2019 Master's Thesis

Secure Naïve Bayes Classification Protocol over Encrypted Data Using Fully Homomorphic Encryption

A Thesis Submitted to the Department of Computer Science and Communications Engineering, the Graduate School of Fundamental Science and Engineering of Waseda University in Partial Fulfillment of the Requirements for the Degree of Master of Engineering

Submission Date: July 22nd, 2019

Advisor: Prof. Hayato Yamana
Research guidance: Research on Parallel and Distributed Architecture

Department of Computer Science and Communications Engineering,
Graduate School of Fundamental Science and Engineering,
Waseda University
Student ID: 5117FG27-0

Yoshiko Yasumura

Abstract

Machine learning classification has a wide range of applications. In the big data era, a client may need to classify a large amount of data that has many features, resulting in a heavy computation at the client. Using a cloud server to outsource such classification tasks, we can reduce this computational burden. At the same time, an entity may wish to provide a classification model and classification services to such clients as a part of Machine Learning as a Service (MLaaS). However, applications such as medical diagnosis require sensitive data from the entity and the client that they may not want to reveal to the cloud such as the classification model and client's data.

Fully homomorphic encryption (FHE), in which an arbitrary number of arithmetic operations can be performed over encrypted data without decryption, enables secure computation. By applying FHE to machine learning classification, the client can outsource classification tasks to a cloud server without revealing any data. However, the existing studies on machine learning classification over FHE do not achieve the scenario of outsourcing classification tasks to a cloud server while preserving the privacy of the classification model, client's data and result from the cloud.

In this work, we apply FHE to a naïve Bayes classifier and propose a secure classification protocol in which we preserve the privacy of the classification model, client's data, and result while outsourcing the computations to a cloud server. In our protocol, the cloud does not learn anything about the classification model, client's data or result, and the client learns only the result. To the best of our knowledge, our work is the first to present a concrete classification protocol that satisfies the above scenario.

We implemented our protocol on HElib and tested its performance by measuring its computation time and communication cost. Our experimental results show that our protocol runs in 1.5 s with 4.0 MB of communication cost for classifying a data using a two-class classifier.

Contents

1. Introduction

In recent years, machine learning classification has been used in various applications from spam classification to medical diagnosis. In the big data era, a client may need to classify a large amount of data that has many features, leading to a heavy computational burden at the client's local resource. To reduce this computational burden on the client, it can outsource the classification tasks to a cloud server and obtain the classification result. However, applications such as medical diagnosis require sensitive data that the client may not want to reveal to the cloud. Clients can encrypt their sensitive data to ensure its privacy then upload them to the cloud, but classification cannot be performed when the data are encrypted by a traditional encryption scheme such as the advanced encryption standard (AES).

Fully homomorphic encryption (FHE) [1] is an encryption scheme that achieves secure computation by enabling an arbitrary number of arithmetic operations over encrypted data without decryption. By applying FHE to machine learning classification, we can realize secure classification over encrypted data at a cloud server while preserving the privacy of the client's data. By doing so, the client can securely outsource their data classification tasks and reduce the computational burden at the client side. There are, however, several challenges when applying FHE to machine learning such as its high computational cost as well as its difficulty to compute over real numbers, perform divisions and use if-statements. Thus, the application of FHE to machine learning is currently widely researched.

In this work, we consider the scenario in which an entity wishes to provide a classification model and classification services at a cloud server as a part of Machine Learning as a Service (MLaaS), but while preserving the privacy of the classification model because it is trained using data that are as equally sensitive as the client's data. In such scenario, in addition to preserving the privacy of the client's data, preserving the privacy of the classification model becomes important as well. Thus, the ideal scenario would be to outsource the client's classification tasks to a cloud server while preserving the privacy of the client's data and result from the cloud as well as preserving the privacy of the classification model from both the cloud and client.

Several studies have been conducted on the secure classification of encrypted data using FHE for various classifiers. Studies such as [2-7] proposed various secure classification protocols under different system models for different scenarios. These methods, however, do not achieve the ideal scenario of outsourcing the classification tasks to a cloud server while preserving the privacy of the client's data and result from the cloud, and the privacy of the classification model from both the cloud and client. Studies that focus on both training and classification over encrypted data have been conducted as well. Studies such as [8-13] proposed machine learning methods in which the functions used in the classifiers are approximated so that the training can be performed using FHE. Although these studies focus on the classification phase as well, they do not mention any real-world system model or protocol for this phase.

Kim et al. [14] and Li et al. [15] proposed a protocol for both secure training and classification in which they introduce a third party who holds the secret key of the system and is responsible for decryption of all ciphertext, which we will refer to as a decryption server in this paper. By doing so, both the cloud server and client will not be able to decrypt each other's data. In Kim et al.'s work, however, some information about the classification model and client's data is leaked to the decryption server through the intermediate results during classification. In Li et al.'s work, they use expensive proxy re-encryption to re-encrypt ciphertexts. Their proxy re-encryption is based on the bootstrapping technique proposed by Gentry [1] which involves heavy computation that lasts from a few seconds to a few minutes [16] and can lead to a potential bottleneck. In fact, this proxy re-encryption is unnecessary as the protocol can be realized without it. In addition, because the proposed protocol by Li et al. is generic, it does not indicate any concrete classification method using FHE despite its limitations, which is the challenging part when applying FHE to any application.

As such, the existing studies on machine learning classification over FHE do not achieve the ideal scenario of outsourcing the classification tasks to a cloud server while preserving the privacy of the client's data and result from the cloud, and the privacy of the classification model from both the cloud and client. In these works, it is either i) the classification is performed at the cloud, but some information about the classification model, client's data and/or intermediate results is revealed to the participating parties [4, 7, 14], or ii) the privacy of the classification model, client's data, and the intermediate results are preserved, but the classification is performed at the client [2, 3, 7]. In case of Li et al.'s work [15], their protocol achieves the ideal scenario, but adopts expensive proxy re-encryption and does not describe any concrete classification method.

In this paper, we focus on the naïve Bayes classifier and propose a secure classification protocol to achieve the ideal scenario. Although we focus on the naïve Bayes classifier in this work, our protocol can be applied to other machine learning algorithms with some modifications to the protocol. Like in the works by Kim et al. [14] and Li et al. [15], we introduce a trusted third party, which we refer to as a trusted authority (TA), who holds the secret key of the FHE used in the system and is responsible for the decryption of all ciphertexts. By introducing a TA who holds the secret key, we can ensure the privacy of the classification model from both the cloud and client, and the client's data from the cloud because they cannot decrypt anything without the secret key. However, if we simply let the TA decrypt all ciphertexts, the classification result will be known to the TA, which the client may not desire. Thus, we blind the intermediate and classification results so that the TA does not learn anything even after decryption. Also, our protocol does not require the expensive proxy re-encryption as in Li et al.'s work.

More concretely, our proposed protocol satisfies the following requirements:

- classification is outsourced to a cloud server;
- the classification model is stored in the cloud as a ciphertext;
- participating parties do not learn anything about the classification model, client's data or result;
- the client learns only the classification result and nothing else.

The contributions of our work are as follows: First, our protocol satisfies all the above requirements as opposed to the existing works in which at least one requirement remains unsatisfied. Second, our protocol does not involve the expensive and unnecessary proxy re-encryption used in Li et al.'s work and thus achieves higher computational efficiency. To the best of our knowledge, our work is the first to present a concrete classification method that satisfies all the above requirements.

The rest of this paper is organized as follows. In Section 2, we review FHE and the naïve Bayes classifier. In Section 3, we present related works on machine learning over FHE in detail. In Section 4, we propose our system model and the secure classification protocol. In Section 5, we present our experimental results. In Section 6, we present some techniques that can be applied to our proposed protocol to optimize the computational cost. Finally, we give our conclusion in Section 7.

2. Preliminaries

2.1. Fully homomorphic encryption (FHE)

Fully homomorphic encryption (FHE) [1] is an encryption scheme that achieves secure computation by enabling an arbitrary number of arithmetic operations over encrypted data without decryption. The notion of FHE was first introduced by Rivest et al. [17] in 1978. After their work, the realization of FHE remained as an open problem until the first feasible construction was given by Gentry [1] in 2009. Following Gentry's work, many FHE schemes were proposed over the years such as the GSW scheme proposed by Gentry et al. [18], the BGV scheme proposed by Brakerski et al. [19], and the B/FV scheme based on works by Brakerski [20] and Fan and Vercauteren [21].

Using FHE to encrypt data, we can securely outsource computations over encrypted data to a third party without revealing any information on the data. There are, however, several challenges to FHE such as its high computational cost as well as its difficulty to compute over real numbers, perform divisions, and use if-statements. Also, multiplication operations are computationally more expensive than addition operations. Despite these current challenges, there are many potential applications of FHE to various tasks such as data mining [22], database queries [23], and machine learning [2-15] so that computation can be outsourced to a third party, such as a cloud server, while preserving the privacy of sensitive data. Thus, studies on FHE and its application are currently widely under research.

FHE consists of three algorithms: $\mathsf{KeyGen}, \mathsf{Enc}_{pk}$, and Dec_{sk}. Here, KeyGen takes FHE parameters $param$ and security parameter λ as inputs, and outputs a pair of secret key sk and public key pk; $\mathsf{Enc}_{pk}(m)$ denotes a ciphertext of message m, which may be a vector of values, encrypted by a given pk; and $\mathsf{Dec}_{sk}(ct)$ denotes the decryption result of a ciphertext ct decrypted by a given sk. Concretely, FHE allows addition and multiplication operations such that $\mathsf{Dec}_{sk}(\mathsf{Enc}_{pk}(m_1) \oplus \mathsf{Enc}_{pk}(m_2)) = m_1 + m_2$ and $\mathsf{Dec}_{sk}(\mathsf{Enc}_{pk}(m_1) \otimes \mathsf{Enc}_{pk}(m_2)) = m_1 m_2$, where $\oplus$ and $\otimes$ are homomorphic addition and multiplication operations, respectively. It is also possible to perform a homomorphic operation between a ciphertext and plaintext, e.g. $\mathsf{Dec}_{sk}(\mathsf{Enc}_{pk}(m_1) \oplus m_2) = m_1 + m_2$ and $\mathsf{Dec}_{sk}(\mathsf{Enc}_{pk}(m_1) \otimes m_2) = m_1 m_2$, which is computationally less expensive than a homomorphic operation between two ciphertexts. For simplicity, we use the same symbols $\oplus$ and $\otimes$ to denote homomorphic operations between a ciphertext and a plaintext as well.

Smart and Vercauteren [24] proposed a packing (or batching) technique for FHE in which multiple values can be encrypted into a single ciphertext so that single instruction multiple data (SIMD) style operations can be performed over each encrypted value. More specifically, using the packing technique, we can construct a ciphertext consisting of s slots, where s depends on the FHE parameters, then encrypt multiple values by storing each value into a slot and encrypting it as a single ciphertext. A SIMD style operation between two ciphertexts (or between a ciphertext and a plaintext vector) takes place as a slot-wise operation. Moreover, the slots within the ciphertext can be rotated, like in a linear array. Using the packing technique and SIMD style operations, we can efficiently process multiple values simultaneously.

2.2. Naïve Bayes classifier

The naïve Bayes classifier is a simple yet powerful classifier that is used for a wide range of applications. The classifier is based on Bayes' theorem and assumes that features are independent of each other.

The naïve Bayes classifier takes input data $\boldsymbol{x} = (x_1, x_2, \ldots, x_f)$, where f is the number of feature values, and outputs a classification probability $p_i = p(C = c_i | X = \boldsymbol{x})$ for each class c_i, where $i = 1, 2, \ldots, N$ and N is the number of classes. The classification model consists of class probability $p(C = c_i)$ for each class c_i, which is the probability that class c_i occurs, and conditional probability $p(X_j = x_j | C = c_i)$ for each x_j and each c_i, which is the probability that feature value x_j occurs in c_i. Based on Bayes' theorem and an assumption that features are independent of each other, the classification probability p_i is calculated as follows:

$$\begin{aligned} p_i &= p(C = c_i | X = \boldsymbol{x}) \\ &= \frac{p(C = c_i) p(X = \boldsymbol{x} | C = c_i)}{p(X = \boldsymbol{x})} \\ &= p(C = c_i) \prod_{j=1}^{f} p(X_j = x_j | C = c_i). \end{aligned} \tag{1}$$

Once p_i for each class is obtained, the classifier takes the class c_i with the highest p_i and outputs it as the final classification result.

In this work, we use log probabilities to calculate the classification probability p_i. Using log probabilities, we can calculate p_i using only addition instead of multiplication, and thus reduce the computational cost over FHE. Taking the logarithm of Equation (1), we obtain the following:

$$p'_i = \log p(C = c_i) + \sum_{j=1}^{f} \log p(X_j = x_j | C = c_i) \tag{2}$$

3. Related Work

In this section, we introduce the related works on machine learning over FHE in more detail. In Section 3.1, we introduce related works focusing on classification of encrypted data using FHE. In Section 3.2, we introduce related works focusing on both training and classification using FHE.

3.1. Secure classification over FHE

In this section, we introduce related works that focus only on classification of encrypted data using FHE. In all these works, it is assumed that the classification model has already been trained.

Bost et al. [2] proposed secure classification protocols for three classifiers: naïve Bayes, decision trees, and hyperplane decisions. Their protocols use two additive homomorphic encryption schemes that enable only addition operations, namely quadratic residuosity (QR) and Paillier, and FHE to perform classification. In their work, the classification model is assumed to be trained on plaintext data, then encrypted and stored in a cloud server. At classification, the encrypted model is sent from the cloud server to the client, who then computes the classification probability for each class. Then, the client interacts with the cloud server multiple times using the two additive HE to obtain the index of the class with the highest probability. In their protocol, FHE is used in the decision tree classifier to efficiently compute data using SIMD. Their protocol, however, has disadvantages that the client computes the classification probability and must interact multiple times with the cloud server, causing computational burden on the client.

Wood et al. [3] proposed a secure classification protocol for the naïve Bayes classifier based on [2] with a different comparison method. However, it still has the same disadvantages that the client computes the classification probability and must interact multiple times with the cloud server.

Sun et al. [4] proposed an FHE scheme and applied it to the same classifiers as in [2]. In their work, the classification model is stored at the cloud server as a ciphertext. At classification, the classification probabilities are computed at the cloud, but multiple interactions occur between the client and cloud when computing for the class with the highest probability. More importantly, some information about the classification model is revealed to the client through the intermediate results that are decrypted by the client.

Khedr et al. [5] focused on two classifiers, Bayesian spam filter and decision tree, and proposed an optimized FHE scheme for the two classifiers based on mathematical observations of FHE. Although they have applied their proposed FHE scheme to the two classifiers and proposed a classification method, they did not indicate any real-world system model or protocol.

Dowlin et al. [6] proposed a classification method for neural networks using FHE. In their method, the classification model is trained using a regular neural network and plaintext data. To perform classification of encrypted data over FHE, they replaced functions used in the neural network (e.g., replaced the activation function with a low-degree polynomial) so that FHE can be used efficiently. Although they have indicated the classification method, they did not indicate any real-world system model or protocol as well.

Park et al. [7] proposed two secure classification protocols for the naïve Bayes classifier in which there is a data provider that trains the classification model on plaintext data, a cloud server, and multiple clients. The first protocol is a server-centric protocol, in which classification is outsourced to a cloud server, but the model is stored in the cloud as a plaintext. In the server-centric protocol, clients have their own pair of keys that is used to encrypt their data when outsourcing classification to the cloud. Because the model needs to be encrypted for each client to perform classification, the model cannot be stored in the cloud as a ciphertext. The second protocol is a user-centric protocol, in which the classification model is encrypted but the classification is performed at the client (user). In the user-centric protocol, a single pair of keys is used within the system and is maintained by the cloud server. Instead of storing the classification model in the cloud, the data provider encrypts it and directly sends it to the client. Then, the client performs the classification, blinds the result and sends it to the cloud for decryption. Once decrypted, the cloud sends back the blinded result to the client, who then unbinds it to obtain the classification result. In this protocol, computational burden is induced on the client as the client performs the classification. In summary, the two protocols have a trade-off between the privacy of the classification model and computational burden on the client.

3.2. Secure training and classification over FHE

In this section, we introduce related works that focus on both training a classification model over encrypted data and classification of encrypted data using FHE.

Aslett et al. [8] proposed a tailored algorithm for naïve Bayes classifier and decision trees in which the operations within the classifiers are replaced with addition and multiplication operations such that FHE can be used to train a classification model over encrypted data. Graepel et al. [9] focused on Linear Means classifier and Fisher’s Linear Discriminant classifier in which they expressed the two classifiers using a low-degree polynomial to avoid division operations, which is difficult to perform using FHE. Several studies have been conducted on training neural networks [10, 11] and logistic regression [12, 13] in which they approximate the activation functions in neural networks and the sigmoid functions in logistic regression using low-degree polynomials so that FHE can be used. These studies on secure training over encrypted data using FHE focus on approximating the functions used in the classifiers so that the training can be performed using FHE. Although these works also indicate the classification method, they do not mention any real-world system model or protocol.

Training classifiers over encrypted data has a major challenge in its computational complexity. As noted in [6], training classifiers, especially neural networks, is currently computationally expensive even with plaintext data. Thus, applying FHE to neural network will make the training even more expensive and thus not practical for real-world systems. In addition, training over encrypted data will make it difficult to inspect data and the trained model, and to tune the parameters for training the neural network.

Kim et al. [14] proposed a secure training and classification protocol for the naïve Bayes classifier in which the privacy of the classification model is preserved while outsourcing the computation to a cloud server. In their protocol, they introduce a decryption server who holds the secret key and is responsible for decryption of all ciphertexts. In their protocol, the class indexes are permuted so that the decryption server does not learn the actual index of the intermediate results and the classification result. However, the decryption server learns all the values that are being compared, and thus learns some information about the client's input and the classification model. In addition, their proposed protocol involves many multiplication operations, which can, in fact, be optimized by using log probabilities and addition operations.

Li et al. [15] proposed a general protocol for both secure training and classification over encrypted data. In their work, the privacy of the classification model, client's data, and its result are preserved while outsourcing the computation to a cloud server. In their protocol, each client has their own public key pk_c and secret key sk_c pair, which is used to encrypt their own data when outsourcing their data to the cloud for computation. As computation cannot be performed when data are encrypted under a different key, proxy re-encryption based on Gentry's [1] bootstrapping technique is used to re-encrypt the ciphertexts so that it is encrypted under a common public key pk_0. To prevent the cloud from decrypting the client's data, a decryption server, which holds $sk_0$and is responsible for decrypting the intermediate results, is introduced.

In the training phase, they consider the scenario where data providers (clients) wants to jointly train a classifier without revealing their data to each other. Each data provider encrypts their data with their pk_c and sends the encrypted data to the cloud along with a re-encryption key that is needed to re-encrypt the data. Upon receiving the encrypted data and re-encryption key, the cloud server re-encrypts the data so that they are encrypted under pk_0. Then, the cloud server trains the classifier using the encrypted data.

Similarly, in the classification phase, each client encrypts their data with their keys pk_c, then sends the encrypted data and re-encryption key to the cloud. The cloud re-encrypts the data under pk_0, then performs the classification. Once the classification has been performed, the cloud blinds the result with a random number and sends it to the decryption server, which decrypts the blinded result. The decryption server encrypts the blinded result with pk_c and sends it to the cloud. The cloud server removes the random number from the ciphertext using pk_c, then sends it to the client, who decrypts the data with sk_c to obtain the classification result.

Their proposed protocol preserves the privacy of the classification model, client's data, and result. However, bootstrapping, which the proxy re-encryption is based on, is an expensive operation and may produce a bottleneck because the cloud will need to re-encrypt all clients' data. In fact, this proxy re-encryption is unnecessary as the clients can encrypt their data using pk_0 from the start of the protocol. In addition, because their protocol is generic, it does not indicate any concrete training or classification method using FHE, which is the challenging part when applying FHE to any applications.

3.3. Summary of classification protocols

We summarize the classification protocols of the related works in Table 1. We omit works that do not indicate any real-world system models, protocols or methods as we focus on devising a secure classification protocol and method in this paper. Many works focus on the naïve Bayes classifier because it is a simple and fast yet powerful classifier and one of the common classifiers that is used for various applications.

As can be seen from Table 1, the past studies on machine learning classification over FHE do not achieve the ideal scenario of outsourcing the classification tasks to a cloud server while preserving the privacy of the client's data and result from the cloud, and the privacy of the classification model from both the cloud and client. In these works, it is either i) the classification is performed at the cloud, but some information about the classification model, client's data and/or intermediate results is revealed to the participating parties, or ii) the privacy of the classification model, client's data, and the intermediate results are preserved, but the classification is performed at the client.

Table 1. Summary of classification protocols of related work

Year	Related Work		Classifier	Privacy of		Classification at	Number of parties
				Classification model	Intermediate results		
2015	Bost et al. [2]		Naïve Bayes, decision tree, hyperplane decision	Preserved	Preserved	Client	2
2018	Wood et al. [3]		Naïve Bayes	Preserved	Preserved	Client	2
2018	Sun et al. [4]		Naïve Bayes, decision tree, hyperplane decision	Preserved	Not preserved	Cloud	2
2017	Park et al. [7]	Server-centric	Naïve Bayes	Not preserved	Preserved	Cloud	3
		User-centric		Preserved	Preserved	Client	3
2018	Kim et al. [14]		Naïve Bayes	Preserved	Not preserved	Cloud	3

4. Proposed Method

In this section, we present our naïve Bayes classification protocol. We consider the scenario in which we have i) a model provider who wants to provide a classification model for a classification service while preserving the privacy of their classification model, and ii) clients who want to outsource the classification task to reduce the computational burden at the client while preserving the privacy of their data and the result. As such, we consider the scenario in which the classification task is outsourced to a cloud server, which we refer to as a computation server (CS), while preserving the privacy of the classification model, client's data, and classification result. We also introduce a trusted third party, which we refer to as a trusted authority (TA), who holds the secret key of the FHE used in the system and is responsible for the decryption of all ciphertexts.

In our protocol, no participating parties learn anything about the classification model, client's data, or classification result during classification of data. The client learns only the classification result and does not learn anything about the classification model that was used to classify the data. Although the TA is responsible for decryption of all ciphertexts, we ensure that it does not learn any information on the classification model, client's data, or result. Like other works [7, 15], we assume that all participating parties are honest-but-curious, i.e., they follow the protocol but try to learn some information, and that the participating parties do not collude with each other.

In this work, we assume that the classification model is trained over plaintext by a model provider, who then encrypts and uploads the classification model to a CS. To encrypt the classification model with FHE, the class probabilities and the conditional probabilities need to be represented as integers. Thus, after taking the logarithm of the probabilities, each value is scaled and rounded to integers. Precisely, we define the above operation as $p'(Y) = \lfloor S \log p(Y) \rceil$, where S is the scaling factor.

In our protocol, we use the BGV scheme proposed by Brakerski et al. [19], which allows us to encrypt vector of integers and use SIMD style operations. The client's data are represented as a feature vector $\boldsymbol{x}$ and encrypted as a single ciphertext.

4.1. Protocol Overview

A high-level overview of our classification protocol is shown in Fig. 1, where we have a model provider, multiple clients, a CS, and a TA. First, the model provider encrypts the classification model and uploads it to the CS for storage. At classification, a client encrypts their data and generates a permutation vector, then sends them to the CS as a query. Upon receiving the query, the CS computes the classification probability p'_i for each class and then permutes the results using the permutation vector. To obtain the permuted class index with the highest p'_i, the CS interacts with the TA to perform comparisons. After all comparisons, the TA sends the obtained permuted index to the client, who reverses the permutation to obtain the classification result. During the protocol, the CS and TA do not learn anything about the classification model or the client's data.

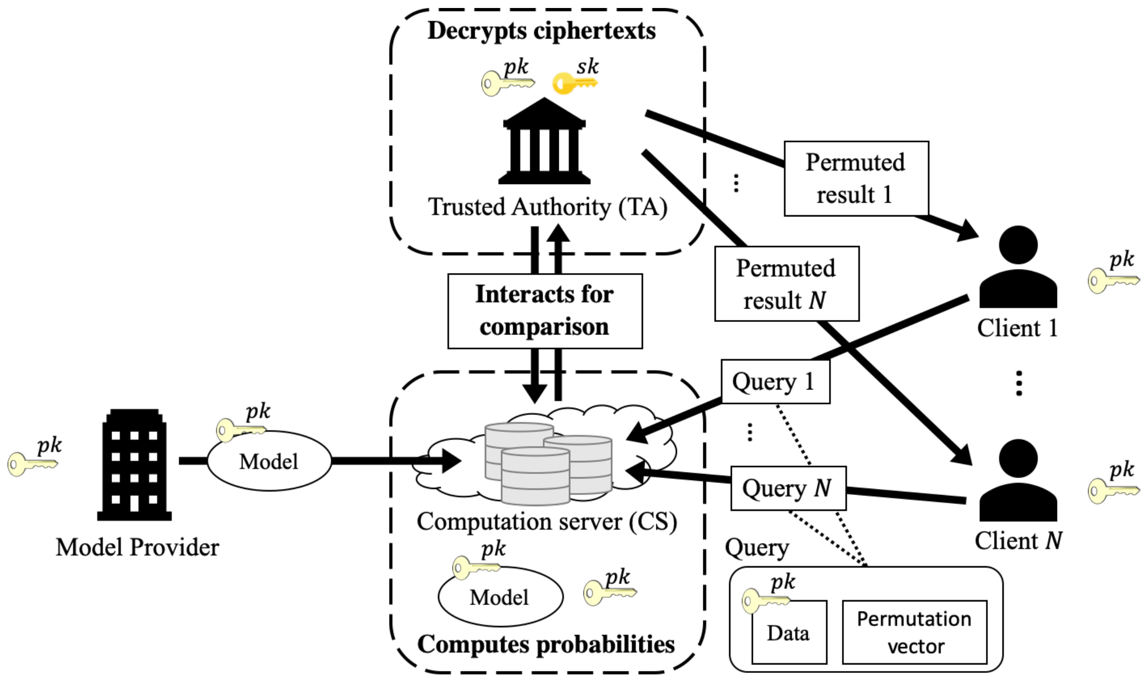


Fig. 1. Overview of our system model and protocol

4.2. Setup and Model Encryption

At setup, the TA runs FHE's KeyGen algorithm with given $param$ and λ to generate a pair of secret key sk and public key pk. The TA distributes pk to the CS, model provider and clients. The model provider then encrypts the trained classification model using pk. From here on, we use $[\![m]\!]$ to denote $\mathsf{Enc}_{pk}(m)$.

To efficiently utilize the SIMD style operations of FHE, the model provider stores the class probability for class c_i and all conditional probabilities for class c_i into a vector $\boldsymbol{v}_{c_i}$ of length f+1, where $\boldsymbol{v}_{c_i}[j] = p'(X_j = x_j|C = c_i)$ and $\boldsymbol{v}_{c_i}[f+1] = p'(C = c_i)$, then encrypts it as a single ciphertext. Thus, the encrypted classification model is a set of ciphertexts $[\![\boldsymbol{v}]\!] = \{[\![\boldsymbol{v}_{c_1}]\!], [\![\boldsymbol{v}_{c_2}]\!], \dots, [\![\boldsymbol{v}_{c_N}]\!]\}$, where N is the number of classes. After encrypting the classification model, the model provider uploads it to the CS so that clients can use it for classification. The information of N, f, and how the vector was constructed must be made public so that the clients can construct the feature vector accordingly.

4.3. Classification Protocol

At classification, a client represents his/her data as a feature vector $\boldsymbol{x}$ of length $f+1$ based on the public vector information. $\boldsymbol{x}$ consists of 0 and 1 where $\boldsymbol{x}[j]$ is set to 1 iff feature x_j exists, and 0 otherwise. In addition, $\boldsymbol{x}[f+1]$ is set to 1 as well. Then, the client encrypts $\boldsymbol{x}$ with pk and also generates a permutation vector π of length N, where N is the number of classes, which will be later used to permute the ciphertexts to hide the classification result from the TA. The client then sends the encrypted data $[\![\boldsymbol{x}]\!]$ and permutation vector π to the CS as a query. Upon receiving the query from the client, the CS performs the classification protocol as shown in Algorithm 1, which we describe below.

For each class c_i, the CS multiplies $[\![\boldsymbol{x}]\!]$ with encrypted model $[\![\boldsymbol{v}_{c_i}]\!]$ to extract the probabilities corresponding to the client's data. If the value of $\boldsymbol{x}$ at the j-th position is 1, the value of $\boldsymbol{v}_{c_i}$ at the j-th position corresponding to $p'(X_j = x_j | C = c_i)$ can be extracted; otherwise, it will be 0. Then, the CS sums the values in all the slots using TotalSums operation proposed by Halevi and Shoup [25], which is shown in Algorithm 2, to obtain the classification probability $[\![\boldsymbol{p'}_i]\!]$, where all slots contain the value p'_i. Once $[\![\boldsymbol{p'}_i]\!]$ for each class is calculated, the CS permutes the ciphertexts using π as $\{[\![\boldsymbol{p'}_{\pi(i)}]\!]\}_{i \in [1,N]}$. Then, the CS interacts with the TA to perform comparisons so that the TA obtains $index$, which is the permuted class index $\pi(i)$ with the highest p'_i, but without revealing any information about the client's data or the classification model to the CS or TA from the intermediate results.

Our comparison protocol, which is based on Wood et al.'s work [3], compares two values at a time. For simplicity, we denote the two values to be compared as $[\![\boldsymbol{a}]\!]$ and $[\![\boldsymbol{b}]\!]$, which contain value a and b, respectively, in all the slots. Let $\mathcal{G}$ denote a set of linear polynomials of the form $g(x) = Ax$ where A is a positive integer. As g is a linear polynomial with positive coefficient, we obtain $g(a) - g(b) \geq 0$ when $a \geq b$ and $g(a) - g(b) < 0$ otherwise. It is crucial that g is randomly chosen from sufficiently large range of polynomials in $\mathcal{G}$ while ensuring that $g(x)$ does not exceed the plaintext space.

In our protocol, the CS randomly choose $g \in \mathcal{G}$ at each comparison and computes $[\![\boldsymbol{h}]\!] = [\![g(\boldsymbol{a})]\!] \ominus [\![g(\boldsymbol{b})]\!] = g([\![\boldsymbol{a}]\!]) \ominus g([\![\boldsymbol{b}]\!])$, where $\ominus$ is a homomorphic subtraction. Since $g(x)$ is the same as a scalar multiplication, $g([\![\boldsymbol{a}]\!]) = [\![g(\boldsymbol{a})]\!]$. The CS sends $[\![\boldsymbol{h}]\!]$ to the TA who decrypts it to obtain h. If $h \geq 0$, the TA updates $index$ and generates a ciphertext $[\![\boldsymbol{d}]\!] = [\![\boldsymbol{1}]\!]$, which is an encryption of a vector whose values are all 1. Otherwise, the TA does not update $index$ and generates $[\![\boldsymbol{d}]\!] = [\![\boldsymbol{0}]\!]$, which is an encryption of a vector whose values are all 0. The TA sends $[\![\boldsymbol{d}]\!]$ to the CS, who then computes $([\![\boldsymbol{d}]\!] \otimes [\![\boldsymbol{a}]\!]) \oplus ((1 \ominus [\![\boldsymbol{d}]\!]) \otimes [\![\boldsymbol{b}]\!])$. The resulting ciphertext is a ciphertext of the higher value, which will be used at the next comparison. The comparison is repeated until we finish comparing all values. In the protocol, the TA does not learn which class is the actual classification result from $index$ because it does not know π. Also, as g is different for every comparison, the TA will not be able to learn anything about the classification model or client's data from h. Once all comparisons are made, the TA sends $index$ to the client, who reverses the permutation to obtain the actual index i and the classification result c_i.

Algorithm 1: Classification protocol

```
Client's inputs: encrypted data $[\![\boldsymbol{x}]\!]$, permutation vector $\pi$
CS's inputs: encrypted model $[\![\boldsymbol{v}]\!] = \{[\![\boldsymbol{v}_{c_1}]\!], [\![\boldsymbol{v}_{c_2}]\!], \dots, [\![\boldsymbol{v}_{c_N}]\!]\}$, a set of polynomials $\mathcal{G}$
TA's inputs: secret key $sk$, public key $pk$
1  CS:
2    for $i$=1 to $N$ do
3      $[\![\boldsymbol{temp}]\!] \leftarrow [\![\boldsymbol{x}]\!] \otimes [\![\boldsymbol{v}_{c_i}]\!]$          ▷extract matching probabilities
4      $[\![\boldsymbol{p}'_i]\!] \leftarrow \mathrm{TotalSums}([\![\boldsymbol{temp}]\!])$
5    end for
6    Permute ciphertexts as $\{[\![\boldsymbol{p}'_{\pi(i)}]\!]\}_{i \in [1,N]}$
7  TA:
8    $index \leftarrow 1$
9  CS:
10   $[\![\boldsymbol{max}]\!] \leftarrow [\![\boldsymbol{p}'_{\pi(1)}]\!]$
11 for $i$=2 to $N$ do
12   CS:
13     Randomly choose $g \in \mathcal{G}$
14     $[\![\boldsymbol{h}]\!] \leftarrow g([\![\boldsymbol{p}'_{\pi(i)}]\!]) \ominus g([\![\boldsymbol{max}]\!])$
15     Sends $[\![\boldsymbol{h}]\!]$ to TA
16   TA:
17     $h \leftarrow \mathrm{Dec}_{sk}([\![\boldsymbol{h}]\!])$          ▷all slots of $[\![\boldsymbol{h}]\!]$ are value $h$
18     if $h \geq 0$: $d \leftarrow 1$, $index \leftarrow i$
19     if $h < 0$: $d \leftarrow 0$
20     $[\![\boldsymbol{d}]\!] \leftarrow \mathrm{Enc}_{pk}(d)$
21     Sends $[\![\boldsymbol{d}]\!]$ to CS
22   CS:
23     $[\![\boldsymbol{max}]\!] \leftarrow ([\![\boldsymbol{d}]\!] \otimes [\![\boldsymbol{p}'_{\pi(i)}]\!]) \oplus ((1 \ominus [\![\boldsymbol{d}]\!]) \otimes [\![\boldsymbol{max}]\!])$
24 end for
25 TA:
26   Sends $index$ to client
27 Client:
28   $i \leftarrow \pi^{-1}(index)$
29   Output class $c_i$ as the classification result
```

Algorithm 2: TotalSums(u) [25]

```
Input: FHE ciphertext $u$
1  $v \leftarrow u, e \leftarrow 1, n \leftarrow$ number of slots in $u$
2  $k \leftarrow$ number of bits in $n$                    ▷if $n = 5, k = 3$
3  for $i$=$k$-2 down to 0 do
4    $v \leftarrow v \oplus (v \gg e)$
5    $e \leftarrow 2e$
6    $b \leftarrow bit_j(n)$                              ▷$j$-th bit of $n$ where bit 0 is the LSB
7    if $b$=1:
8      $v \leftarrow u \oplus (v \gg e)$
9      $e \leftarrow e + 1$
10   end if
11 end for
12 Return $v$ as the result
```

4.4. Security Intuition

We provide an intuitive security analysis of our protocol. In our protocol, a client sends his/her encrypted data and a permutation vector to the CS as a query. The CS calculates the classification probability $[\![p'_i]\!]$ over FHE using the encrypted classification model and encrypted data. Because $[\![p'_i]\!]$ is calculated over FHE, the CS does not learn anything. When comparing values to find the class index with the highest p'_i, the CS first permutes the vector of ciphertexts that encrypts the classification probabilities, then sends to the TA the difference between two encrypted values that is blinded by g, which is randomly chosen from $\mathcal{G}$ at every comparison. As the difference is blinded, the TA does not learn anything about the values that are being compared or their difference when it decrypts the ciphertexts. Based on the decrypted result, the TA updates $index$ to the permuted class index $\pi(i)$ with the highest p'_i and generates a new ciphertext. Although the TA knows $index$, it does not learn the actual class index as the TA does not know π and thus cannot reverse the permutation. The TA sends the new ciphertext to the CS who performs computation to obtain a ciphertext that encrypts the higher value, but the CS does not learn anything because the computation is performed over FHE. At the end of all comparisons, the TA sends $index$ to the client, who reverses the permutation to obtain the final classification result. As the client receives only the class index, it does not learn anything about the classification model that was used to classify the data.

Thus, no participating parties learn anything about the classification model, client's data, or classification result during classification, and the client learns only the classification result and does not learn anything about the classification model that was used to classify the data.

5. Experimental Evaluation

In this experimental evaluation, we evaluate the performance of our protocol by measuring its computation time and communication cost. To assess the communication cost, we measured the size of the ciphertexts that are transferred throughout the entire system instead of measuring the communication time because communication time depends on the network bandwidth.

We implemented and evaluated only our protocol because Li et al.'s work [15], which is the only work that achieves the ideal scenario, does not have a concrete classification method and involves heavy proxy re-encryption, which is based on the bootstrapping technique by Gentry [1] that lasts from a few seconds to a few minutes [16].

We compare the performance of our classification protocol for the following two cases: i) the case in which the number of feature values is fixed but the number of classes vary, and ii) the opposite case in which the number of classes is fixed but the number of feature values vary.

5.1. Data Sets and Experimental Environment

In our experimental evaluation, we used the Breast Cancer Wisconsin (Original) data set (2 classes with 9 feature attributes) and the Car Evaluation data set (4 classes with 6 feature attributes) from the UCI machine learning repository [26]. The Breast Cancer data set has 9 feature attributes with 10 feature values each and the Car Evaluation data set has 6 feature attributes with 21 distinct feature values in total. As client's data will be represented as a feature vector of length $f + 1$, where f is the number of feature values, the length of the feature vector is 91 and 22 for the Breast Cancer data set and the Car Evaluation data set, respectively. We modified the Car Evaluation data set to evaluate the performance for two-class and three-class classification by eliminating some classes from the data set as needed. The classification models for the two data sets used in our experiment were trained on plaintext data. Then, the class and conditional probabilities were represented as integers in the range of 0 to 79 by scaling and rounding after taking the logarithm of the probabilities. From Equation (2), the highest possible classification probability p'_{max} for the Breast Cancer data set and the Car Evaluation data set is 790 and 533, respectively. We summarize the above information in Table 2.

We implemented our proposed secure classification protocol using the FHE library HElib[1], which implements the BGV scheme using C++. In our experiment, we used the parameters listed in Table 3, where the parameters $m, p, r, \log q$ are the FHE system parameters $param$ and λ is the security parameter. The parameters were chosen so that sufficiently large range of g can be randomly chosen from $\mathcal{G}$. With these parameters, a ciphertext have 218 slots available, which is a sufficient number of slots to encrypt a feature vector for both data sets.

[1] https://github.com/homenc/HElib

The experiment was run on 64-bit Ubuntu 16.04 LTS, Intel Core i7-4770 CPU @ 3.40 GHz x 4, with 23.5 GiB memory using a single thread. Although our system scenario involves a model provider, client, CS, and TA, the same machine was used for all parties.

Table 2. Experimental Data Sets

Data Set	Number of classes	Number of feature attributes	Number of feature values	Length of feature vector	p'_{max}
Breast Cancer	2	9	90	91	790
Car Evaluation	2, 3, 4	6	21	22	533

Table 3. FHE Parameters

m	p	r	$\log q$	λ
11,119	2	18	180	119

5.2. Experimental Results

In this section, we present the experimental results of our classification protocol. Each experimental result is the average of 20 trials.

Before we present the experimental results of our classification protocol, we first present the computation times for the basic FHE algorithms: KeyGen, Enc, and Dec. We measured the time it takes to set up the system by generating the public key and secret key pair, to encrypt a single vector, and to decrypt a ciphertext during comparison. We also measured the size of a newly encrypted ciphertext. The results are presented in Table 4. As the results show, the KeyGen algorithm takes approximately 40 seconds, but this algorithm needs to be run only once at the initial setup. Thereafter, the encryption and decryption of a single vector requires only several milliseconds.

In our protocol, we encrypt the class probability and all conditional probabilities for class c_i as a single vector. Thus, the computation time required for the model provider to encrypt the classification model and the communication cost to upload the encrypted model increase linearly with respect to the number of classes *N*.

Next, we present the computation time and the communication cost for classifying a single feature vector of data using the Breast Cancer Wisconsin (Original) data set. We also present the result for two-class, three-class, and four-class classifiers using the Car Evaluation data set, which has been modified as described previously.

In this experiment, we measured the computation time for the whole protocol, from the client encrypting a feature vector and generating a permutation vector to the client obtaining the classification result by reversing the permutation on the result returned from the TA. To assess the communication cost, we measured the total size of the ciphertexts that were transferred throughout the entire system from start to finish. More specifically, we measured the communication cost between the client and CS, and between the CS and TA where most of the communication takes place. We omit the communication cost between the TA and client because only a plaintext of a few bytes will be sent from the TA to the client.

The experimental results are summarized in Table 5 and Fig. 2. In Fig. 2, we show only the experimental results for the Car Evaluation data set. As we can see, the computation time and communication cost increase linearly with respect to the number of classes. We can also see that the experimental results for both two-class classification data sets are very similar, demonstrating that the number of feature attributes or feature values does not affect the results of our protocol. In addition, the communication cost between the client and TA is the same regardless of the number of classes because the client only needs to send a single ciphertext and a plaintext permutation vector.

In fact, the burden on the client is very small as the client only needs to encrypt his/her data and generate a permutation vector that is sent as a plaintext. Because the client obtains his/her classification result as a plaintext, the burden on the client to classify data is just a single encryption operation and sending of the ciphertext and permutation vector, which is 0.016 s computation time and 1.5 MB communication cost.

To provide more insight, we measured the computation time for calculating p'_i for a single class c_i and obtained a computation time of 0.695 s. Because we calculate p'_i for each class, the total computation time for calculating p'_i for all classes is $O(N)$, where N is the number of classes. This consumes more than half of the computation time of the entire protocol. We also measured the computation time and communication cost for comparing two values and obtained 0.144 s and 4.0 MB, respectively. We can see that the computation time for comparison is smaller than the computation time for calculating p'_i. The total computation time and communication cost for obtaining the class with the highest p'_i is $O(N)$, where N is the number of classes. Also, we can see that most of the communication cost of the entire protocol occurs during these comparisons, which is between the CS and TA.

Lastly, we note that the experimental results do not include the communication time that will exist in a real-world system. Instead, as mentioned previously, we measured the total size of the ciphertexts that would be transferred throughout the system to determine the communication cost because the communication time depends on the network bandwidth. Thus, in a real-world system, our protocol requires additional time for classification. However, the additional communication time should not be a concern because the size of the data transferred from the client to the CS is not large (about 1.5 MB) and we can assume that the network bandwidth between the CS and TA will be large.

Our experimental results show that our protocol runs in 1.6 s with 4.0 MB of communication cost for classifying a data using a two-class classifier, increasing linearly with respect to the number of classes, while preserving the privacy of the classification model, client's data, and classification results. The burden on the client is very low because the protocol requires only a single encryption of his/her data (about 0.016 s) and the generation of the permutation vector (a few milliseconds), which is sent to the CS as a query of about 1.5 MB in size, and the client receives the classification result as a plaintext. The computation time and communication cost at the CS and TA increase in comparison, but this will not be a problem because such servers have large computational resources and large network bandwidth.

Table 4. KeyGen, Encryption, and Decryption Times

Time (s)			Ciphertext Size (MB)
KeyGen	Enc	Dec	
38.577	0.016	0.008	1.5

Table 5. Computation Time and Communication Cost

Data Set	Number of classes	Time (s)	Transferred Data Size (MB)		
			Between Client and CS	Between CS and TA	Total
Breast Cancer	2	1.549	1.5	2.5	4.0
Car Evaluation	2	1.551		2.5	4.0
	3	2.420		6.5	8.0
	4	3.283		10.5	12.0

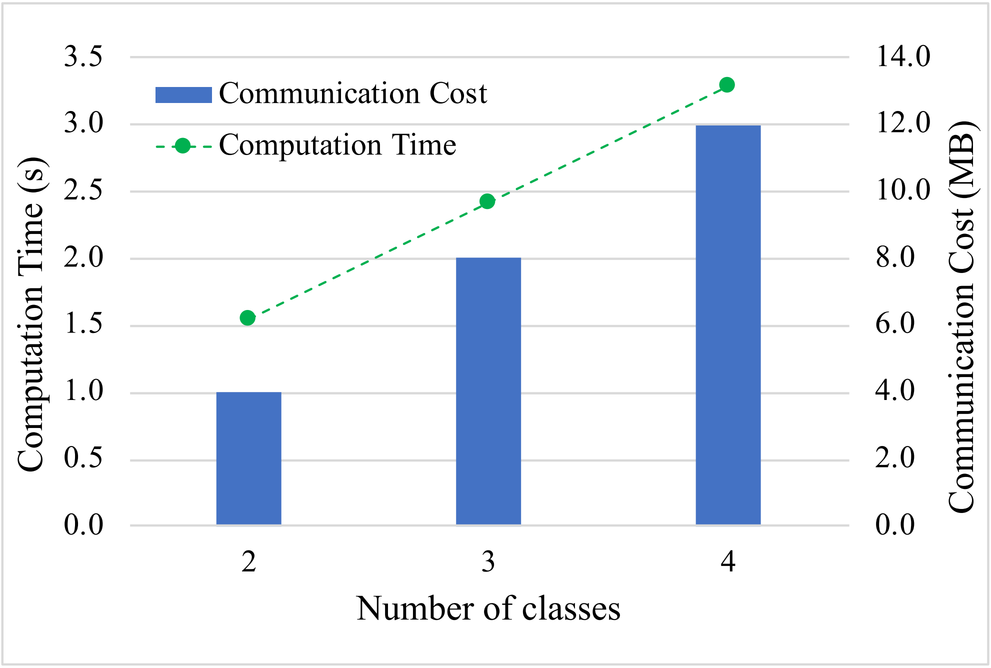


Fig. 2. Computation Time and Communication Cost for Car Evaluation Data Set

5.3. Discussion

In this experimental evaluation, we implemented and evaluated only our protocol's performance because Li et al.'s work [15] does not have a concrete classification method and involves heavy proxy re-encryption, which is based on the bootstrapping technique by Gentry [1] that lasts from a few seconds to a few minutes [16]. Their proxy re-encryption is performed once per data to re-encrypt the data under a common public key. However, the proxy re-encryption is only an initial step before performing the classification, whose method is abstracted in the paper. In fact, their proxy re-encryption is unnecessary as the clients can encrypt their data using the system's common public key instead of using their own set of keys. Thus, while their proxy re-encryption lasts from a few seconds to a few minutes in addition to the actual classification of data, our classification protocol completes in approximately 3.3 s for a four-class classifier. Thus, our proposed protocol has higher computational efficiency.

In our experiment, each of the probabilities in our classification model were represented as an integer in the range of 0 to 79. We note that it is possible to express each probability using a larger range of integers, such as 0 to 200, and still obtain the same computation time and communication cost as in our experimental results as the computation time and communication cost are independent of the values that are encrypted.

Similarly, we can classify data that has more number of feature attributes or feature values and obtain the same computation time and communication cost because we perform operations over all the slots within the ciphertext using the SIMD style operations. Thus, the number of operations, computation time, and communication cost is independent of the number of feature attributes or feature values within a data. In fact, we have already shown in our experimental results that the number of feature attributes or feature values within a data set does not affect the computation time and communication cost of our classification protocol.

It is important to note, however, that we must ensure that $g \in \mathcal{G}$ will be randomly chosen such that $g(p'_{max})$, where p'_{max} is the highest possible classification probability, does not exceed the plaintext space. Also, if the number of feature values exceeds the number of slots within a ciphertext, some modifications of the protocol will be needed, such as using two ciphertexts to represent the feature vector. In the above two cases, we could also choose a different set of FHE parameters to increase the plaintext space and/or the number of slots, which may require a longer computation time and higher communication cost depending on the parameters.

6. Optimization of the Proposed Protocol

The proposed classification protocol and the experimental results, shown in Section 4 and 5 respectively, can be regarded as the basic protocol and its performance for classifying a single data without any optimization for given data sets. Thus, the performance of the proposed protocol is the same for any data set; independent from the way the probabilities are represented or the length of a feature vector.

In this section, we present some techniques that can be applied to our proposed protocol so that the computation time can be optimized for each data set. In Section 6.1, we present some techniques for optimizing the computation time when classifying a single data. In Section 6.2, we present a technique for classifying multiple data simultaneously using a single ciphertext by extending the techniques from Section 6.1. In Section 6.3, we indicate a part in the proposed protocol that can be run in parallel.

6.1. Optimization for single data classification

In this section, we present a technique that can be applied to our proposed protocol in order to optimize the computation time when classifying a single data.

In Section 6.1.1, we describe the technique for optimizing the computation time by modifying the TotalSums operation. In Section 6.1.2, we present additional modification to the classification protocol that is necessary to ensure the privacy of intermediate results. In Section 6.1.3, we present the expected change in computation time between the unmodified and modified protocol.

6.1.1. Optimization of the TotalSums operation

In Section 5.3, we have discussed that the computation time and communication cost of our proposed protocol is independent from the number of feature attributes or feature values within a data set. This is because we perform the TotalSums operation on *all* the slots within a ciphertext when calculating the classification probability p'. However, to calculate p', we only need to sum s slots within the ciphertext, where s is the length of the client's feature vector. Thus, by summing only s slots within the ciphertext, we can reduce the computation time during classification. The required modification of the TotalSums operation, whose original operation is shown in Algorithm 2, is very small since we only need to initialize n with the length s of a feature vector instead of initializing with the number of slots in a ciphertext.

Table 6 shows the computation time for performing TotalSums on 218 slots (all slots in a ciphertext), 91 slots (length of a feature vector for the Breast Cancer data set), and 22 slots (length of a feature vector for the Car Evaluation data set). As we can see from the result, the computation time decreases with respect to the number of slots. The decrease in computation time is not linear with respect to the number of slots because the TotalSums operation runs in $O(\log s)$ where s is the number of slots we perform the summation on. As we sum the values within a ciphertext to calculate p' for each class c_i, the total expected decrease in computation time during classification is linear to the number of classes.

Table 6. Computation time of unmodified and modified TotalSums

Number of slots	Unmodified	Modified	
	218 slots	91 slots	22 slots
Time (s)	0.622	0.474	0.234

It is important to note, however, that reduction in the number of slots s which we sum does not necessary improve the computation time. As it can be seen from Fig. 3, the computation time could largely change between two consecutive values. This is mainly because the number of operations in TotalSums depends on the number of bit 1 in s (ex. $91 = 1011011_2$). Thus, it is possible to further optimize the computation time by adding dummy feature values to the feature vector in order to adjust s for more optimal computation time or by using the unmodified TotalSums operation. Concretely, we can choose an optimal number of slots s_{new} such that $s \leq s_{new} \leq s + w$, where w is a margin for adjusting s, to optimize the computation time. An example for when w=8 is shown in Fig. 4. In this case, we can improve the computation time by up to 1.45 times compared to that of s without any adjustment.

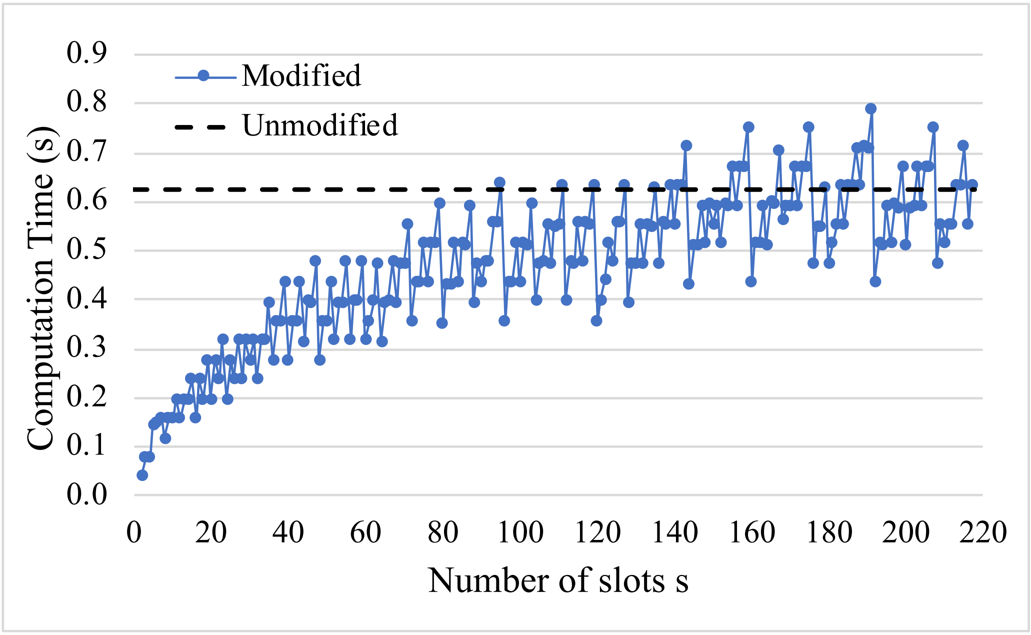


Fig. 3. Computation time of TotalSums for different number of slots

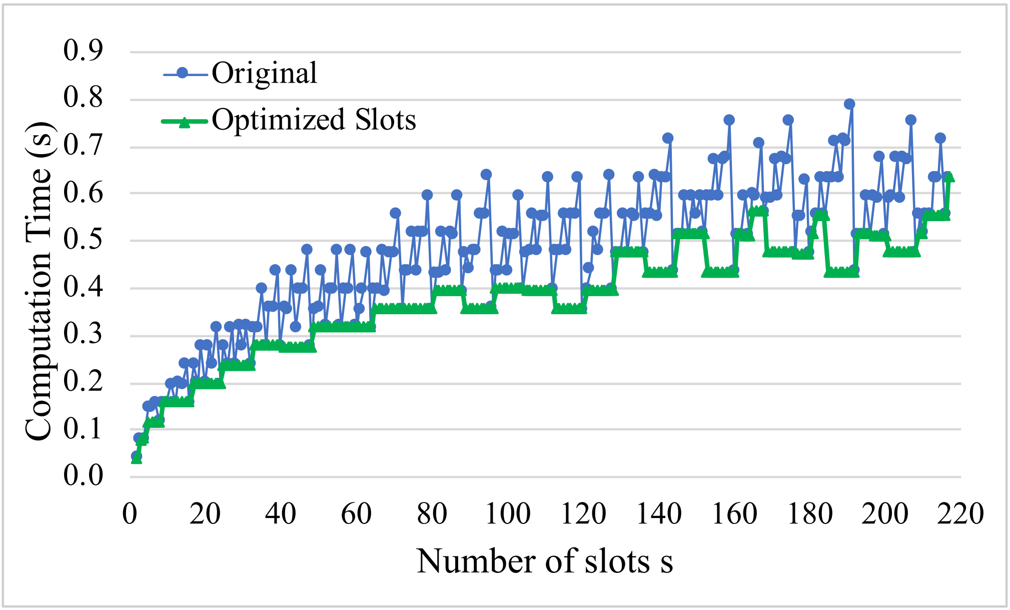
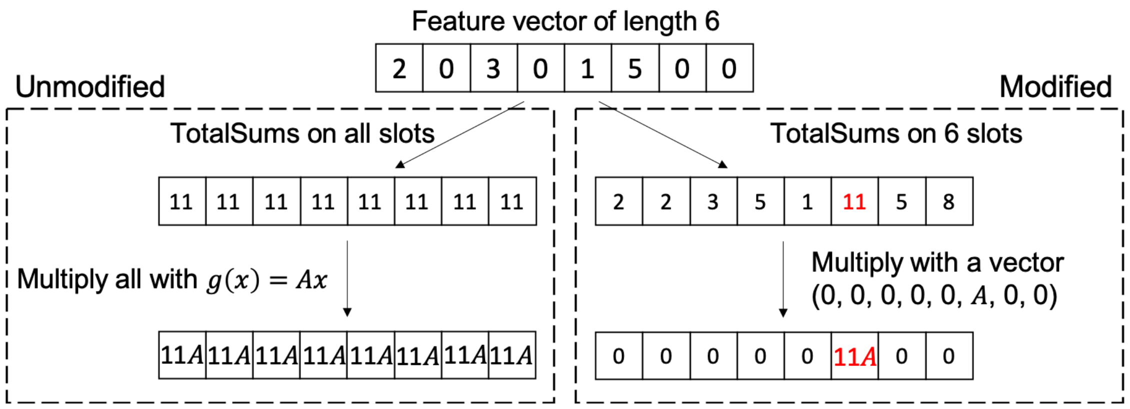


Fig. 4. Computation time of TotalSums after adjusting the number of slots

6.1.2. Modification to the classification protocol

To securely perform classification without revealing any data to any parties, we require one additional modification to the classification protocol. During the comparisons in our proposed protocol, the CS randomly chooses $g \in \mathcal{G}$, calculates $[\![\boldsymbol{h}]\!] = g([\![\boldsymbol{a}]\!]) \ominus g([\![\boldsymbol{b}]\!])$, then sends $[\![\boldsymbol{h}]\!]$ to the TA. The TA decrypts $[\![\boldsymbol{h}]\!]$ to obtain h and continues the protocol. When TotalSums operation is performed on all slots in a ciphertext, the resulting ciphertext have the summed value in all the slots of a ciphertext. Thus, it was ensured that $[\![\boldsymbol{h}]\!]$ will have value h in all the slots. However, when we modify the TotalSums operation to perform summation on s slots, only one slot within the ciphertext have the value h while other slots contain partial summations. If we continue the protocol with only the modified TotalSums operation and apply g to all slots within a ciphertext, the function g will be revealed to the TA from the values when it is decrypted. If g is revealed, the TA will be able to obtain some information on the classification model and client's data from the decrypted result. Thus, it becomes necessary to mask all the other slots by multiplying the slots with 0. As such, instead of multiplying all slots with g, we multiply the ciphertext with a one-hot vector in which the position corresponding to where the summed value is located is set to g and all other positions are set to 0. The position for setting g can be easily determined from the data set. Thus, the resulting ciphertext $[\![\boldsymbol{h}]\!]$ will also be a one-hot vector where one slot is h and all other slots are 0. Upon receiving and decrypting $[\![\boldsymbol{h}]\!]$ to obtain h, the TA continues the same protocol as described in Section 4.

Fig. 5 shows an example of the resulting ciphertext and the needed operations after the TotalSums operation for both the unmodified and modified cases.



Feature vector of length 6
2 0 3 0 1 5 0 0
Unmodified
Modified
TotalSums on all slots
11 11 11 11 11 11 11 11
Multiply all with $g(x) = Ax$
$11A$ $11A$ $11A$ $11A$ $11A$ $11A$ $11A$ $11A$
TotalSums on 6 slots
2 2 3 5 1 11 5 8
Multiply with a vector (0, 0, 0, 0, 0, A, 0, 0)
0 0 0 0 0 $11A$ 0 0

Fig. 5. Difference between the unmodified and modified TotalSums operation

In FHE, a multiplication operation between a ciphertext and a plaintext vector containing non-uniform values is computationally more expensive than a multiplication between a ciphertext and a plaintext vector that contains uniform values. Thus, the computation time for each comparison of the modified protocol becomes more expensive than that of the unmodified protocol. Table 7 shows the computation time for comparing two values after performing TotalSums on 218 slots (all slots in a ciphertext), 91 slots (length of a feature vector for the Breast Cancer data set), and 22 slots (length of a feature vector for the Car Evaluation data set). We can see from the result that, indeed, the computation time for each comparison has increased for both data sets after the modification of the protocol. Also, we can see that the time for comparison is different between the case for 91 slots and 22 slots. This is due to the characteristics of FHE that the computation time is affected by prior operations performed on the ciphertext. In Fig. 6, we show the computation time for comparing two values for when we perform the TotalSums operation on different number of slots. From the figure, we can see that although some variations occur, the computation time is mostly constant.

Table 7. Computation time for comparing two values

Number of slots	Unmodified	Modified	
	218 slots	91 slots	22 slots
Time (s)	0.143	0.365	0.333

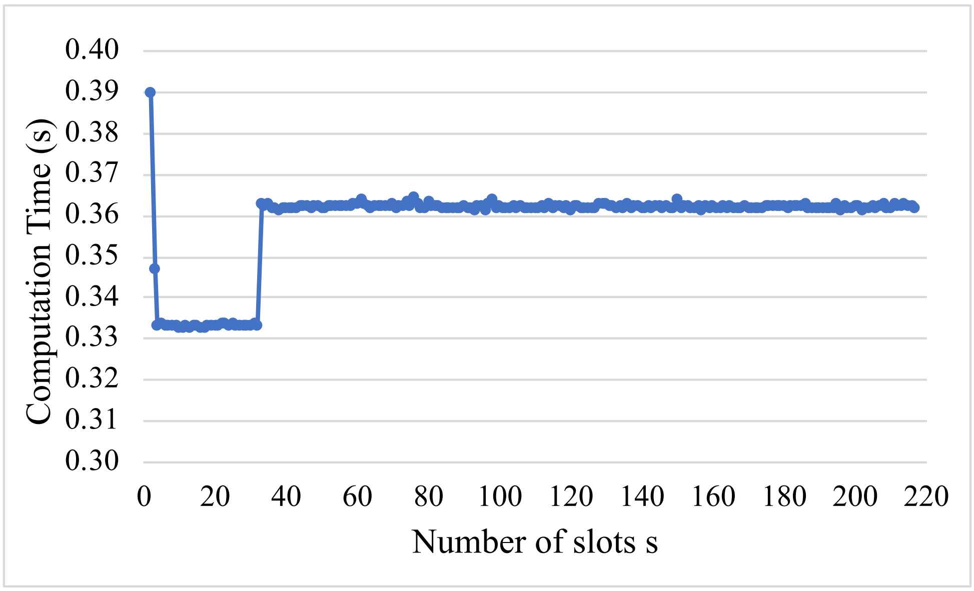


Fig. 6. Computation time of comparing two values for different number of slots

6.1.3. Change in computation time

In Table 8, we show the total computation time of the classification protocol after the two modifications presented in Section 6.1.1 and 6.1.2. From the results, we can see that the modified classification protocol has better computation time compared to that of the unmodified protocol. Also, while the unmodified protocol has same computation time for both two-class classifiers, the modified protocol now has different computation time because the TotalSums operation is now modified for each data set. We note that there is no change in the communication cost as the number and the size of ciphertexts that is transferred throughout the system does not change from these modifications.

Table 8. Computation time of the modified classification protocol

Data set	Length of feature vector	Number of classes	Time (s)	
			Unmodified	Modified
Breast Cancer	91	2	1.549	1.458
Car Evaluation	22	2	1.551	0.962
		3	2.420	1.601
		4	3.283	2.240

We note that although we can optimize the classification protocol by modifying the TotalSums operation and the comparison protocol, there may be cases where the unmodified protocol has better performance than the modified protocol. The total expected change in classification time between unmodified and modified protocol is $\Delta Classification = (\Delta Compare * (N-1)) - (\Delta TotalSum * N)$, where N is the number of classes. When $\Delta Classification$ is positive, the computation time increases compared to the unmodified protocol. In Fig. 7, we show the total estimated change in classification time $\Delta Classification$ when $N = 2$ for different number of slots s which we perform the $TotalSum$ on. As we can see from the figure, for larger value of s, there are several instances in which $\Delta Classification$ is positive, indicating that the computation time increases compared to the unmodified protocol. In such cases, using the unmodified protocol will be more efficient when classifying a single data.

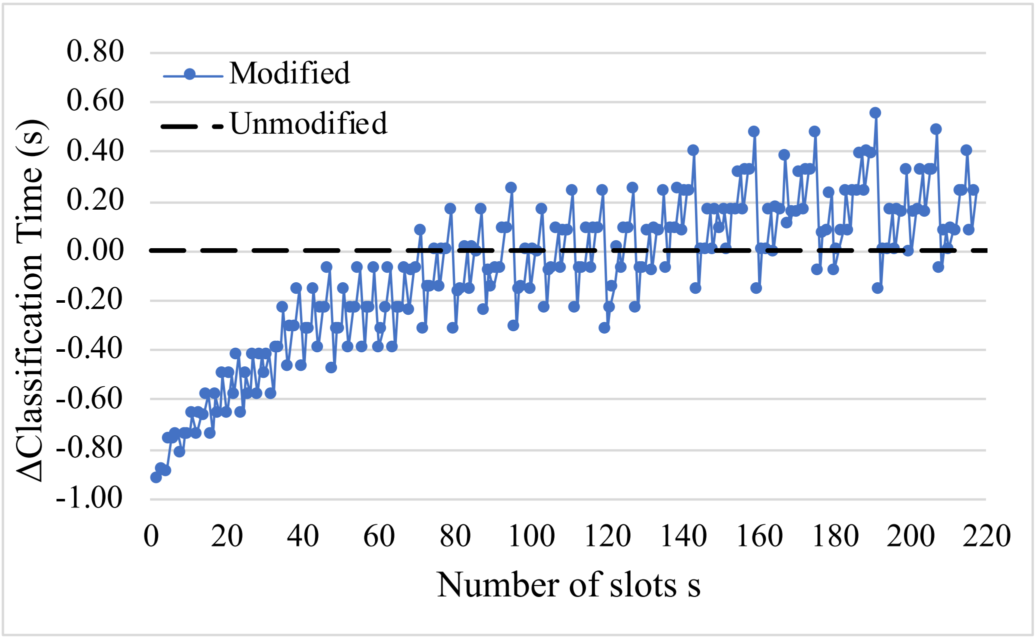


Fig. 7. Estimated change in classification time for different number of slots when N=2

6.2. Classification of Multiple Data

By extending the modifications presented in Section 6.1, we can classify multiple data simultaneously using a single ciphertext. In the modified protocol, we perform summation on only s slots in a ciphertext, where s is the length of a feature vector, then proceed with the protocol by focusing on the slot that contains the classification probability p'. Thus, if there are sufficient slots, we can pack multiple feature vectors into a single ciphertext and perform classification on multiple data simultaneously using a single ciphertext. By performing the modified TotalSums operation on s slots, we can calculate p' for each encrypted feature vector at the same time. Then, at comparison, the CS chooses different $g \in \mathcal{G}$ for each data to be compared and masks all other slots so that it becomes 0, similar to what we described in Section 6.1. The TA continues the protocol, but now managing $index$ for each data and $[\![\boldsymbol{d}]\!]$ is now a vector whose slots corresponding to each data are set to 0 or 1 depending on the decrypted value h for that data and the rest are set to 0. No additional modifications are necessary to classify multiple data using a single ciphertext.

Table 9 shows the experimental results of the modified classification protocol for classifying multiple data using a single ciphertext. The number of data per ciphertext, which we denote as $numData$, is calculated as $numData = \lfloor s/t \rfloor$ where s is the length of a feature vector and t is the total number of slots within a ciphertext, which is 218 in our experiment. As we can see from Table 9, the computation time is not largely different from the experimental results shown in Table 8 when we are classifying a single data per ciphertext but with the optimized summation. The computation time for classifying multiple data is slightly longer because the TA now updates $index$ and set value for a slot within $[\![\boldsymbol{d}]\!]$ by checking the decrypted value for each data. Of course, as we are now classifying multiple data at once, the throughput of the classification protocol has increased. Although we packed 2 and 9 data per ciphertext for the Breast Cancer data set and Car Evaluation data set, respectively, it is possible to pack fewer data into a ciphertext at classification as well.

We finally note that in Fig. 7, we have shown that the computation time for classifying a single data with the modified protocol may be longer than that of the unmodified protocol depending on the number of slots. However, in cases in which we can pack multiple data into a single ciphertext, the total computation time may increase but the throughput of the modified protocol may be higher compared to the unmodified protocol.

Table 9. Computation time of the modified classification protocol

Data set	Length of a feature vector	Number of data per ciphertext	Number of classes	Time (s)	Time (s) per data
Breast Cancer	91	1	2	1.458	1.458
		2		1.464	0.732
Car Evaluation	22	1	2	0.962	0.962
			3	1.601	1.601
			4	2.240	2.240
		9	2	0.964	0.107
			3	1.615	0.179
			4	2.268	0.252

6.3. Parallelization of Comparison

In our classification protocol, the CS and TA perform the comparisons one at a time, updating $[\![\boldsymbol{max}]\!]$ to a ciphertext of the higher value over FHE. We can optimize this step by running multiple comparisons in parallel so that the comparisons occur in a tree-like structure rather than linearly. Although the total number of comparisons that will be made is the same, we can reduce the computation time as they will be running in parallel.

7. Conclusion

In this paper, we proposed a secure naïve Bayes classification protocol over encrypted data using FHE. Our proposed protocol allows a model provider to provide a classification model and classification service while preserving the privacy of the classification model from all participating parties and allows clients to outsource data classification to a cloud server while preserving the privacy of their data and classification result from the participating entities. The client learns only the classification result and does not learn anything about the classification model that was used to classify the data.

Our experimental results show that our protocol runs in 1.5 s with 4.0 MB of communication cost for classifying a data using a two-class classifier while preserving the privacy of the classification model, client's data, and results. Our experimental results also show that the computation time and communication cost increases linearly with respect to the number of classes. The burden on the client at classification is very small, making the utmost use of the cloud servers by outsourcing the classification tasks. In addition, we indicated several techniques that could be used to modify our protocol in order to optimize the computation time for a given data set such as optimization of the TotalSums operation, and extension of it to enable classification of multiple data simultaneously. By doing so, we reduced the computation time from 1.5 s to 1.0 s for a two-class classification of the Car Evaluation data set while classifying 9 data at once. Although we focused on naïve Bayes in this work, our protocol can be easily modified for other machine learning algorithms.

Potential future works are as follows: i) measure the real-world communication cost and time with varying network bandwidth to evaluate the performance of the proposed classification protocol, and ii) apply the proposed classification protocol to other machine learning algorithms. Also, choosing the optimal FHE parameters for a given dataset is an open problem.

Acknowledgement

Upon completion of this paper, I would like to thank Professor Yamana who has given me many advices and support towards completion of this thesis. I would also like to thank all the lab members, especially members of SC group, for various advices and discussions throughout the years.

This work was supported by JST CREST Grant Number JPMJCR1503, Japan.

Reference

[1] Gentry, C.：Fully homomorphic encryption using ideal lattices. In: Proc. of the Annual ACM Symposium on Theory of Computing (STOC ’09), pp. 169–178 (2009).

[2] Bost, R., Popa R.A., Tu, S., Goldwasser, S.: Machine learning classification over encrypted data. In: Network and Distributed System Security Symposium (NDSS), pp. 1–14 (2015).

[3] Wood, A., Shpilrain, V., Najarian, K., Mostashari, A., Kahrobaei, D.: Private-Key Fully Homomorphic Encryption for Private Classification. In: International Congress on Mathematical Software (ICMS 2018), pp. 475–481 (2018).

[4] Sun, S., Zhang, P., Liu, J.K., Yu, J., Xie, W.: Private machine learning classification based on fully homomorphic encryption. In: IEEE Transactions on Emerging Topics in Computing (Early Access), pp. 1–13 (2018).

[5] Khedr, A., Gulak, G., Vaikuntanathan, V.: SHIELD: scalable homomorphic implementation of encrypted data-classifiers. In: IEEE Transactions on Computers, vol. 65, Issue 9, pp. 2848–2858 (2016).

[6] Dowlin, N., Gilad-Bachrach, R., Laine, K., Lauter, K., Naehrig, M., Wernsing, J.: CryptoNets: Applying neural networks to encrypted data with high throughput and accuracy. In: International Conference on Machine Learning, pp. 201–210 (2016).

[7] Park, H., Kim, P., Kim, H., Park, K.W., Lee, Y.: Efficient machine learning over encrypted data with non-interactive communication. In: Computer Standards & Interfaces, vol. 58, pp. 87–108 (2017).

[8] Aslett, L.J.M., Esperança, P.M., Holmes, C.C.: Encrypted statistical machine learning: new privacy preserving methods. In: arXiv preprint arXiv:1508.06845 (2015).

[9] Graepel, T., Lauter, K., Naehrig, M.: ML Confidential: Machine Learning on Encrypted Data. In: Information Security and Cryptology – ICISC 2012, LNCS, vol. 4586, pp.1–21 (2012).

[10] Chabanne, H., Wargny, A., Milgram, J.: Privacy-preserving classification on deep neural network. Cryptology ePrint Archive, Report 2017/035 (2017).

[11] Hesamifard, E., Takabi, H., Ghasemi, M., Jones, C.: Privacy-preserving Machine Learning in Cloud. In: Proc. of the 2017 on Cloud Computing Security Workshop (CCSW ’17), pp. 39–43 (2017).

[12] Chen H., Gilad-Bachrach, R., Han, K., Huang, Z., Jalali, A., Laine, K., Lauter, K.: Logistic regression over encrypted data from fully homomorphic encryption. Cryptology ePrint Archive, Report 2018/462 (2018).

[13] Han, K., Hong, S., Cheon, J.H., Park, D.: Efficient Logistic Regression on Large Encrypted Data. Cryptology ePrint Archive, Report 2018/662 (2018).

[14] Kim, S., Omori, M., Hayashi, T., Omori, T., Wang, L., Ozawa, S.: Privacy-Preserving Naive Bayes Classification Using Fully Homomorphic Encryption. In: International Conference on Neural Information Processing (ICONIP 2018), pp. 349–358 (2018).

[15] Li, P., Li, J., Huang, Z., Gao, C., Chen, W., Chen, K.: Privacy-preserving outsourced classification in cloud computing. In: Cluster Computing, March 2018, vol. 21, Issue 1, pp. 277–286 (2018).

[16] Chen, H., Han, K.: Homomorphic Lower Digits Removal and Improved FHE Bootstrapping. Cryptology ePrint Archive, Report 2018/067 (2018).

[17] Rivest, R., Adleman, L., Dertouzos, M.: On data banks and privacy homomorphisms. In: Foundations of Secure Computation, pp. 169–180 (1978).

[18] Gentry, C., Sahai, A., Waters, B.: Homomorphic Encryption from Learning with Errors: Conceptually-Simpler, Asymptotically-Faster, Attribute-Based. In: Advances in Cryptology – CRYPTO 2013, LNCS, vol. 8042, pp. 75–92 (2013).

[19] Brakerski, Z., Gentry, C., Vaikuntanathan, V.: (Leveled) fully homomorphic encryption without bootstrapping. In: Proc. of the 3rd Innovations in Theoretical Computer Science Conference, pp. 309–325 (2012).

[20] Brakerski, Z.: Fully Homomorphic Encryption without Modulus Switching from Classical GapSVP. In: Advances in Cryptology – CRYPTO 2012, LNCS, vol. 7417, pp. 868–886 (2012).

[21] Fan, J., Vercauteren, F.: Somewhat Practical Fully Homomorphic Encryption. Cryptology ePrint Archive, Report 2012/144 (2012).

[22] Imabayashi, H., Ishimaki, Y., Umayabara, A., Sato, H., Yamana, H.: Secure Frequent Pattern Mining by Fully Homomorphic Encryption with Ciphertext Packing. In: Data Privacy Management and Security Assurance, LNCS, vol. 9963, pp. 181–195 (2016).

[23] Chen, H., Laine, K., Rindal, P.: Fast Private Set Intersection from Homomorphic Encryption. In: Proc. of the 2017 ACM SIGSAC Conference on Computer and Communications Security (CCS '17), pp. 1243–1255 (2017).

[24] Smart, N.P., Vercauteren, F.: Fully homomorphic SIMD operation. In: Designs, Codes and Cryptography, vol. 71, issue 1, pp. 57–81 (2014).

[25] Halevi, S., Shoup, V.: Algorithms in HElib. In: Advances in Cryptology – CRYPTO 2014, LNCS, vol. 8616, pp. 554–571 (2014).

[26] Dua, D., Graff, C.: UCI Machine Learning Repository. Tech. rep. (2019), http://archive.ics.uci.edu/ml

Publications

- **<u>Yoshiko Yasumura</u>**, Hiroki Imabayashi, and Hayato Yamana, "Attribute-based Proxy Re-encryption Method for Revocation in Cloud Storage", Proc. of 2017 IEEE Int'l Conf. on Big Data, pp. 4858–4860, (2017).
- **<u>Yoshiko Yasumura</u>**, Hiroki Imabayashi, and Hayato Yamana, "Attribute-based Proxy Re-encryption Method for Revocation in Cloud Storage: Reduction of Communication Cost at Re-encryption", Proc. of 2018 IEEE 3rd International Conference on Big Data Analysis (ICBDA), pp. 312–318, (2018).