cloud* makes these transformations, it then performs addition on the ciphertexts and dot products. The security of the sum can be proved in the following manner:

Algorithm 3 Modified Protocol for SVM Training

procedure KEY-AGREEMENT

A, B secretly agree on an internal private key $S_{int} = [I||T_{int}]$ completely hidden from the cloud.

procedure KEY-SWITCHING

A uploads key-switching matrix $M_{S_1 \to S_{int}} = \begin{bmatrix} (vec(S_1)^T)^* - T_{int}A_a + E_a \\ A_a \end{bmatrix}$.

B uploads key-switching matrix $M_{S_2 \to S_{int}} = \begin{bmatrix} (vec(S_2)^T)^* - T_{int}A_b + E_b \\ A_b \end{bmatrix}$.

A or B uploads $M_{S_{int}^T S_{int} \to S_c} = \begin{bmatrix} (vec(S_{int}^T S_{int})^* - T_s A_{ab} + E_{ab} \\ A_{ab} \end{bmatrix}$.

procedure COMPUTATION OF ENCRYPTED DOT PRODUCT BY THE CLOUD

Using $M_{S_1 \to S_{int}}$, the *cloud* transforms the encryption of $\boldsymbol{x}_A^*$ and $\boldsymbol{y}_A^*$ to encryptions with underlying secret key $S_{int} = [I||T_{int}]$.

Using $M_{S_2 \to S_{int}}$, the *cloud* transforms the encryption of $\boldsymbol{x}_B^*$ and $\boldsymbol{y}_B^*$ to an encryption with underlying secret key $S_{int} = [I||T_{int}]$.

Cloud adds $\mathcal{E}_{S_{int}}(\boldsymbol{x}_A^*) + \mathcal{E}_{S_{int}}(\boldsymbol{x}_B^*)$ to output $\mathcal{E}_{S_{int}}(\boldsymbol{x})$.

Cloud adds $\mathcal{E}_{S_{int}}(\boldsymbol{y}_A^*) + \mathcal{E}_{S_{int}}(\boldsymbol{y}_B^*)$ to output $\mathcal{E}_{S_{int}}(\boldsymbol{y})$.

Cloud gets the encryption of the dot product $\mathcal{E}(\boldsymbol{x}^T\boldsymbol{y})$ with underlying secret key $S_{int}^T S_{int}$ of which the cloud is completely unaware.

Using the uploaded matrix $M_{S_{int}^T S_{int} \to S_c}$, *cloud* then transforms $\mathcal{E}(\boldsymbol{x}^T\boldsymbol{y})$ into $\mathcal{E}_{S_c}(\boldsymbol{x}^T\boldsymbol{y})$ and then decrypt to find $\boldsymbol{x}.\boldsymbol{y}$.

procedure COMPUTATION OF GRAM MATRIX BY THE CLOUD

Same as Algorithm 1.

procedure COMPUTATION OF ENCRYPTED WEIGHT-VECTOR BY THE CLOUD

Same as Algorithm 1.

procedure COMPUTATION OF THE BIAS ITEM b

Same as Algorithm 1.

Let $\mathcal{A}_C$ denote the semi-honest adversary corrupting the cloud in the real world. The view of $\mathcal{A}_C$ includes as inputs the encrypted vectors $\{\mathcal{E}_{S_{int}}(\boldsymbol{x}_A^*)$, $\mathcal{E}_{S_{int}}(\boldsymbol{x}_B^*)\}$ and output $\mathcal{E}_{S_{int}}(\boldsymbol{x}_A^*) + \mathcal{E}_{S_{int}}(\boldsymbol{x}_B^*)$. In the ideal world setting, a *simulator* $\mathcal{A}_{ideal}$ computes $\{\mathcal{E}_{S_{int}}(\mathbf{1}), \mathcal{E}_{S_{int}}(\mathbf{2})\}$, outputs their sum and returns $\{\mathcal{E}_{S_{int}}(\mathbf{1}), \mathcal{E}_{S_{int}}(\mathbf{2})$, $\mathcal{E}_{S_{int}}(\mathbf{1}) + \mathcal{E}_{S_{int}}(\mathbf{2})\}$ to $\mathcal{A}_C$. Since $\mathcal{A}_C$ has no knowledge of the private key S_{int}, therefore if it can distinguish the views from the ideal and real worlds with non-negligible probability then it leads to an attack on the semantic security of the Integer Vector Encryption scheme. Therefore, it is secure to compute the sums. Lastly, computing the dot products is secure because the view of $\mathcal{A}_C$ includes $\{\mathcal{E}_{S_{int}}(\boldsymbol{x}), \mathcal{E}_{S_{int}}(\boldsymbol{y})\}$ as inputs and $\mathcal{E}(\boldsymbol{x}^T\boldsymbol{y})$ as output which is indistinguishable from the simulated view $\{\mathcal{E}_{S_{int}}(\mathbf{1}), \mathcal{E}_{S_{int}}(\mathbf{2})$, $\mathcal{E}_{S_{int}}(\mathbf{1}^T\mathbf{2})\}$.

From the above discussion we now have the following theorem.

Theorem 5. *Under the assumption that the integer vector encryption is semantically secure, it is infeasible for the cloud to infer about the original data vectors of the parties with non-negligible probability during the SVM training phase given in Algorithm 3.*

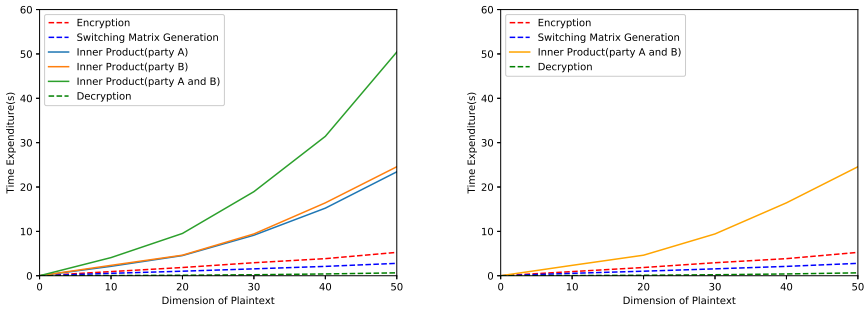
Remark 1. We note that the modified protocol can be made fully secure, that is no partial information about the inner products will be revealed if we assume the existence of two non-colluding semi-honest servers.

5 Experimental Results and Comparison

We use Number Theory Library (NTL) [1] for our implementation. The configuration of our PC is Ubuntu 16.04 64 bit operating system with Intel Core(TM) $i5$ $CPU(2$ $cores)$, 3.60 GHz and 8 GB memory. Party A and Party B are two parties who own training samples such that if A owns some entries of a sample, then other entries are owned by B. For example, if the sample dimension is 10 and A owns any 6 entries, then the rest 4 entries of that sample are with B. In Fig. 3a, we show running time for encryption, key-switching matrix generation, dot product computation and decryption using algorithm 1. In Fig. 3b, we show the running times of the same operations when we use algorithm 3. We take 100 many samples of data vectors with dimensions varying from 10 to 50.

Encryption and decryption for both the parties are very efficient as can be seen from Fig. 3. Using algorithm 1, in order to compute the encrypted Gram matrix, the *cloud* needs to compute $2\times\frac{100\times101}{2}+100\times99 = 20000$ many products. In general, for n samples, number of products to be computed by the cloud will be $2n^2$. However, in algorithm 3, the cloud needs to compute $\frac{n(n+1)}{2}$ many products.

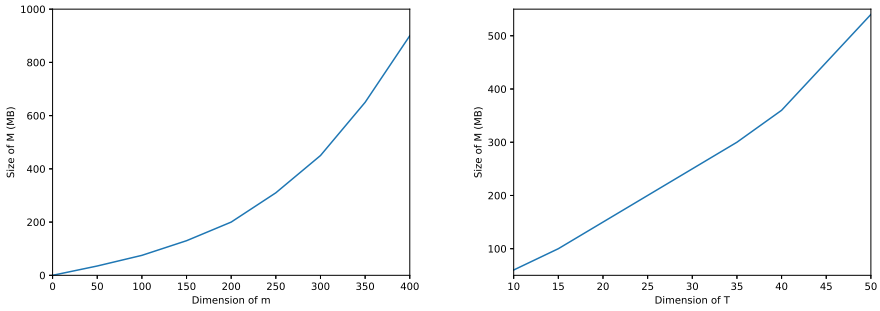
The works [8] and [10] give asymptotic results on complexity and no concrete results. We compare our scheme with [9] and [20]. We encrypt 3.6×10^5 samples of dimension 10. The time required for encryption is 158 s, whereas [20] needs 69 s, and [9] needs 360 s. Experimental Platform for [9] (8x Intel Xeon CPU, 3.6 GHz) and [20] (Intel Core(TM) i7 CPU(2 cores), 2.2 GHz) is better than us. So, integer vector encryption performs better than FHE, which is consistent with the test results of [21].



(a) Algorithm 1 (b) Algorithm 3

Fig. 3: Performance of our scheme

The communication overhead is mainly due to transmission of key-switching matrices. The overhead depends on dimension of sample (m) and dimension of the secret key (T). In Fig. 4a, we fix $T = 10$ and vary m of samples from 50 to 400. The size of key-switching matrix M changes from MB to ≈ 1 GB. We observe that, if T is fixed, overhead is $O(m^2)$ which is consistent with [20]. In Fig. 4b, we fix the dimension of m at 100 and vary the dimension of T from 10 to 50. We observe that the size of M varies from 50 MB to ≈ 500 MB.



(a) When m varies (b) When T varies

Fig. 4: Communication overhead between a user and the cloud

6 Conclusion

We have discussed two outsourced privacy preserving protocols for inner product of vectors when their entries are distributed among parties. It finds natural ap-

plication in private SVM when the underlying database is arbitrarily partitioned. Allowing some minimal leakage about the inner products our scheme provides an efficient algorithm for SVM training. Fully secure computation is possible with our methodology if existence of two non-colluding servers is assumed. Our protocol for SVM classification is fully secure. Estimating the trade off between efficiency and privacy can be an interesting problem for further study.

References

1. NTL: A Library for doing Number Theory. *http : //www.shoup.net/ntl/*
2. Agrawal, R. and Srikant, R., "Privacy-preserving Data Mining", ACM SIGMOD Conference on Management of Data 2000: 439-450 (2000).
3. Boneh, D., Goh, E.-U. and Nissim, K., "Evaluating 2-DNF Formulas on Ciphertexts", TCC 2005: 325-341 (2005).
4. Cortes, C. and Vapnik, V., "Support-Vector Networks", Machine Learning 20(3): 273-297 (1995).
5. Du, W. and Atallah, M.J., "Privacy-Preserving Cooperative Statistical Analysis", ACSAC 2001: 102-110 (2001).
6. Goethals, B., Laur, S., Lipmaa, H. and Mielikainen, T., "On Private Scalar Product Computation for Privacy-Preserving Data Mining", ICISC 2004: 104-120 (2004).
7. Lindell, Y. and Pinkas, B., "Privacy Preserving Data Mining", J. of Cryptology 15(3): 177-206 (2002).
8. Liu, F., Ng, W.K. and Zhang, W., "Encrypted Scalar Product Protocol for Outsourced Data Mining", IEEE CLOUD 2014: 336-343 (2014).
9. Liu, F., Ng, W.K. and Zhang, W., "Encrypted svm for outsourced data mining", IEEE CLOUD 2015: 1085-1092 (2015).
10. Liu, F., Ng, W.K. and Zhang, W., "Secure scalar product for big-data in mapreduce", IEEE Big Data Service 2015: 120-129 (2015).
11. Lopez-Alt, A., Tromer, E. and Vaikuntanathan, V., "On-the-fly multiparty computation on the cloud via multikey fully homomorphic encryption", STOC 2012: 1219-1234 (2012).
12. Mehnaz, S. and Bertino, E., "Privacy-Preserving Multi-Party Analytics over Arbitrarily Partitioned Data", CLOUD 2017: 342-349 (2017).
13. Peter, A., Tews, E. and Katzenbeisser, S.," Efficiently Outsourcing Multiparty Computation Under Multiple Keys", IEEE Trans. Info. Forensics and Security 8(12): 2046-2058 (2013).
14. Vaidya, J. and Clifton, C., "Privacy preserving association rule mining in vertically partitioned data", KDD 2002: 639-644 (2002).
15. Vaidya, J., Yu, H. and Jiang, X., "Privacy-preserving SVM classification", Knowl. Inf. Syst. 14(2): 161-178 (2008).
16. Wang, B., Li, M., Chow, S. S. and Li, H., "Computing encrypted cloud data efficiently under multiple keys", IEEE CNS 2013: 504-513 (2013).
17. Wang, B., Li, M., Chow, S. S. and Li, H., "A tale of two clouds: Computing on data encrypted under multiple keys", IEEE CNS 2014: 337-345 (2014).
18. Yu, H., Vaidya, J. and Jiang, X., "Privacy-Preserving SVM Classification on Vertically Partitioned Data", PAKDD 2006: 647-656 (2006).
19. Yu, H., Jiang, X. and Vaidya, J., "Privacy-preserving SVM using nonlinear kernels on horizontally partitioned data", SAC 2006: 603-610 (2006).

20. Zhang, J., Wang, X., Yiu S.M., Jiang Z.L. and Li, J., "Secure Dot Product of Outsourced Encrypted Vectors and its Application to SVM", SCC@AsiaCCS 2017: 75-82 (2017).
21. Yu, A., Lai, W.L. and Payor, J., "Efficient integer vector homomorphic encryption", https://courses.csail.mit.edu/6.857/2015/files/yu-lai-payor.pdf (2015).
22. Zhou, H. and Wornell, G.W.,"Efficient homomorphic encryption on integer vectors and its applications", ITA 2014: 1-9 (2014).