Computer Standards & Interfaces 000 (2017) 1–22

Contents lists available at ScienceDirect

Computer Standards & Interfaces

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

Efficient machine learning over encrypted data with non-interactive communication[☆]

Heejin Park[a,1], Pyung Kim[b,1], Heeyoul Kim[c], Ki-Woong Park[d], Younho Lee[b,*]

[a] Division of Industrial and Information System Engineering, Graduate School of Policy & IT, SeoulTech, Republic of Korea
[b] ITM Division, Department of Industrial and Systems Engineering, SeoulTech, Republic of Korea
[c] Department of Computer Science, Kyonggi University, Korea
[d] Department of Computer and Information Security, Sejong University, Korea

ARTICLE INFO

Keywords:
Privacy-preserving classification
Fully homomorphic encryption
Applied cryptography
Security

ABSTRACT

In this paper, we describe a protocol framework that can perform classification tasks in a privacy-preserving manner. To demonstrate the feasibility of the proposed framework, we implement two protocols supporting Naive Bayes classification. We overcome the heavy computational load of conventional fully homomorphic encryption-based privacy-preserving protocols by using various optimization techniques. The proposed method differs from previous techniques insofar as it requires no intermediate interactions between the server and the client while executing the protocol, except for the mandatory interaction to obtain the decryption result of the encrypted classification output. As a result of this minimal interaction, the proposed method is relatively stable. Furthermore, the decryption key is used only once during the execution of the protocol, overcoming a potential security issue caused by the frequent exposure of the decryption key in memory. The proposed implementation uses a cryptographic primitive that is secure against attacks with quantum computers. Therefore, the framework described in this paper is expected to be robust against future quantum computer attacks.

1. Introduction

With the explosive increase in computing power over recent decades, machine learning methods that leverage this power have become extremely useful in various fields. Machine learning is now commonplace in applications such as spam classification, medical diagnosis, and credit evaluations. The broad field of data classification often relies on a two-step machine learning approach. The first is a training step that involves determining the parameter values for the classification algorithm. This step uses sample data (which we call ‘training data’) that have already been mapped to classification results. The second, prediction step classifies the data using the trained parameters (which we call the ‘model’). Consequently, the classification results for the data are obtained.

Classification can be used in many areas. For example, it can be used for services that predict future diseases and potential health risk factors based on survey results or medical history data. It is also possible to measure the value of assets based on a users financial status. In addition, users can assess their financial credit rating by providing certain information (e.g., by supplying their credit card balance and financial assets).

Many of these services require sensitive information from users. In the examples above, users must provide medical information to the classification algorithm in order to receive medical services. To predict credit ratings, sensitive financial information must be provided to the classification algorithm. The risk of such sensitive information being exposed to the public is one of the main obstacles to the wider application of such classification services. Thus, it is of vital importance to offer adequate protection to private and sensitive information.

Moreover, the security of the classification model itself is important. To obtain a high-quality model, a considerable amount of training data is required. Thus, the value of such models is very high. As a result, it is imprudent to offer general users any information regarding the model when providing the above services, because this privilege may be abused, i.e., by selling the information to other service providers.

Therefore, it is essential to perform data classification without exposing the models or the sensitive information used as input for the

[☆] This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIP) (NRF-2016R1C1B2011022,NRF-2016R1A4A1011761).

[*] Corresponding author.
E-mail addresses: rhaos@seoultech.ac.kr (H. Park), pkim0305@nslab.kaist.ac.kr (P. Kim), heeyoul.kim@kgu.ac.kr (H. Kim), woongbak@sejong.ac.kr (K.-W. Park), younholee@seoultech.ac.kr (Y. Lee).

[1] Co-first authors.

https://doi.org/10.1016/j.csi.2017.12.004
Received 12 July 2017; Received in revised form 8 December 2017; Accepted 8 December 2017
Available online xxx
0920-5489/© 2017 Elsevier B.V. All rights reserved.

classification algorithm. This problem is called the privacy-preserving data classification problem.

A definition of this problem was first articulated in [1], and some solutions were proposed in [2–4]. Specifically, in a classification service, there are two types of actors—the client serving as a user and the server providing the classification service. The server has a model w and a classification algorithm $C()$, and the user has a feature vector $\vec{x}$ denoting the data to be classified. The user sends a service request, which includes $\vec{x}$, to the server. The server computes $C(w, \vec{x})$ and returns the result to the user. During this process, the user should obtain only $C(w, \vec{x})$ and should not acquire any additional information about w. In the case of a server, no information about $\vec{x}$ and $C(w, \vec{x})$ should be known during the classification process.

Previous studies [1–4] have not been able to run $C()$ with the server alone without exposing $\vec{x}$, owing to the limitations of the cryptographic tools used. Instead, $C()$ can be computed by invoking a protocol between the server and the client. For this process, the necessary information must be shared securely between the client and the server. These methods require additional operations, which increase the computational cost and the number of communications over the entire system. This is a great burden for the following reasons.

The first is efficiency. To operate $C()$ with an interactive protocol consisting of multiple rounds, the server and client must store and manage the state of operations and communications. Moreover, there is a disadvantage in that a protocol that is composed of multiple rounds is difficult to implement reliably in a situation where the communication channel is unstable. In particular, in a situation where the server is faced with a large number of users, such as in a cloud environment, maintaining communication with all users creates a heavy burden on the server. Furthermore, the communication pattern between the server and client is different from the communication pattern that occurs in general web applications. Therefore, existing methods are unsuitable for web-based applications [5]. As web applications are popular and convenient among those using the Internet, this is a clear disadvantage.

The second reason is security. In previous methods, the client uses a private key during protocol execution for decryption operations. This means that the clients private key is frequently loaded into memory. This situation is undesirable because the memory contains data in plaintext form that can be easily exposed in the event of an external attack.

In this paper, we propose a new protocol to run a classification algorithm in a privacy-preserving manner with only a single round of communication. In the proposed protocol, if all necessary information is given, it is possible to perform $C()$ on either the server or client side without the help of the other party. Because no communication occurs during the $C()$ operation, the proposed protocol consists of simply passing the input of the $C()$ algorithm and returning the algorithm execution result, i.e., the classification result. Therefore, the protocol party who does not perform $C()$ can be offline while the other party is running $C()$. Thus, it is possible to perform the classification protocol using a communication method that does not require both the sender and the receiver to be online at the same time, much like e-mail. Furthermore, all information used to compute $C()$ is provided in the form of ciphertexts. Therefore, the protocol party who computes $C()$ cannot acquire any information about the inputs of $C()$ if they are supplied by another party to the protocol.

In addition, the proposed method only requires a private key during the decryption process, i.e., when acquiring the classification results of the user. This private key, then, is not required until the decryption process. Therefore, the private key information is only loaded into memory for a very short period of time, mitigating the security problem stemming from the memory residence of the private key.

To obtain the above advantages, the proposed protocol employs fully homomorphic encryption (FHE). With the operations supported by an FHE scheme, we can implement any algorithm that can be efficiently computed, such that it can run with any inputs that are encrypted using the encryption algorithm in the FHE scheme [6]. The output from running the implementation also takes the form of FHE ciphertext. The user then needs to decrypt this in order to obtain the result.

The process of the proposed protocol is as follows. First, the classification algorithm is implemented using the operations provided by the FHE method, such that it can operate when the FHE ciphertext input is given. The implementation results module is placed on the server. In addition, the model information is placed on the server in a special form, such that the server can perform the operations with FHE ciphertext-type inputs. A client wishing to use the classification service encrypts his/her input using a public key generated by the client. The client then sends this input information to the server, and the server carries out the classification with the encrypted input from the client and the model information. The execution results are delivered to the client, and the client obtains the classification results after decryption with his/her private key.

The proposed protocol is suitable for a cloud environment in which the server has high computing power, because classification is performed in the server. In addition, the proposed protocol cannot encrypt the model, owing to the nature of FHE. Therefore, if the cloud server is exposed to an external attack, the model information is likely to be captured by the attacker. To resolve this problem, we propose another classification protocol. In this protocol, the model information is encrypted with the server's key and delivered to the client prior to classification. When the classification is executed, the client uses the encrypted model and their own input, which is in plaintext form. The client then sends the results to the server. The server decrypts the results and returns them to the client. With this method, when the client sends the classification result to the server, it randomizes the classification result. Thus, the server cannot know the classification results after decryption. Hence, insofar as the model exists in an encrypted form, a pre-distribution of the model is possible before the protocol is executed. Therefore, the proposed method can be considered a single-round protocol.

In this study, we implement a Naive Bayes (NB) classifier, a machine learning tool used to predict the classes of various data (e.g., diseases in the Clinical Decision Support System [7]) using the operations in an FHE scheme [8–11]. The reason for choosing the NB classifier is that, despite its simplicity, it is known to be better suited to medical diagnosis than other sophisticated methods [12,13], and it has never been implemented exclusively with operations in an FHE scheme.

We realized the proposed protocols with the implemented privacy-preserving NB classifier. The main hurdle was the performance of the FHE operations: previous methods did not implement protocols exclusively with FHE because of the slow performance of FHE operations. In this paper, we describe how the performance can be improved using various techniques.

We implemented the proposed protocol using HELib [9–11]. The implementation environment was an Intel (R) Xeon (R) ES-1650 v3 @ hexa-core processor with 64GB RAM running Ubuntu 16.04 LTS as the server. We used an Intel i7-6700 3.40 GHz @ quad-core processor with 16GB RAM running Ubuntu 14.04 LTS as the client. After the implementation, we evaluated the classification performance using the actual data in [14]. The server required approximately 69 s to perform the classification and the client required approximately 90 s. These results confirm that the proposed method can be used in a practical environment requiring high security.

The remainder of this paper is organized as follows: Section 2 summarizes the related work. In Section 3, we introduce the NB classifier and FHE as preliminary information. Section 4 describes the system model and motivation for the proposed protocol, outlining the goals of the proposed privacy-preserving classification protocols. The proposed protocols and an explanation of the NB implementation with FHE operations are described in Section 5, and a performance evaluation is presented in Section 6. Finally, Section 7 concludes the paper.

2. Related work

Studies on privacy-preserving machine learning have been diverse, as explained in [15]. Many of them are not focused on protecting user input for classification, or on the privacy of the model for the classification process. Rather, they tend to focus on protecting the privacy of the collected training data. Examples of such methods are randomization [16,17] and anonymization [18], which concentrate on transforming the collected training data with minimal impact on the overall training outcome.

Privacy protection in classification protocols has recently been defined as the problem of successfully completing the classification, providing the classification results to the client while preserving the privacy of the feature vector $\vec{x}$, which is the input of the user (i.e., client), and the model information w, which is owned by the server [1]. That is, no private information from either party should be exposed to the other during classification.

A common solution to this problem is to use a secure two-party computation (STC) protocol that can be applied to arbitrary algorithms [19–23]. If the machine learning algorithm is converted to an STC protocol, the algorithm can be executed in a privacy-preserving manner. However, significant computation and memory resources are needed to run such a protocol. In addition, the classification results given by such protocols are somewhat inaccurate [1]. In [1], a privacy-preserving NB protocol was implemented using the STC protocol of [21,22]. However, it was confirmed that more than 256GB of system memory was required to run the NB classification for data with three features. In [24], the probability that simple operations such as addition, ARGMIN, etc. would fail when using limited memory of 4GB or less was 71.4% for [21] and 14.3% for [22]. Thus, without significant memory resources, it is difficult to create a high-accuracy, privacy-preserving classifier using this approach.

For this reason, many attempts have been made to create a customized protocol that combines various methods. With these combined methods, the core-element operations necessary for implementing the classifier take the form of a protocol that can be performed with encrypted data. Thus, the classification algorithm can be executed by running the relevant protocols [1–4] in sequence. In this case, a number of interactions occur between the server and the client. In one study [1], NB and hyper-plane classification protocols were implemented using additive homomorphic encryption, which (unlike FHE) provides only a limited type of computation. To implement a decision tree classifier, both additive homomorphic encryption and FHE are used to construct the protocol. In [2], NB and hyper-plane classifiers were implemented using commodity-based cryptography, where both the client and server share the necessary operations in a secure manner through pre-distributed correlated data in the setup phase. A subsequent study [3] improved the performance of the classifier proposed in [2] by updating the building blocks. Moreover, [4] proposed an NB classification protocol for medical diagnosis using a customized protocol based on additive homomorphic encryption and a customized multiplication protocol.

Unfortunately, all of the above combined methods are inefficient: they require multi-round interactions between the server and client. In addition, intermediate values must occasionally be decrypted during protocol execution, resulting in the security problem of the private key information residing in memory [1–3].

One technique that suffers this type of problem is the DGK protocol [25], which is used for numerical comparisons with classification protocols [1–3]. Under DGK, it is assumed that the user has encrypted data and the server has a private key. The user transmits his/her data to the server containing the secret key in order to compare the encrypted numbers in the data. The server decrypts the encrypted numbers, performs the comparison, and then transmits the results to the user. Basic interaction is thus essential with this protocol.

However, DGK allows the server to know the two numbers being compared, thus violating the privacy of the user data. To prevent this, the user can transform the original data using a random number before delivering the result to the server. The server performs a comparison operation based on the modified data. In this case, however, because the original data have been transformed, the server needs to perform additional interactions with the user to exchange more relevant data and perform more decryptions. Such an increase in interaction is a heavy burden on the server—it must maintain communication while working with many users. Furthermore, frequent decryptions are not desirable, given the increased exposure of the decryption key, which might become the main target for attackers.

Existing protocols have another security problem: a side-channel attack can be performed by analyzing protocol executions. If a classification protocol is a combination of building blocks, the pattern of the messages generated during the interaction for each building block, the total number of communications, and the communication time depend on the building block currently being executed, which risks exposing additional information about the user input and model. For example, in the case of a decision tree classifier [1], information about the depth of the model can be exposed by measuring the tree evaluation time. Although an improved version [3] managed to reduce the computation time required for tree evaluations, there is still a risk that the depth of the model will be exposed.

Many non-interactive privacy-preserving classification protocols that avoid the above problems have been studied [26–28]. In such schemes, message transmission only occurs when the client sends the input and receives the classification result. However, most of these methods have problems in terms of classification accuracy or can support only a weak level of security, where the privacy-protection problem defined in [1] cannot be solved. For instance, [26] implemented a decision tree and a random forest classifier that requires only one round of interaction, using additive homomorphic encryption and the oblivious transfer protocol (OTP). However, because this method exposes the depth of the classification tree, which is part of the model, its security is weak. Moreover, a special method is used to implement the two classifiers, making it difficult to apply this method to other cases.

There have been other attempts at performing only one round of interaction [27,28], and these approaches are suitable for implementing arbitrary classifiers. These methods utilize somewhat homomorphic encryption (SWHE), which is similar to FHE, but with a limited depth of multiplication operations in the circuit. They implement linear means and Fisher's linear discriminant classifiers, which include privacy-preserving model training. In addition to the advantage of one-round interaction, these methods also have the security advantage that no user feature information is exposed to the server. However, with the limited multiplication depth of SWHE, the method requires the user to decode the final output to determine the classification result. In doing so, the user obtains information regarding the range of output values corresponding to each classification result class. Therefore, such methods do not satisfy the security requirements in [1], insofar as model information is exposed to users.

There has also been research on the FHE method itself. In [28], the authors exploit Cox proportional-hazards regression and a linear regression classifier that takes encrypted patient data as input. However, this method focuses only on protecting the user's input. Information about the model required for the regression process is disclosed to everyone, including the patient. Specifically, when a user receives a regression result, the probability value used for classification is transmitted, rather than the class value. This probability represents model information, and therefore does not satisfy the privacy requirements proposed in [1].

As mentioned above, a common problem [27,28] is the lack of a comparison operation between ciphertexts. This forces the user to determine the class value of the user input by comparing the regression results to a reference value. Consequently, model information is exposed to the user. In [29], the authors proposed an improved method that mitigates

Table 1
Comparison of existing privacy-preserving classification protocols in terms of security, message transmission efficiency, and number of intermediate decryptions.

	Classifier	Communication (# of rounds)	# of decryptions	Leaked information
	Naive Bayes	$4(t-1)+1$	$3(t-1)$	
[1]	Hyper-plane	$4(t-1)+1$	$3(t-1)$	
	Decision tree	4.5	$6t-1$	Possibility of side-channel attack to extract depth of tree
[2]	Naive Bayes	$(q+1)+t+1.5$	–	
	Hyper-plane	$(q+1)+t+2.5$	–	
[3]	Decision tree	$q+\log t+2.5$		Possibility of side-channel attack to extract depth of tree
	Hyper-plane	$(q+1)+t+1.5$		
[4]	Naive Bayes	$\log t+2$	$t+1$	
[26]	Decision tree	1	1	Depth of tree
	Random forest			
[27]	Linear Means	1	1	Model info.
	Fisher's linear discriminant			
[28]	Cox proportional hazards regression	1	1	Model info.
	Linear regression			

this, relative to [27,28]. To do so, [29] deals with statistical analysis techniques such as matrix multiplication, comparison operations, a contingency table, and linear regression over FHE ciphertexts. In particular, linear regression and the comparison operation allow for the design of a privacy-preserving linear regression classifier. For example, when a user encrypts an input and sends it to the server, the server performs linear regression using the model and the transmitted user input. Because the server can perform the comparison operation to classify the results of the regression over the encrypted data using FHE, the user ultimately receives only the class value corresponding to the classification result. However, because the computation time required for the comparison process and the required storage space both increase in proportion to the exponent of the bit length of the input data and model data, the comparison operation is only practical for limited-sized integer comparison.

Compared to the privacy-preserving comparison operations based on a garbled circuit in [30], the comparison operations of [29] are slightly faster than the comparison of integers that are less than 2^{16}. However, if the maximum value is increased to approximately 2^{24}, [30] requires at least 20 times more computation time and at least 10^6 times more storage.

In [31], the authors address useful techniques such as Bayesian spam filtering and decision tree evaluations over FHE-encrypted data. In their work, the database stores keywords and the probabilities that e-mails containing these words are spam. If any of the keywords is included in the e-mail, the Bayesian spam filter determines whether each word contained in the encrypted e-mail is included in the keyword set stored in the database. If we know which keywords in the DB are contained in the email and their associated probabilities, we can easily calculate the probability that the email is spam. Unfortunately, [31] does not describe a method for calculating this probability when the keyword matching result is encrypted by FHE. Thus, the performance analysis in [31] focused exclusively on the keyword-matching operation, without considering the calculation of the probability of spam using the probability stored in the database. For decision tree classification, when a comparison result is given for each node, the study suggests a tree evaluation method. However, this assumes that the comparison results between the user input and tree node value are known. Thus, the study does not provide details of the comparison itself.

Table 1 summarizes the methods described in this section in terms of the number of necessary message transmissions, the number of decryptions performed during protocol execution, and the information exposed. In describing [26] in the table, we used the superior method in terms of security and number of message transmissions among the two proposed methods. In the case of [4], it is possible to return the best $k(\geq 1)$ results, but we assumed $k = 1$ when calculating the values in the table. The number of message transmissions in [1–4] was calculated assuming that messages that can be transmitted simultaneously are sent together.

3. Preliminaries

This section describes some basic concepts used in the rest of this paper, and defines the symbols and notation used. First, the NB classifier is described. Then, we introduce FHE, which is essential for the implementation of the proposed protocol. The symbols and notation used in the remainder of this paper are given in Table 2.

3.1. Naive Bayes classifier

The NB classifier is very efficient and has been widely used in applications such as text classification [32,33], medical diagnosis [34,35], and system performance management [36]. The NB classifier is briefly described as follows.

Given a set of feature vectors with d elements $D(= D_1 \times D_2 \times \cdots \times D_d)$ and a set of classification results $C_r = \{c_1, \cdots, c_t\}$ for an input feature vector $\vec{x} \in D$, the classifier determines the corresponding classification results $c_y \in C_r$. In other words, if $C_w(\vec{x})$ is the result from running classifier C_w with $\vec{x}$, the NB classifier returns y to represent c_y. In the NB classifier, y can be determined with the following formula:

$$
\begin{aligned}
y &= argmax_{1\leq i\leq t} Pr[C = c_i | X = \vec{x}] \\
&= argmax_{1\leq i\leq t} \frac{Pr[C = c_i] \cdot Pr[X = \vec{x} | C = c_i]}{Pr[X = \vec{x}]} \\
&= argmax_{1\leq i\leq t} Pr[C = c_i] \cdot Pr[X = \vec{x} | C = c_i] \qquad (1)
\end{aligned}
$$

Given $\vec{x} = (x_1, \cdots, x_d)$, it is assumed all elements $x_1, \cdots, x_d$ are conditionally and independently drawn from one another. Thus, the following formula can be derived based on Bayes' rule:

$$
Pr[X = \vec{x} | C = c_i] = \prod_{j=1}^{d} Pr[X_j = x_j | C = c_i] \qquad (2)
$$

Table 2
Notation.

Notation	Description
sk	Private key in FHE
pk	Public key in FHE
Encrypt	Encryption algorithm in FHE (**Encrypt**())
Decrypt	Decryption algorithm in FHE (**Decrypt**())
[*a*, *b*]	$\{a, a+1, \cdots, b-1, b\}$
$\oplus$	**Encrypted_XOR**() in FHE
$\cdot$	**Encrypted_Multiply**() in FHE
s	Number of slots available in a ciphertext
u	Smallest unit of a slot rotating in the **Rotate** operation. Slot movement using **Rotate** can only be done in multiples of u slots.
$<<_R i$ / $>>_R i$	Rotate the slot inside the ciphertext to the left/right by $i \cdot u$ slots ($0 \le i < \lfloor s/u \rfloor$) using the **Rotate** operation
$<<_S i$ / $>>_S i$	Shift the slot inside the ciphertext to the left/right by i slots ($0 \le i < s$) using the **Shift** operation
t	Size of the set of classification result classes ($= \lvert C_r \rvert$)
C_r	Set of classification result classes ($= \{c_1, c_2, \cdots, c_t\}$)
c_i	ith class in the set of classification result classes ($i \in [1, t]$)
C	Random variable associated with the set C_r
$\vec{x}$	User's input feature vector $\vec{x} = (x_1, x_2, \cdots, x_d) \in D$
d	Number of attributes in $\vec{x}$
x_j	jth element in $\vec{x}$ ($j \in [1, d]$)
X	Random variable drawn from D
X_j	Random variable drawn from D_j ($j \in [1, d]$)
D_j	Domain of all possible x_j, where $j \in [1, d]$
$v_{j,\,k}$	kth element of D_j, where $j \in [1, d]$ and $k \in [1, \lvert D_j \rvert]$
D	$D_1 \times D_2 \times \cdots \times D_d$
w	Model of a classifier
C_w	Classifier defined as a function from $D \rightarrow C_r$
K	Constant number multiplied by the log-scale probabilities to render them integers
S	Array of ciphertexts to store $K \log Pr[C = c_i]$ in the NB classifier, where ($1 \le i \le t$)
q	Maximum bit length that can be used to represent one value in the NB classifier
LSB	Ciphertext that has 1 in the least significant slot and 0 in the other slots
MAX	Ciphertext that has 1 in q slots that are in the designated positions. This will be referred to as a ciphertext containing $2^q - 1$.
U	Ciphertext of the user input feature vector $\vec{x}$
R_1	Resultant ciphertext in the proposed classifier in the server-centric setting, which stores $C_w(x)$
R_2	Resultant ciphertext in the proposed classifier in the user-centric classification, which stores $C_w(x) >>_R r$, where r is randomly selected from $\{0, 1, \cdots, \lfloor s/u \rfloor\}$
DP	Data provider

Using the above formula, (1) can be rewritten as follows:

$$y = \underset{1 \le i \le t}{\operatorname{argmax}}\, Pr[C = c_i] \prod_{j=1}^{d} Pr[X_j = x_j | C = c_i]$$

$$= \underset{1 \le i \le t}{\operatorname{argmax}} (\log Pr[C = c_i] + \sum_{j=1}^{d} \log Pr[X_j = x_j | C = c_i]) \quad (3)$$

From (3), we can perform NB classification if the following sets are available:

$$\{\log Pr[C = c_i] | i \in \{1, \cdots, t\}\}$$

$$\{\log Pr[X_j = x_j | C = c_i] | i \in \{1, \cdots, t\}, j \in \{1, \cdots, d\}\}$$

Therefore, we can treat the above sets as a model for the NB classifier.

3.2. Fully homomorphic encryption

In the proposed protocol, we adopt the FHE method [8–11,37] to implement a privacy-preserving classifier. It is easy to demonstrate the feasibility of this method, because its implementation is open; it is based on the Homomorphic Encryption Library (HELib) [13]. HELib's method is one of the most practical FHEs yet developed [38–41], and provides the following operations (see Fig. 1).

- **Setup**(1^λ): takes a security parameter λ and returns a system parameter *param*.
- **KeyGen**(*params*): takes *param*, the result of the **Setup** operation, and returns a private/public key pair (sk, pk).
- **Encrypt**($params, pk, \mathbf{m}$): takes a plaintext $\mathbf{m}$, *params*, and public key *pk*. It outputs the encryption result (ciphertext) *ctxt*. Here, $\mathbf{m}$ may contain a vector of plaintexts.
- **Decrypt**($params, sk, ctxt$): takes a ciphertext *ctxt*, a private key *sk*, and a security parameter *param*. It outputs a plaintext $\mathbf{m}$ if *ctxt* is the result of **Encrypt**($params, pk, \mathbf{m}$), where (sk, pk) is the result of running **KeyGen**(*params*). Otherwise, it returns $\perp$.
- **Encrypted_XOR**($params, pk, ctxt_1, ctxt_2$): returns $ctxt_3$, where **Decrypt**($params, sk, ctxt_3$) produces $\mathbf{m_1} \oplus \mathbf{m_2}$, if $ctxt_1$ is the result of **Encrypt**($params, pk, \mathbf{m_1}$) and $ctxt_2$ is the result of **Encrypt**($params, pk, \mathbf{m_2}$), where (sk, pk) is the result of **KeyGen**(*params*). Otherwise, it returns $\perp$.
- **Encrypted_Multiply**($params, pk, ctxt_1, ctxt_2$): similar to **Encrypted_XOR**, but the result of the decryption of $ctxt3$, the output of this algorithm, is $\mathbf{m_1} \cdot \mathbf{m_2}$ (component-wise multiplication).
- **Recrypt**($params, pk, ctxt$): returns $ctxt'$, which contains the same plaintext as *ctxt*. For this operation to work correctly, the depth of the multiplication operations from ciphertexts created by **Encrypt** to *ctxt* must be less than or equal to α. Then, $ctxt'$ can be multiplied by the depth $\alpha - \beta$, where α is the value determined in *params* and β is the value associated with the **Recrypt** operation. To perform multiplication operations at depths greater than $\alpha - \beta$, this operation must be performed again after performing multiplications at a depth of $\alpha - \beta$.
- **Pack**($params, pk, m_0, m_1, \cdots, m_{s-1}$): takes arbitrary plaintexts $m_0, \cdots, m_{s-1}$ and generates a vector of plaintexts $\mathbf{m}$.
- **UnPack**($params, pk, \mathbf{m}$): returns $m_0, \cdots, m_{s-1}$ encoded in $\mathbf{m}$.
- **Rotate**($params, pk, ctxt, i$): rotates the plaintexts in the ciphertext *ctxt* to the left by $i \cdot u$ slots. Here, u is the minimum unit of data movement provided by automorphism without increasing the noise, and i has a range of $-s/u < i < s/u$. If $i < 0$, it is rotated to the right by $-iu$.
- **Shift**($params, pk, ctxt, i$): moves the plaintexts in the ciphertext to the left by i on the slot basis, and fills the last i slots on the right with 0s. Here, i has a range of $-s < i < s$; if $i < 0$, it is moved to the right by $-i$. Unlike **Rotate**(), this increases the noise in the ciphertext.

Fig. 1. FHE algorithms [43].

HELib [42] provides the ability to transform multiple independent single-bit-sized plaintexts into a single plaintext such that it can be encrypted with a single encryption operation. In the transformed plaintext, one slot is assigned to each original plaintext bit, and this slot structure is preserved in the ciphertext. The total number of slots for the ciphertext is given by s, whose value is determined by the selected parameter. This is also related to the security of the FHE scheme and the speed of operations in FHE. The **Pack**/**UnPack** algorithms support the ability to pack/unpack multiple plaintext bits into/from a single plaintext vector for encryption.

HElib's FHE scheme relies on the Ring-LWE (Learning With Errors) problem [44] for its security. In schemes that rely on this problem, the encryption algorithm normally adds a small amount of noise to the ciphertext at the end of the enciphering operation. If the size of the noise is small, the corresponding noise can be canceled in the decryption operation with the correct decryption key, allowing the correct plaintext to be obtained by running the decryption operation. This noise value is a key factor in ensuring the security of the Ring-LWE problem.

Unfortunately, the noise value increases when performing operations between ciphertexts. If the size of the noise in a ciphertext exceeds a certain threshold, which depends on the security parameters, the ciphertext cannot be decrypted correctly. Therefore, when developing an application based on encrypted data with FHE, it is necessary to manage the noise inside the FHE ciphertexts. The noise included in the ciphertexts of FHE increases slightly when the **Encrypted_XOR** is performed and increases significantly when the **Encrypted_Multiply** is performed. Thus, after a certain number of these operations, we need to reduce the noise in the ciphertext to allow more operations to be performed. The **Recrypt** operation plays this role. This operation produces a new ciphertext that has the same plaintext as the original ciphertext, but contains only a small amount of noise. As some ciphertext operations are executed with the new ciphertext while running **Recrypt**, the number of ciphertext operations possible with the new ciphertext is smaller than that of a ciphertext created by the encryption operation.

We manage the number of consecutive multiplications[2] using ciphertexts for noise management. Based on this, we can estimate the size of the noise in the ciphertexts and can determine when the **Recrypt** operation is necessary.

4. System model and motivation

The system model proposed in this study is as follows. There are three types of entities in this system. First, there is a Data Provider (DP). During the system setup step, the DP performs a training process to create a classification model based on a pre-owned dataset. Second, there is a server, which receives a classification service request from a user and provides a prediction result to the user. Third, there is a user (i.e., client), who requests a classification service using their own information from the server. Finally, we assume that the server and client follow the Honest-But-Curious (HBC) adversary models, in that the server wants to obtain the private information of the client and vice-versa. However, they are assumed to obey the protocols [45]. The main focus of this study is services that operate under the above system model with the privacy-preserving requirement. A typical example is the Clinical Decision Support System (CDSS) [7], which performs classification based on data collected by a hospital consortium or government agency. Based on the underlying model, CDSS provides automatic diagnostic services to doctors and patients. The privacy-protection requirements of these services are as follows. First, to use the automatic diagnostic service of CDSS, doctors or patients must submit the patient information required as input for the classification task. At this time, to protect patient privacy, neither the inputs provided to CDSS during the protocol execution nor the classification results corresponding to the diagnosis should be exposed to CDSS. Second, for CDSS, the model information is a high-cost asset. Therefore, the model information stored in the server should not be exposed to the doctors or patients who use the service.

However, existing machine learning methods that are not privacy-preserving cannot satisfy the above conditions. Indeed, [46,47] show how serious privacy breaches can arise from open access to models, classification results, and user data in machine learning. Specifically, [46] shows that it is possible to infer private information used to generate the model values using published model values, and [47] illustrates a method for identifying the user corresponding to a classification result by combining the classification results with other non-personally identifiable information.

From the above argument, if conventional machine learning techniques are used, attackers could acquire patients medical information by applying the methods in [46] and [47]. This is a serious problem, and may also violate medical-records legislation. Therefore, it is important to protect the privacy of the model information used by the classifier, as well as securing the user input to the classifier and the prediction results.

The security/functional requirements for privacy-preserving classification can be summarized as follows.

- *Privacy-preserving*: when the user input and classification results are exposed to the server that provides the classification service, the service provider can acquire sensitive information about the user, a potentially serious privacy violation. Therefore, the information should not be exposed to the server. Moreover, the servers model requires a large amount of data for its generation, which entails considerable cost. Therefore, if this information is exposed to the user, the service provider operating the server risks suffering financial harm. Thus, these privacy-preserving requirements should be maintained even in the face of quantum computing attacks.
- *Non-interactive*: if multiple rounds of interactions are performed between the server and the user when running the classification protocol, both the user and the server must be online during these interactions, and status information must be managed to keep the protocol running. This can be burdensome to both the service and the users, and makes it difficult to provide the service in the form of a web application. In addition, the communication patterns from running a protocol can be used for side-channel attacks. Therefore, communication between the server and the user should be minimized. Ideally, all work should be completed in one round of communication (i.e., input transmission and receipt of results).
- *Minimal use of decryption when executing the protocol*: during protocol execution, the user and the server must minimize the decryption of intermediate protocol results, because these are vulnerable to attacks when private key information is present in memory over a long period of time.

5. Proposed protocols

In this section, we propose a new privacy-preserving classification protocol that meets the requirements stated in the previous section. The key idea of the proposal is to use FHE to enhance security, along with various optimization techniques to improve the performance of the proposed protocols FHE operations.

5.1. Protocol overview

We propose two protocols that operate in different settings. The first protocol is aimed at an environment where there is excellent available computing power for the server and the support of high-level system security to keep the model secure. In this case, the goal is to ensure the privacy of the users information against the server. An overview of this server-centric protocol is shown in Fig. 2-(A). The second protocol targets an environment in which the server needs to ensure the security of the model, because of its relatively low computational complexity and security. In such an environment, the user's information operates only in its own computing environment, and the server must provide model information to the user in order to perform the classification operation for the client. The server only needs to decrypt the results of the operation and transmit them to the user. An overview of this user-centric protocol is shown in Fig. 2-(B).

In the first model (Fig. 2-(A)), the server only has the user's public key (*pk*), and performs classification using the encrypted user input and a plaintext form of the model. Because the classification is performed by the server, its results are encrypted with the user's public key. Thus, the server cannot obtain information about the classification results. Because the user receives only the final encrypted results from the server,

[2] This value is determined by the multiplicative depth of the circuit that is executed with the ciphertext. Because **Recrypt** is very costly, we need to design our algorithm's multiplicative circuit depth to be as shallow as possible for efficiency.

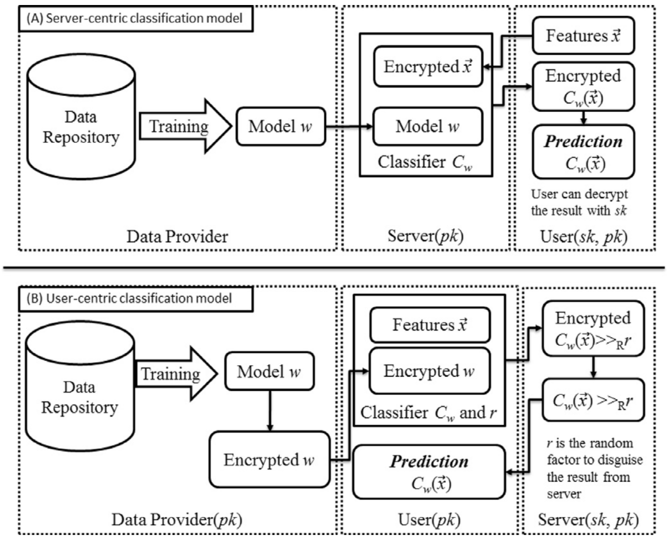


Fig. 2. Overview of the proposed protocols: (A) server-centric, (B) user-centric.

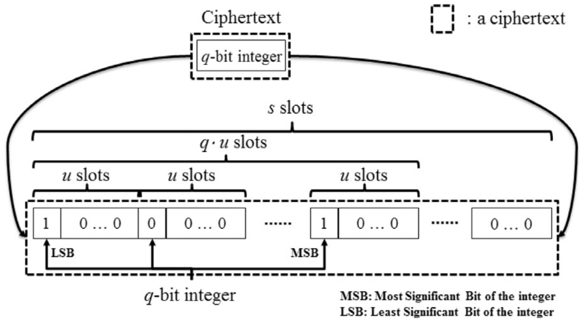


Fig. 3. Structure of a ciphertext storing a q-bit integer.

it is difficult to know information about the model used for the classification. The user has the private key (*sk*) and can obtain the classification results by decrypting the output received from the server.

In the second protocol (Fig. 2-(B)), the user performs the classification directly, and the model information used to do so is protected by encryption using the server's public key. Users have access to the server's public key (*pk*). In this environment, the DP generates a model and then encrypts it with the server's public key. The model is then released to the user during the initial stage. The user performs classification using his/her input information and the model, and then transmits the classification results to the server and requests decryption, which is done using the servers private key (*sk*). The user hides the classification results with a random number in the encrypted classification results before the decryption request, making it difficult for the server to deduce the classification results after decryption. After receiving the decryption results from the server, the user removes the random number used to hide the classification results and obtains the final results. In the server-centric protocol, the setup is completed only when the model is stored in the server. Therefore, when this procedure is completed, the user can send input information encrypted with his/her public key to the server in order to proceed with the classification. For the user-centric protocol, the encrypted model information must be distributed to all users, incurring the cost of ensuring the integrity of the information. Thus, the server-centric protocol has the advantage in terms of the cost required to distribute the model. Nevertheless, the user-centric protocol has several advantages over the server-centric protocol. First, the server-centric protocol needs to store a plaintext form of the model in the server. In the environment assumed by this protocol, each user transmits the input feature vector $\vec{x}$ encrypted with his/her own key to protect $\vec{x}$. There-

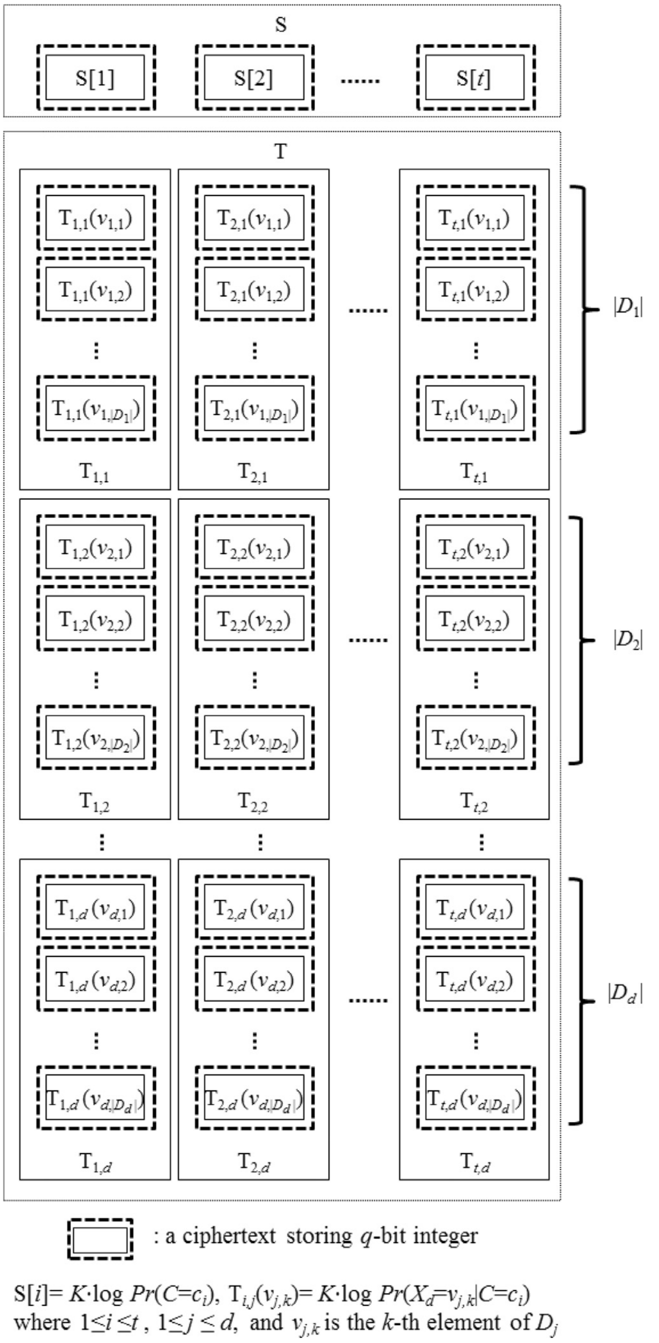


Fig. 4. Illustration of encrypted model $w(S, T)$.

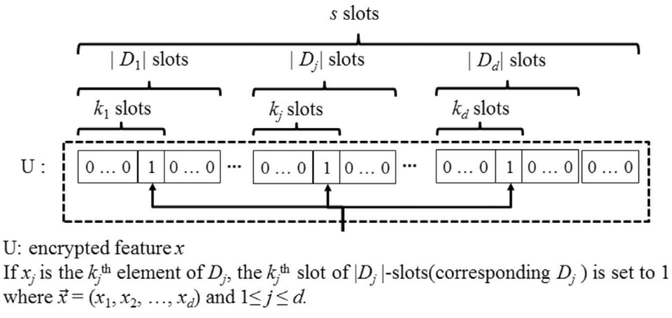

s slots
$|D_1|$ slots
$|D_j|$ slots
$|D_d|$ slots
k_1 slots
k_j slots
k_d slots
U :
0 … 0
1
0 … 0
…
0 … 0
1
0 … 0
…
0 … 0
1
0 … 0
0 … 0
U: encrypted feature x
If x_j is the k_j^{th} element of D_j, the k_j^{th} slot of $|D_j|$-slots(corresponding D_j) is set to 1 where $\vec{x} = (x_1, x_2, \ldots, x_d)$ and $1 \le j \le d$.

Fig. 5. Structure of U, an encrypted user input $\vec{x} = (x_1, x_2, \cdots, x_d)$.

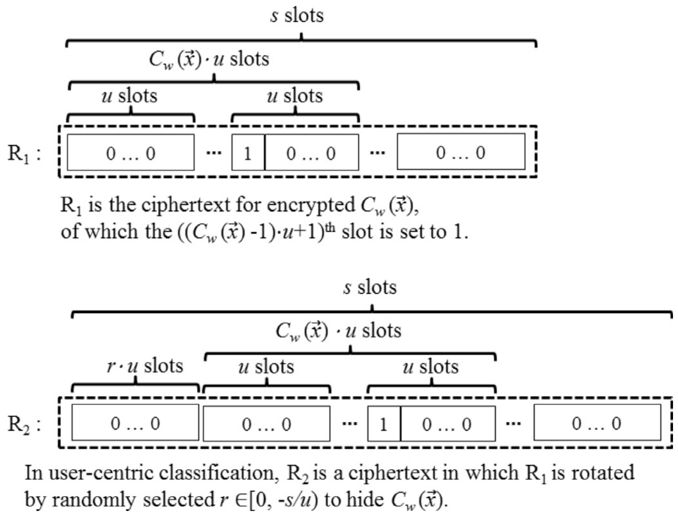

s slots
$C_w(\vec{x})\cdot u$ slots
u slots
u slots
R1 :
0 … 0
…
1
0 … 0
…
0 … 0
R1 is the ciphertext for encrypted $C_w(\vec{x})$, of which the $((C_w(\vec{x})-1)\cdot u+1)^{th}$ slot is set to 1.
s slots
$C_w(\vec{x})\cdot u$ slots
$r\cdot u$ slots
u slots
u slots
R2 :
0 … 0
0 … 0
…
1
0 … 0
…
0 … 0
In user-centric classification, R2 is a ciphertext in which R1 is rotated by randomly selected $r\in[0, -s/u)$ to hide $C_w(\vec{x})$.

Fig. 6. Ciphertext structure of the classification result: $R_1(= C_w(\vec{x}))$ and $R_2(= C_w(\vec{x}) >>_R r)$.

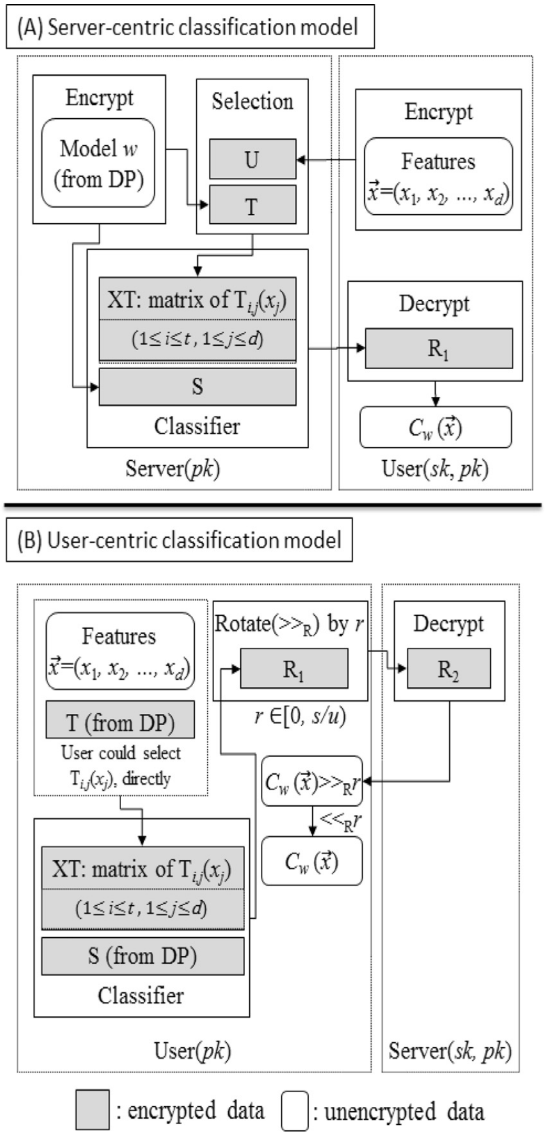

(A) Server-centric classification model
Encrypt
Model w (from DP)
Selection
U
T
Encrypt
Features
$\vec{x}=(x_1, x_2, \ldots, x_d)$
XT: matrix of $T_{i,j}(x_j)$
$(1\leq i\leq t, 1\leq j\leq d)$
S
Classifier
Decrypt
R1
$C_w(\vec{x})$
Server(pk)
User(sk, pk)
(B) User-centric classification model
Features
$\vec{x}=(x_1, x_2, \ldots, x_d)$
T (from DP)
User could select $T_{i,j}(x_j)$, directly
Rotate($>>_R$) by r
R1
$r\in[0, s/u)$
Decrypt
R2
$C_w(\vec{x})>>_R r$
$<<_R r$
$C_w(\vec{x})$
XT: matrix of $T_{i,j}(x_j)$
$(1\leq i\leq t, 1\leq j\leq d)$
S (from DP)
Classifier
User(pk)
Server(sk, pk)
▨ : encrypted data
□ : unencrypted data

Fig. 7. Schematic view of the proposed NB protocols.

fore, to perform classifications with various $\vec{x}$ encrypted with different keys, the model exists in plaintext form and must be encrypted with the public key used to encrypt $\vec{x}$. For this reason, the model cannot be managed in an encrypted form. Therefore, the security of the model in the server depends on the system-level security of the server. In contrast, the user-centric model exists only as an encrypted value. Consequently, the probability that the information will be leaked is very small. Moreover, because there is no model value inside the server, the management cost of the server data is low. Second, there is the advantage that the operations required for machine learning are distributed to each user. Even if the cloud has a lot of resources, running an application based on FHE is computationally expensive. For example, one study of machine learning using FHE [27] showed that the proposed classifier required approximately 0.3–21 s per run. Even if we assume a cloud environment with sufficient resources, providing services to multiple users at the same time can be problematic in practical terms. For user-centric classification (Fig. 2-(B)), in contrast, the server has minimal computational overheads, insofar as it exclusively performs decryption. In addition, in the case of a non-interactive protocol configuration, the server can collect the classification results from users and transmit the results with only one broadcast. This is possible because the classification results are hidden by random numbers added by individual users.

The following subsections discuss the practical implementation of the privacy-preserving machine learning protocols in the two environments described above. The tool used in this study is FHE with the NB classifier.

5.2. Proposed implementation for the Naive Bayes classifier

This subsection describes the implementation of the privacy-preserving NB classifier.

5.2.1. Data representation

We now address how the values used for classification are represented in ciphertexts. In the NB classifier, the model w refers to all probability values that are used to classify a user's input $\vec{x}\in D$. The classifier, denoted by C_w, is defined as a function $C_w\colon D\to\{c_1,\cdots,c_t\}$. Hereafter, we use i and c_i interchangeably to describe the classification result $C_w(x)$, unless there is some ambiguity. To implement the classifier efficiently as a circuit for FHE operations, the proposed method uses a modified version of the NB classifier, where the logarithms of all probabilities are used as described in Section 3.1. Further, to express the probabilities in ciphertexts, where an integer representation is preferable, they are multiplied by the same K such that all values are represented as integer values with a length of q bits. This is possible because the NB classifier produces the same result with modified values as it does with the original probability values.

According to [48,49], there is no significant loss of classification efficiency when the number of bits used to represent the probabilities is fixed to 10. From this, we can say that setting $q = 32$ is sufficient to represent the probabilities used in the NB classifier. We verify this experimentally in Section 6.2.

Fig. 3 shows the structure of a ciphertext that encrypts a q-bit integer with the proposed NB classifier. To use the **Rotate** algorithm ($<<_R$, $>>_R$)—which does not increase noise, owing to the use of the FHE automorphism—the ciphertext stores 1 bit in the u slot interval, which is the minimum slot-moving unit for the **Rotate** operation. The q-bit data in the ciphertext are stored such that the least significant bit (LSB) is located in the first slot and the most significant bit (MSB) is located in the $((q-1)\cdot u+1)$ th slot. All integer types of data that represent the probabilities in an encrypted form (stored in tables S and T) follow the structure in Fig. 3.

The model w for the NB classifier is the logarithm of the probabilities that are necessary to calculate Eq. (3) from Section 3.1. If we assume $D_j = \{v_{j,1}, v_{j,2}, \cdots, v_{j,|D_j|}\}$ $(1\leq j\leq d)$, the entire model can be repre-

(a) Server-centric NB classifier

- **Client's input**: input feature vector $\vec{x} = (x_1, \cdots, x_d)$, pk, sk
- **Server's input**: encrypted log-scale probability tables (S, T), pk

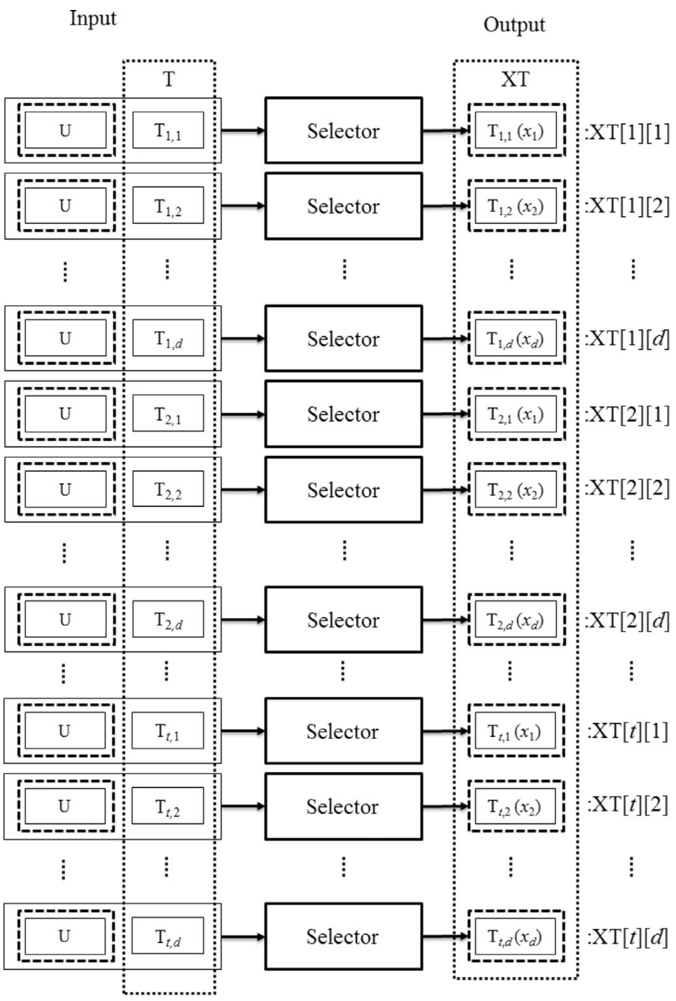
1. (**Client** → **Server**): **Client** generates U by encrypting $\vec{x}$ with pk and sends it to **Server**.
2. (**Server**): extracts XT, where $XT[i][j]$ = (an encryption of $T_{i,j}(x_j)$) using the SELECTION circuit with U and T. ($T_{i,j}(x_j) \triangleq K \cdot \log Pr[X_j = x_j | C = c_i]$, $i \in [1, t], j \in [1, d]$).
3. (**Server**): calls CLASSIFIER circuit with XT and S as inputs to obtain an output R_1. The resultant R_1 contains a number i, which is the result of $argmax_{i=1}^{t}(K \cdot \log Pr[C = c_i] + \sum_{j=1}^{d} K \cdot \log Pr[X_j = x_j|C = c_i])$=$argmax_{i=1}^{t}(S[i] + \sum_{j=1}^{d} XT[i][j])$. $S[i] + \sum_{j=1}^{d} XT[i][j]$ for each i is calculated using the MULTIPLEADDER circuit in the CLASSIFIER.
4. (**Server** → **Client**): **Server** sends **Client** R_1 and **Client** decrypts R_1 by running **Decrypt**(sk,R_1), and obtains the classification result $C_w(x) \in [1, t]$.

(b) User-centric NB classifier

- **Client's input**: input feature vector $\vec{x} = (x_1, \cdots, x_d)$, encrypted log-scale probability tables (S, T), pk
- **Server's input**: encrypted log-scale probability tables (S, T), sk, pk

1. (**Client**): generates XT with x. Because it is not encrypted, XT can be constructed by copying the corresponding elements from T. There is no need to run the SELECTION circuit.
2. (**Client**): Run CLASSIFIER as in Step 3 in the server-centric setting. This obtains R_1.
3. (**Client** → **Server**): **Client** randomly generates $r \in [0, \lfloor s/u \rfloor]$ and obtains $R_2 \leftarrow R_1 >>_R r$. Send R_2 to **Server**.
4. (**Server** → **Client**): **Server** runs **Decrypt**(sk,R_2) to obtain RET and sends RET to **Client**. **Client** obtains the classification result by running $RET <<_R r$.

Fig. 8. Description of the proposed classifier protocols.



$XT[i][j]$=$T_{i,j}(x_j)$ is a ciphertext for storing $K \cdot \log Pr(X_j = v_{j,k_j} | C = c_i)$
where $1 \le i \le t$, $1 \le j \le d$, and $x_j = v_{j,k_j}$

Fig. 9. Illustration of the SELECTION circuit.

sented with $\{\log Pr[C = c_i] | 1 \le i \le t\}$, $\{\log Pr[X_j = v_{j,k} | C = c_i] | 1 \le i \le t, 1 \le j \le d, 1 \le k \le |D_j|\}$.

To represent the probabilities as integers, we assume that a suitable K is multiplied by every element in the above sets:

$$|K \cdot \max_{i \in [1,t]} \{\log Pr[C = c_i] + \sum_{j=1}^{d} \log Pr[X_j = v_j | C = c_i]\}| < 2^q$$

(where $v_j \in D_j$)

Following the above argument, we can say that sets T and S (see below) constitute the model w. Of course, every element in S, T is in an encrypted form when it is used.

$$S = \{S[i]\}_{1 \le i \le t} = \{K \cdot \log Pr[C = c_i] | 1 \le i \le t\}$$

$$T = \{T_{i,j}(v_{j,k})\}_{1 \le i \le t, 1 \le j \le d, 1 \le k \le |D_j|}$$

$$= \{K \cdot \log Pr[X_j = v_{j,k} | C = c_i]\}_{1 \le i \le t, 1 \le j \le d, 1 \le k \le |D_j|}$$

Fig. 4 illustrates tables T and S. We can access the encrypted elements in the tables using an appropriate index, but the values of the elements are unknown until they are decrypted. S is composed of t ciphertexts and T is composed of $t \cdot \sum_{j=1}^{d} |D_j|$ ciphertexts. In the user-centric protocol, it could be burdensome to the client to store these. However, if we pack them in a compact way, far fewer ciphertexts are necessary: because one ciphertext of FHE is usually able to store 800–1600 bits, if we assume q is around 10, only 1/80–1/160 of the ciphertexts in our original implementation are required. More precisely, if we use the **Pack** algorithm shown in Fig. 1, $\lceil q \cdot t/s \rceil$ and $\lceil q \cdot t/s \cdot \sum_{j=1}^{d} |D_j| \rceil$ ciphertexts are necessary to construct S and T, respectively, as s bits can be contained in a ciphertext.

For classification, the user input $\vec{x} = (x_1, x_2, \cdots, x_d) \in D$ is given in the form of U in Fig. 5. The input is an encrypted bit string in which each bit value is stored in the corresponding slot in a ciphertext. Because we can order every element in D_j as $\{v_{j,1}, v_{j,2}, \cdots, v_{j,d}\}$ ($j \in [1, d]$), we can rewrite $\vec{x} = (v_{1,k_1}, \cdots, v_{d,k_d})$ if $x_j = v_{j,k_j}$ for all $j \in [1, d]$, where $k_j \in [1, |D_j|]$. Here, U contains the information $(k_1, k_2, \cdots, k_d)$ as an underlying plaintext, as seen in Fig. 5; we set only the k_jth bit to 1 and the other bits to 0, among all $|D_j|$ slots that correspond to $D_j (j \in [1, d])$.

The classification results $C_w(\vec{x})$ for the given input $\vec{x}$ are constructed as R_1 or R_2, as shown in Fig. 6. Here, $C_w(\vec{x})$ is only represented as an integer in [1, t] because [1, t] is mapped to a set of output classes of t-elements. To represent $C_w(\vec{x})$ in a ciphertext, we put the bit value 1 into the $((C_w(\vec{x}) - 1) \cdot u + 1)$ th slot, and set the other slots to zero among all $t \cdot u$ slots used in the ciphertext. This is shown in R_1. This form is used for server-centric classification. In the case of user-centric classification, we process R_1 first. Further processing is achieved by generating a random number $r \in [0, \lfloor s/u \rfloor]$ and rotating right by $r \cdot u$ slots to hide the true classification result from the server. The outcome of this operation is shown as R_2 in Fig. 6.

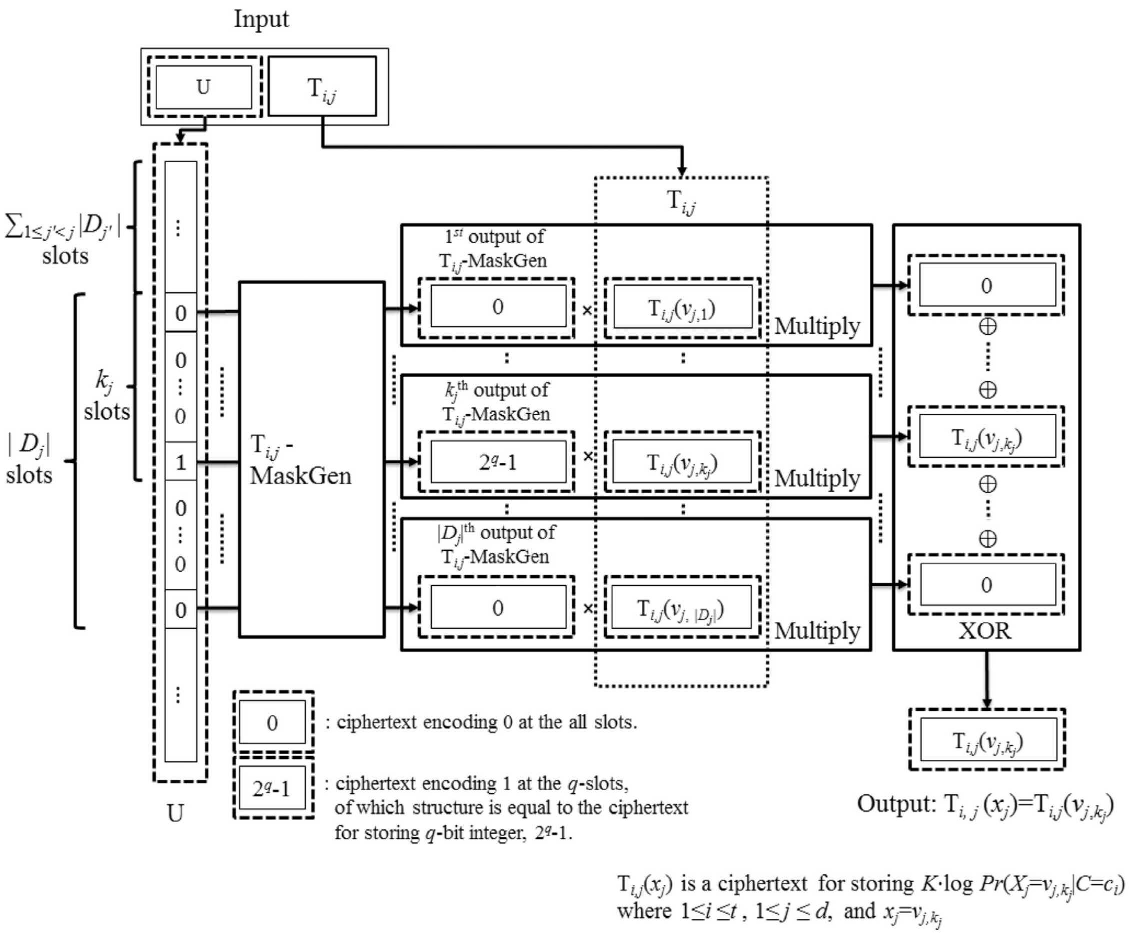

Input
U
T_{i,j}
∑_{1≤j'<j} |D_{j'}| slots
k_j slots
|D_j| slots
0
1
U
T_{i,j} - MaskGen
1st output of T_{i,j}-MaskGen
k_j th output of T_{i,j}-MaskGen
|D_j|th output of T_{i,j}-MaskGen
2^q-1
T_{i,j}(v_{j,1})
T_{i,j}(v_{j,k_j})
T_{i,j}(v_{j,|D_j|})
Multiply
XOR
: ciphertext encoding 0 at the all slots.
: ciphertext encoding 1 at the q-slots, of which structure is equal to the ciphertext for storing q-bit integer, 2^q-1.
Output: T_{i,j}(x_j)=T_{i,j}(v_{j,k_j})
T_{i,j}(x_j) is a ciphertext for storing K·log Pr(X_j=v_{j,k_j}|C=c_i) where 1≤i≤t, 1≤j≤d, and x_j=v_{j,k_j}

Fig. 10. Obtaining $T_{i,j}(v_{j,k_j})$ with the SELECTOR module ($v_{j,k_j} \in D_j$).

5.2.2. Proposed NB protocol

In this subsection, we describe the proposed privacy-preserving NB protocol. We first provide an overview of the proposed protocol, then proceed to the server-centric and user-centric protocols. A schematic view of both protocols is given in Fig. 7. Finally, we detail the components used to build the proposed protocol (Fig. 8).

5.2.3. SELECTION circuit

The SELECTION circuit takes a table *T* and a user input *U* and outputs a table *XT*, where each element $XT[i][j]$ corresponds to the encryption of $T_{i,j}(x_j) \triangleq K \cdot \log Pr[X_j = x_j|C = c_i]$ $(i \in [1, t], j \in [1, d])$. Each $XT[i][j]$ can be obtained by running the SELECTOR module with *U* and every $T_{i,\,j}$ in T $(i \in [1, t], j \in [1, d])$, as shown in Fig. 9.

Fig. 10 illustrates the SELECTOR module. It shows the user input corresponding to the element of D_j, which is v_{j,k_j} in the figure. To process this, the module extracts the corresponding $T_{i,j}(v_{j,k_j})$ $(V_{j,k_j} = x_j)$ through a number of steps. This process is repeated for all *i, j*.

For the bit values of *U*, there is a portion used to represent an element in D_j that is composed of $|D_j|$ slots. To represent v_{j,k_j}, only the k_jth slot is set to 1, while the other values are set to 0 among the $|D_j|$ slots. These $|D_j|$ slot values are given as the input to the MASKGEN function. In the figure, MASKGEN outputs $|D_j|$ ciphertexts of which only the k_jth contains *MAX*, while the others contain zeros. They are multiplied by $T_{i,j}(v_{j,1}), \cdots, T_{i,j}(v_{j,|D_j|})$, respectively, and these multiplication results are combined by running XOR operations. Thus, we obtain the final output.

Fig. 11 shows the implementation of the MASKGEN function. In this implementation, we use the ASSIGN function (shown in Fig. 12) to extract the bit value corresponding to each slot value. The corresponding value is then moved to the slot at the lowest position using the appropriate shift operation. Finally, the SLOTCOPY function (shown in Fig. 13) copies the corresponding bit value to other slots, ultimately generating the results of the MASKGEN function.

Note that all of the above steps work with ciphertexts. Therefore, neither the **Server** nor **Client** performing these steps knows any information about the values being processed, apart from what they originally have as plaintexts.

5.3. CLASSIFIER circuit

In the CLASSIFIER circuit, the final classification result is obtained by using the *XT* table, derived from the SELECTION circuit and the *S* table, which is information from another model. Fig. 14 shows this module. As shown in the figure, we can obtain $R[i] = K \log Pr[C = c_i]$ $+ \Sigma_{j=1}^{d} K \log Pr[X_j = x_j|C = c_i]$ $(= S[i] + \Sigma_{j=1}^{d} XT[i][j])$ by running the MULTIPLEADDER circuit with $S[i]$ and $XT[i][1], \cdots, XT[i][d]$. After obtaining all $R[i]$ values $(i = 1, \cdots, t)$, the classification result R_1 can be derived by running the ARGMAX module with all $R[i]$. The client obtains the classification result by decrypting R_1.

To run this circuit, we need to extract $S[1], \cdots, S[t]$ values that reside in table *S* of packed ciphertexts. Each $S[i]$ is the encryption of $K \log Pr[C = c_i]$, which is of *q*-bit length, and is of the form described in Section 5.2.1.

We describe the details of the extraction procedure. Suppose the packed table *S* is composed of $\lceil q \cdot t/s \rceil$ ciphertexts $S_{packed}[1]$, ⋯, $S_{packed}[\lceil q \cdot t/s \rceil]$. We first calculate the ciphertext $S_{unpacked}[i][j]$ for all $i \in [1, t]$, $j \in [1, q]$, where $S_{unpacked}[i][j]$ contains the *j*’s bit of $S[i]$ in

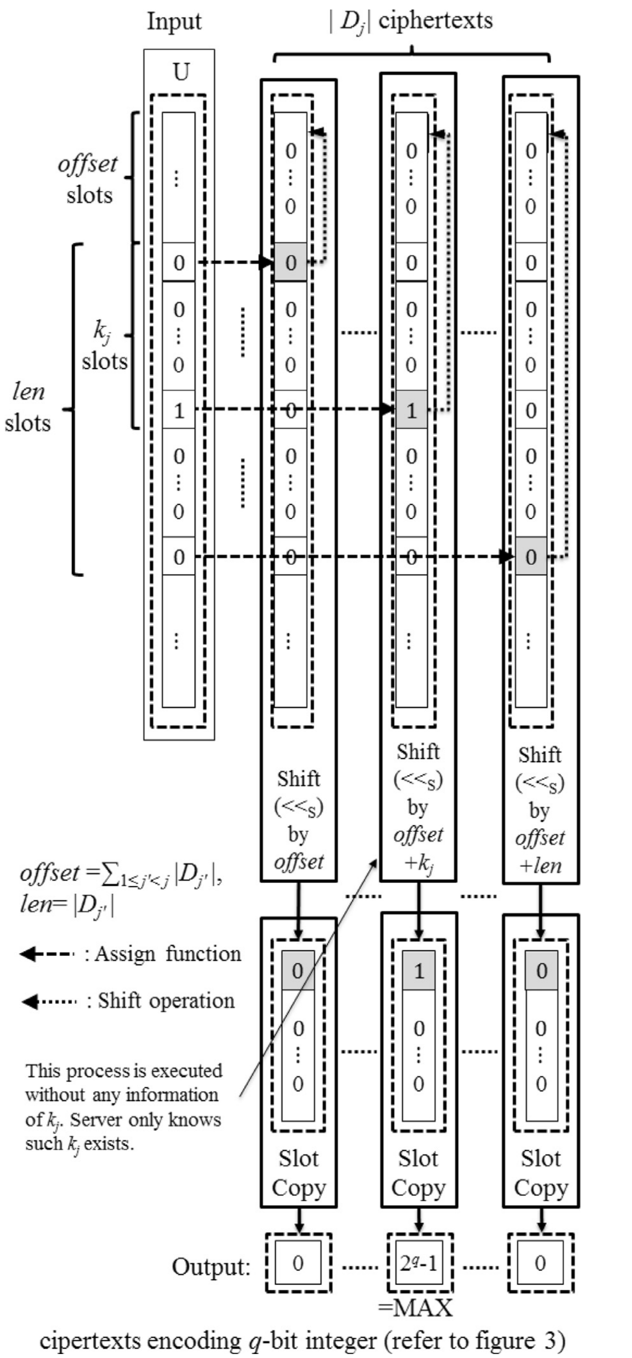

Input
| D_j| ciphertexts
U
offset slots
k_j slots
len slots
0
1
Shift ($<<_S$) by offset
Shift ($<<_S$) by offset $+k_j$
Shift ($<<_S$) by offset $+len$
offset $=\sum_{1\le j'<j}|D_{j'}|$,
len$=|D_{j'}|$
: Assign function
: Shift operation
This process is executed without any information of k_j. Server only knows such k_j exists.
Slot Copy
Slot Copy
Slot Copy
Output:
0
2^q-1
=MAX
0
cipertexts encoding q-bit integer (refer to figure 3)

Fig. 11. MaskGen module processing v_{j,k_j}.

the $j\cdot u$th slot and the other slots are set to zero. We run the following procedure to calculate it:

1. Initialize $S_{unpacked}[i][j]$ to set all the slots have zero bit-value.
2. Run Assign($S_{unpacked}[i][j]$, $S_{packed}[\lceil q\cdot i/s\rceil], (q\cdot(i-1)+j) \mod s$).
3. $S_{unpacked}[i][j] \leftarrow S_{unpacked}[i][j] >>_S ((j-1)\cdot u-(q\cdot(i-1)+j)) \mod s$

The explanation of Assign function is given in Fig. 12 and Appendix A. After the above steps are completed for all $j\in[1,q]$, we can construct $S[i]$ by the following calculation:

$$S[i] \leftarrow S_{unpacked}[i][1] \oplus S_{unpacked}[i][2] \oplus \cdots \oplus S_{unpacked}[i][q]$$

We can apply this procedure to the packed table T in order to extract all $T_{i,j}$s where $i\in[1,t]$ and $j\in[1,d]$.

Because the Classifier circuit is implemented using FHE operations, an inefficient design would make it necessary to execute a recryption operation, which results in a very long execution time. To avoid this, we minimize the multiplicative circuit depth by efficiently implementing the MultipleAdder and ArgMax modules. The pseudo-code for the Classifier circuit is provided in the appendix.

Fig. 15 illustrates the MultipleAdder circuit. This circuit takes an array of ciphertexts and its length n, and returns the encryption of the summation of all elements. We utilize the method in [50] to construct the circuit efficiently.

The MultipleAdder consists of the Encrypted-K-S Adder, which adds two ciphertexts, and the WallaceTree circuit, which reduces the number of n-encrypted numbers to two. We implement the Kogge–Stone Adder [51] using FHE operations when the FHE ciphertexts arrive as inputs. We call this the Encrypted-K-S Adder. It has a multiplicative depth of $\lceil \log q+2\rceil$ when adding two q-bit encrypted numbers.

The WallaceTree circuit adds n ciphertexts of q-bit integers to generate two ciphertexts. Adding the resulting two ciphertexts results in the encrypted value of adding n numbers. The circuit is implemented using a full-adder circuit to add three q-bit encrypted numbers, and requires a multiplicative circuit depth of less than $\log 1.5n$.[3]

As a result, the MultipleAdder circuit requires a multiplicative depth of less than $\log 1.5n+\lceil \log q+2\rceil$ when adding n q-bit numbers. For details on the WallaceTree module and the Encrypted-K-S Adder module, see [50].

The ArgMax module takes an array IN of n q-bit integer ciphertexts and returns the index of the ciphertext with the largest value. The input of this module is the array of ciphertexts (IN) given by executing MultipleAdder in the previous step. The resulting value is represented by a ciphertext in which only the slot corresponding to the index of the ciphertext with the largest value among the input values has a bit value of 1. This can be implemented by reducing the depth of the multiplication circuit using the method in [52].

In this implementation, if the kth ciphertext has the largest value, only the $((k-1)*u+1)$ th slot is set to 1 in the resulting ciphertext. The main reason for this is that, if we generate the result in this format, we can use the **Rotate** operation in place of the **Shift** operation to reduce the multiplicative depth of the circuit. This is possible because ciphertexts containing the input numbers to be compared already store bit values at u slot intervals.

The structure of the ArgMax module is shown in Fig. 16. This process is divided into two steps: generating a two-dimensional array M, in which the elements are ciphertexts that have either 0 or 1 in their first slot, and the Max circuit, which produces a ciphertext from the constructed array M.

As a result of the first step, M becomes a two-dimensional array containing the results of a pair-wise comparison between the ciphertexts input to IN. For example, if $IN[i]>IN[j]$, the lowest slot of $M[i][j]$ is set to 1. Otherwise, $M[i][j]$ stores a value of 0. We do not calculate $M[i][j]$ if $i=j$. If $i<j$, the value is calculated by flipping the bit value in the lowest slot of $M[j][i]$ that has already been calculated. The Comparator module is used to generate M. This module receives two encrypted numbers and returns a ciphertext in which the lowest slot has a bit-value of 1 if the number contained in the first ciphertext is larger than that in the second. Otherwise, the lowest slot value of the returned ciphertext is set to 0. We implement this using some of the comparator implementations proposed in [43]. The difference is that the **Rotate** operation is applied, rather than the **Shift** operation. This reduces the required multiplication depth. The required multiplication depth for this module is $\lfloor \log q+1\rfloor$. The values of $M[i][j]$ ($i<j$ and $i,j\in[1,n]$) are calculated by performing $M[j][i]\oplus 1$.

In the second step, the resulting M is used to obtain the index of the ciphertext with the largest value. This process is done by applying the

[3] We ensure no carry is generated in every addition by making q sufficiently long.

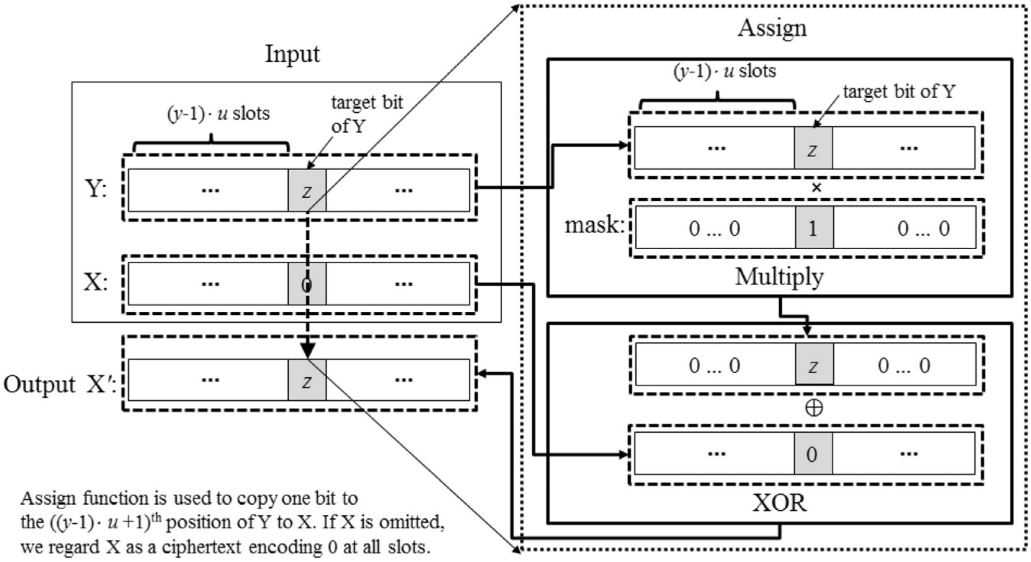

Input
Assign
(y-1)· u slots
target bit of Y
Y:
X:
Output X′:
z
0
...
mask:
0 ... 0
1
×
Multiply
⊕
XOR
Assign function is used to copy one bit to the ((y-1)· u +1)th position of Y to X. If X is omitted, we regard X as a ciphertext encoding 0 at all slots.

Fig. 12. Assign module.

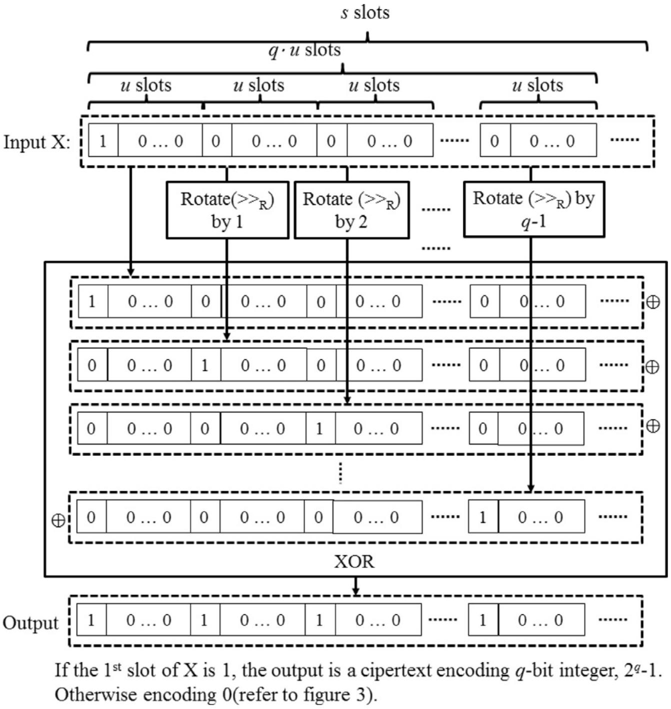

s slots
q · u slots
u slots
Input X:
Rotate(>>R) by 1
Rotate (>>R) by 2
Rotate (>>R) by q-1
XOR
Output
1
0 ... 0
⊕
If the 1st slot of X is 1, the output is a cipertext encoding q-bit integer, 2q-1. Otherwise encoding 0(refer to figure 3).

Fig. 13. Illustration of SlotCopy module working with the encrypted input whose lowest slot is set to 1.

following calculation repeatedly for all $i \in [1, n]$ with the ciphertext R_1, which is initialized to zero beforehand:

$$R_1 \leftarrow R_1 \oplus ((M[i][1] \cdot M[i][2] \cdot .... \cdot M[i][i-1] \cdot M[i][i+1] \cdot ... \cdot M[i][n]) >>_R (i-1))$$

As a result of the above computation, we can obtain ArgMax, whose $(u * (k-1)+1))$ th slot is set to 1. The other slots are set to 0 if *IN*[*k*] has the largest integer in the array. When implemented in this way, the multiplication depth required for the entire ArgMax circuit is $((\log q + 1) + \log n)$. This is much smaller than $((n-1)((\log q + 1) + 1)$, which is the required multiplicative circuit depth of an intuitive method in which the index of the maximum value is obtained by running the comparator circuit $(n-1)$ times in series. For convenience, we provide an illustration of the Max module in the appendix.

6. Performance evaluation

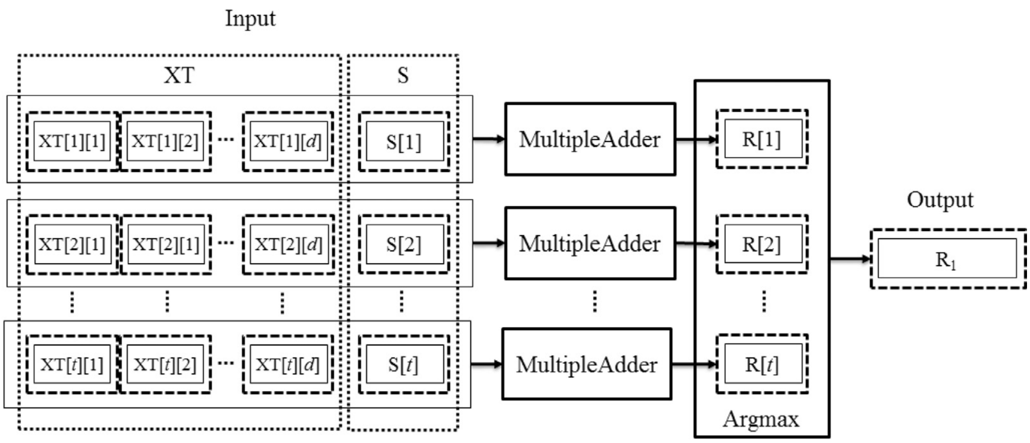
In this section, we analyze the experimental performance of the proposed privacy-preserving NB classification protocols based on FHE operations. The hardware specifications used to conduct the experiments were as follows: Server: Intel (R) Xeon (R) ES-1650 v3 @ hexa-core processor with 64 GB of RAM running Linux Ubuntu 16.04 LTS; Client: Intel i7-6700 3.40 GHz @ quad-core processor with 16 GB RAM running Linux Ubuntu 14.04 LTS. The proposed method was implemented using HELib [9–11]. To the best of our knowledge, HELib is the most efficient library among the existing FHE implementations. We used the pthread library version 2.24 to implement the multi-threaded version of the proposed protocols.

For the experiments, we set the parameter values by considering the following. First, priority was given to providing a sufficient number of slots s in a ciphertext. We considered parameters to support 80-bit or higher security. For efficiency, we selected those that maximized the multiplicative depth of the circuit that can be performed without recryption. In addition, for ease of use, the rotation (automorphism) operation was employed. The parameter values are presented in Table 3.

In the proposed protocols, the parameters related to the input data are q, which pertains to the precision of the input data, d, which is the dimension of the feature vector, and t, which is the class number of the classification results. In this subsection, we examine the relationship between these parameters and the performance of the core modules of the proposed method. The performance evaluation reported in this subsection was conducted in the server environment.

6.1. Performance of the subroutines

Encrypted-K-S Adder, Comparator, and WallaceTree: Fig. 17 shows the performance of the WallaceTree, Encrypted-K-S-Adder, and Comparator modules with respect to q. As shown in the figure, the execution time of Encrypted-K-S-Adder and Comparator increases linearly with $\log q$. The execution time of WallaceTree is not dependent on q, because the number of values to be added using WallaceTree is fixed to d, and all values can be contained in a single ciphertext, even when $q = 128$. Thus, the required FHE operations do not change with respect to q.



XT[i][j] is a ciphertext for storing $K\cdot\log Pr(X_j=x_j|C=c_i)$ where $1\le i\le t$ and $1\le j\le d$.
S[i] is a ciphertext for storing $K\cdot\log Pr(C=c_i)$ where $1\le i\le t$.
R[i] is a ciphertext for storing $\sum_{1\le j\le d}(K\cdot\log Pr(X_j=x_j|C=c_i)+K\cdot\log Pr(C=c_i))$ where $1\le i\le t$.

Fig. 14. Illustration of the CLASSIFIER circuit.

Table 3
HELib parameters chosen for the experiment.

Cyclotomic ring (m)	Lattice dimension ($\phi(m)$)	Plaintext space	Number of slots in a ciphertext	Security level	Maximum multiplicative depth to reach the first recryption	Size of compressed ciphertext	Public key size	Private key size
31775 =52*31*41	24000	GF(2^{20})	1200	93	24	4.3MB	67.5MB	67.03MB

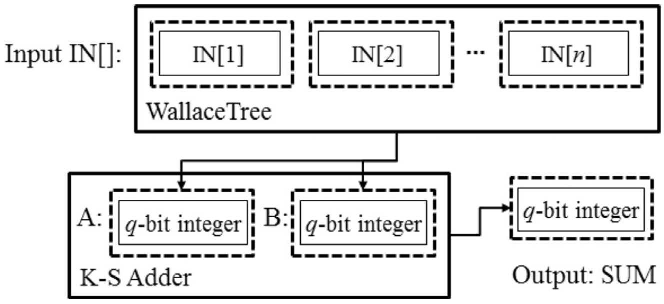


Fig. 15. Illustration of the MULTIPLEADDER circuit.

Fig. 18 shows the performance of each module with respect to d. In the proposed method, as the value of d increases, the number of objects to be added by WallaceTree increases linearly. This is because it is used to calculate the sum of $\log Pr[C=c_j]+\sum_k \log Pr[X_k=x_k|C=c_j]$ for each j. Therefore, as shown in the figure, the execution time of WallaceTree increases in proportion to $\log_{1.5}d$.

ARGMAX and multi-threading issue: the ARGMAX module finds the largest among t values of $\log Pr[C=c_j]+\sum_k \log Pr[X_k=x_k|C=c_j]$ on each $j\in[1,t]$, which is the probability that each output class is the classification result. As described in the previous section, our implementation focuses on minimizing the multiplicative depth of the circuit. The cost of this is an increase in the number of comparator module executions. The circuit requires $(t-1)t/2$ COMPARATOR module executions and a single MAX circuit execution. Therefore, if recryption does not occur, the majority of the execution time is the $(t-1)t/2$ COMPARATOR operations. Fortunately, these comparisons can be performed independently, so multi-threading can be used to improve the performance in a multi-core environment.

Furthermore, the MAX circuit can be executed in parallel because it computes $(M[1][2]\cdot M[1][3]\cdot\dots\cdot M[1][t])\oplus(M[2][1]\cdot M[2][3]\cdot\dots)\oplus\dots\oplus(M[t][1]\cdot\dots\cdot M[t][t-1])$. We can assign separate threads to calculate each multiplicative term to improve the speed.

Table 4
Subroutines where multi-threading is used and the number of threads used in each subroutine.

Subroutines	SELECTOR	MULTIPLEADDER	ARGMAX
# of threads	t	t	$\binom{t}{2}$

Similar to the above case, the MULTIPLEADDER module obtains the probability that each class is the output of classification. This is the input for the ARGMAX module, and can be determined independently for each resulting class. Therefore, if these MULTIPLEADDER modules are run in t-multiple threads at the same time, the overall execution time can be reduced. Additionally, in the case of the server-centric classification, the execution of the SELECTION module, which selects the input to the MULTIPLEADDER modules (the XT table) can also be processed in parallel with t-multiple threads.

Table 4 summarizes the subroutines in which multi-threading is applied and the number of threads used in each subroutine. Note that the numbers are dependent on the number of classes t. In our experiments, t was set to 2 and 4. Because the number of threads is less than the number of cores, the actual number of cores used is the same as the number of threads in the server-centric protocol. However, in the user-centric implementation, the number of cores was four when $t=4$. The use of hyper-threading allowed two threads to be assigned to a core.

Fig. 19 shows the execution results of various modules with multi-threading. This experiment was performed in the server environment, where 12 threads were available for computation.

6.2. *Classification performance*

To evaluate the performance of the proposed classifier, the Breast Cancer Data Set and the Car Evaluation Data Set from the UCI machine learning repository [14] were classified. We assumed that the model values required for classification were generated through training during the initial process. The probabilities were represented as 32-bit values.

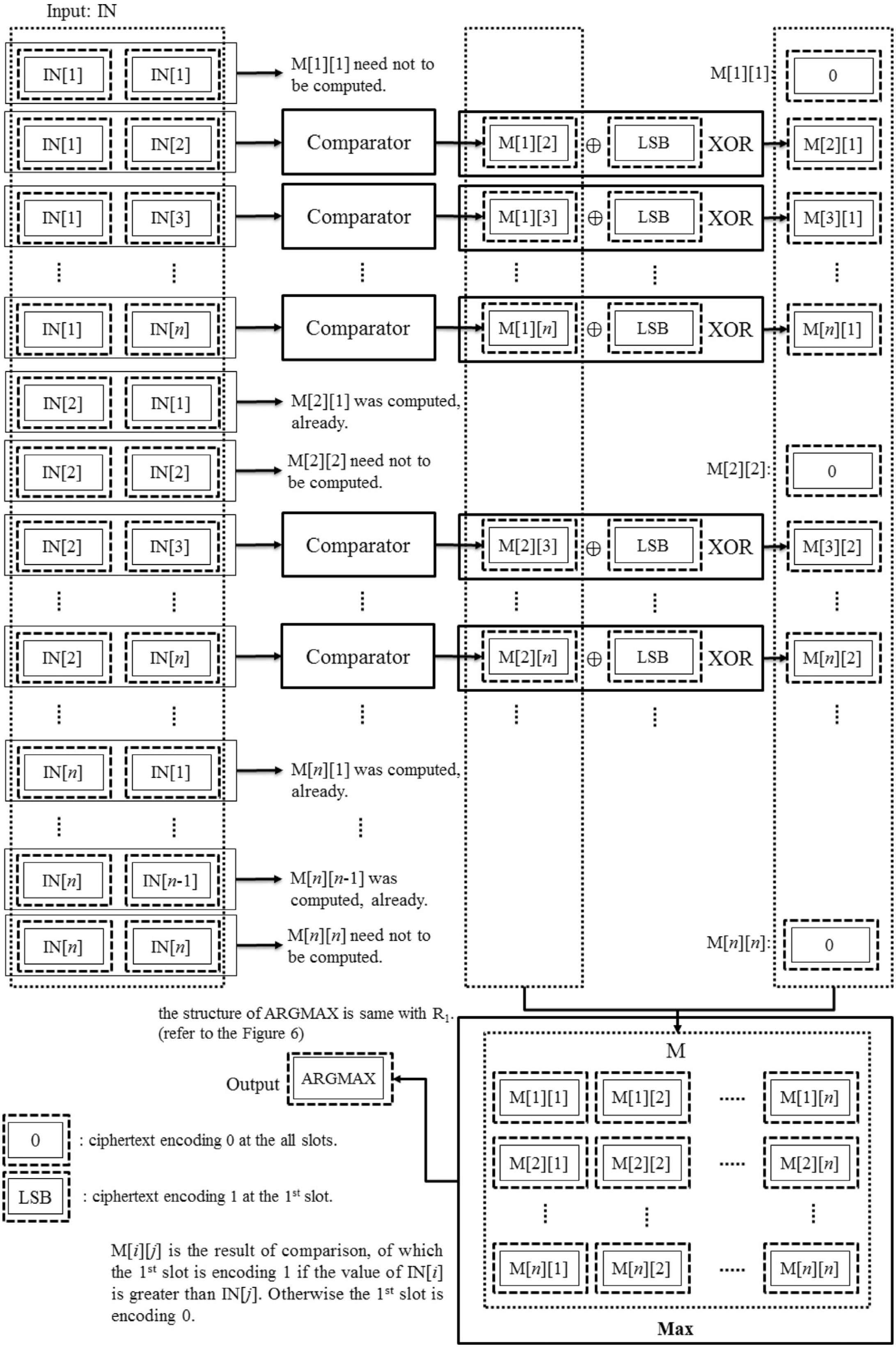

Input: IN
IN[1]
IN[2]
IN[3]
IN[n]
IN[n-1]
M[1][1] need not to be computed.
M[2][1] was computed, already.
M[2][2] need not to be computed.
M[n][1] was computed, already.
M[n][n-1] was computed, already.
M[n][n] need not to be computed.
Comparator
M[1][2]
M[1][3]
M[1][n]
M[2][3]
M[2][n]
⊕
LSB
XOR
M[1][1]:
0
M[2][1]
M[3][1]
M[n][1]
M[2][2]:
M[3][2]
M[n][2]
M[n][n]:
the structure of ARGMAX is same with R₁. (refer to the Figure 6)
Output
ARGMAX
M
M[1][1]
M[1][2]
M[1][n]
M[2][1]
M[2][2]
M[2][n]
M[n][1]
M[n][2]
M[n][n]
Max
: ciphertext encoding 0 at the all slots.
: ciphertext encoding 1 at the 1st slot.
M[i][j] is the result of comparison, of which the 1st slot is encoding 1 if the value of IN[i] is greater than IN[j]. Otherwise the 1st slot is encoding 0.

Fig. 16. Illustration of the *ArgMax* circuit.

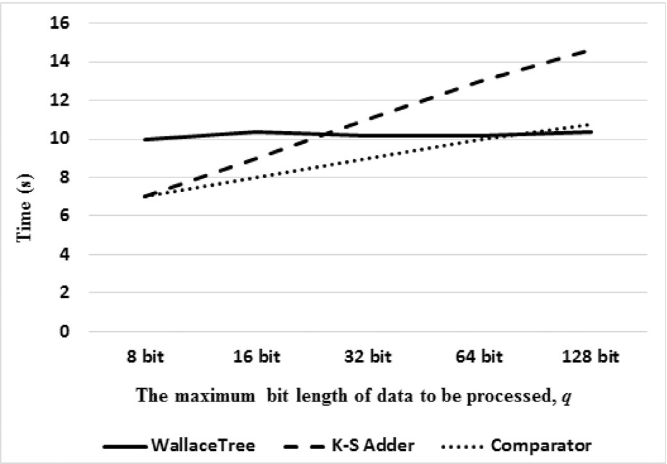


Fig. 17. Comparison of subroutine performance with $q = \{8, 16, 32, 64, 128\}$ $(d = 4, t = 2)$
.

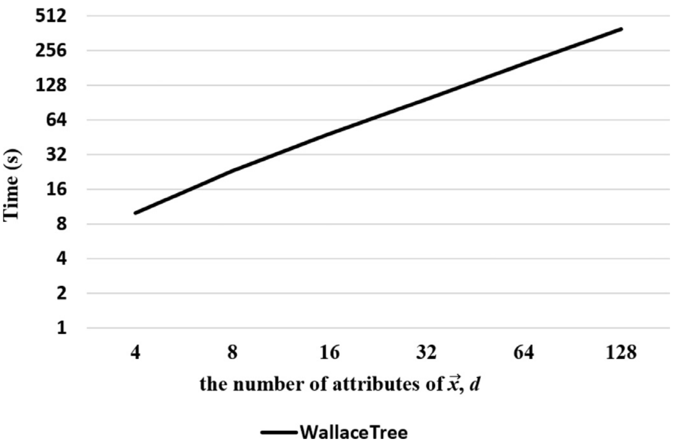


Fig. 18. Comparison of the performance of WALLACETREE with $d = \{4, 6, 8, 10, 12, 14, 16\}$.

For this experiment, we constructed an encrypted version of tables S and T. Because $\sum_{j=1}^{d} |D_j|$ is 90 and 21 in the Breast Cancer and Car Evaluation cases respectively, and the packing technique was applied, only one ciphertext was used to construct S in both cases, with five and three ciphertexts used to construct T in the Breast Cancer and Car Evaluation cases, respectively.

Tables 5 and 6 present the results of the server-centric and user-centric classifications, respectively. These results relate to only the computation time of the server and the client, respectively.

The execution time was approximately 60 s for the server-centric protocol and approximately 80 s for the user-centric protocol. The decryption time was approximately 0.2 s in the server environment and approximately 0.4 s in the client environment. Therefore, if the communication time is considered, the total execution time is expected to be approximately 70 s for the server-centric protocol and approximately 90 s for the user-centric protocol.

Table 7 compares the proposed protocol to existing privacy-preserving NB protocols [1,2,4] in terms of the number of decryptions required for the execution of the protocol and the number of interactions between the server and client. When analyzing the number of interactions, we assumed that messages that can be transmitted at the same time during the execution of the protocol were transmitted simultaneously. That is, we regarded such messages as one-time message transmissions.

According to the analysis results, the proposed protocol requires only one decryption and two interactions. In contrast, [2] uses commodity-based cryptography to share information using pre-distributed correlated data, meaning that no decryption is required during the protocol execution. However, many interactions between the server and client

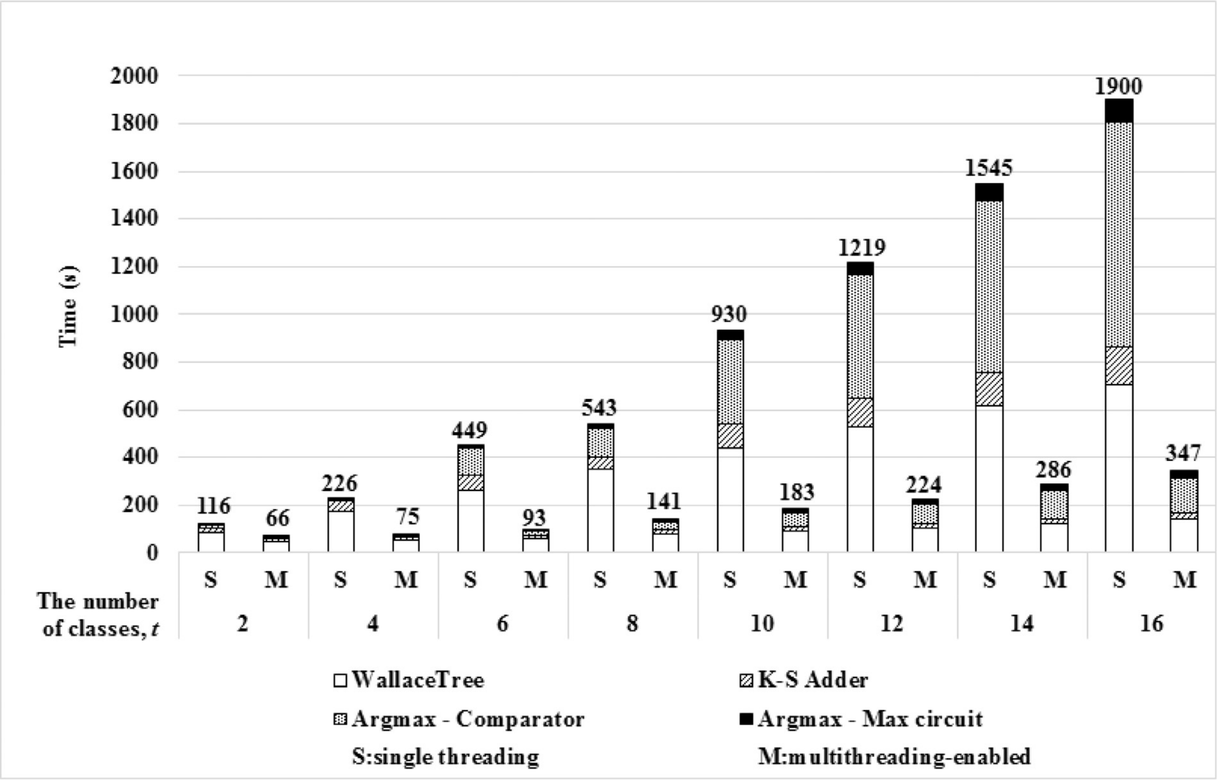


Fig. 19. Performance comparison between single and multi-threaded execution of subroutines ($q = 32,\ d = 16$).

Table 5
Evaluation results of server-centric classification, (COMP=COMPARATOR,q=32, S: server, C: client).

	Specification		Time for classification by Server (second)						Communication (# of ciphertext)	
Data set	Classes	Attributes	SELECTION	WALLACE	E-K-S-	ARGMAX		Total	S→ C	C → S
	t	d	circuit	TREE	ADDER	COMP	MAX			
Breast Cancer	2	9	22	26	10	9	1	68	1	1
Car Evaluation	4	6	21	19	12	14	3	69	1	1

Table 6
Evaluation results of user-centric classification, (COMP=COMPARATOR,q=32, S: server, C: client).

	Specification		Time for classification by User (second)					Communication (# of ciphertext)	
Data set	Classes	Attributes	WALLACE	E-K-S-	ARGMAX		Total	S→ C	C → S
	t	d	TREE	ADDER	COMP	MAX			
Breast Cancer	2	9	47	22	15	1	85	1	1
Car Evaluation	4	6	38	24	23	4	89	1	1

Table 7
Comparison between existing privacy-preserving NB classifiers and the proposed classifier (S: server, C: client).

	Dataset	# of decryptions	# of interactions between S and C
[1]	Breast Cancer	3	10
	Car Evaluation	9	26
[2]	Breast Cancer	–	75
	Car Evaluation	–	79
[4]	Breast Cancer	3	6
	Car Evaluation	5	8
Proposed (server-centric/ user-centric)	Breast Cancer	1	2
	Car Evaluation	1	2

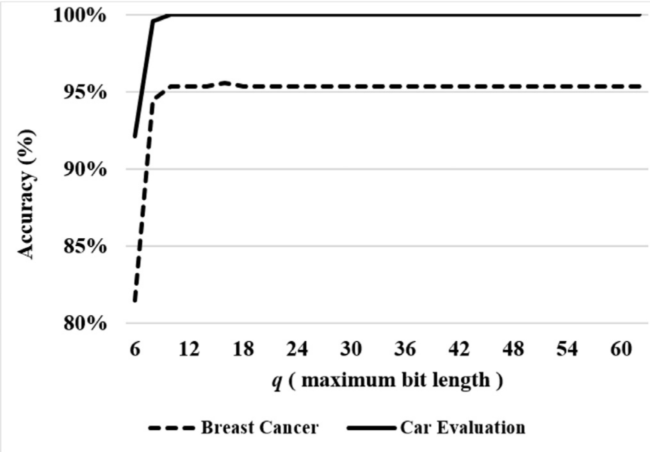


Fig. 20. Relation between q and accuracy of NB classification.

are required when sharing information. In both [2] and [4], the required interactions and decryption times increase in proportion to the number of resultant classes for classification (i.e., t).

Fig. 20 shows the accuracy of the NB protocol classification results for various q. Clearly, the accuracy increases as the bit-length of q increases, up to a value of $q = 7$ in the 'Car Evaluation' case and $q = 4$ in the 'Breast Cancer' case. From this, we can conclude that $q = 32$ is sufficiently high in our setting.

Table 8
Comparison of the resiliency to attacks with quantum computers.

	Used crypto-system	Security assumption	Quantum security
[1]	Paillier [53]	Decisional Composite Residuosity [50]	N
	Goldwasser–Micali[54]	Decisional Composite Residuosity	N
[2]	Paillier [53]	Decisional Composite Residuosity	N
[4]	Secure Multi-party Computation [25]	Decisional Composite Residuosity	N
	Oblivious Transfer [56]	RSA [55]	N
	Oblivious Transfer [57]	Decision Diffie–Hellman[58]	N
Ours	GHS-FHE [9–11]	Ring LWE [59]	Y

Table 8 describes the cryptosystems needed to construct existing privacy-preserving NB classifiers and the proposed protocol. It also shows their underlying security assumptions. The final column of the table shows whether each of these assumptions still holds against quantum computing attacks. Reference [4] describes techniques used to share the initial co-related data between server and client. As this table indicates, the security assumptions of all cryptosystems are not safe from quantum computing attacks, with the exception of Ring LWE [58], which is used in our proposed method.

7. Conclusion

In this paper, we have proposed two non-interactive privacy-preserving classification models and implemented them using an FHE-based privacy-preserving NB classifier. Various efficiency-enhancing techniques were applied to implement the classification protocols such that all the computational circuits were constructed exclusively with operations in FHE.

Unlike existing techniques, the proposed method minimizes the communication between the server and the user (i.e., client). It is also advantageous in terms of security, because there is no decryption operation during the protocol execution. In addition, our proposal is robust to attacks using quantum computers.

We implemented the proposed method using HELib. The implementation results were verified by performing experiments in a conventional server and a client environment. The experimental results show that the two proposed models required approximately 70–90 s to classify test datasets. Although this classification speed may not appear to be practical, we believe that the performance will improve with advances in FHE technology. Future studies will attempt to implement various classification algorithms with FHE operations to achieve practical privacy-preserving classification.

Appendix A. Selection module and its subroutines

Selection module

• Input

U: ciphertext containing a feature vector $\vec{x}$ (refer to Fig. 5)
T: set of the ciphertext of $K \cdot Pr[X_j = v|C = c_i]$ for all $i \in [1, t]$, $j \in [1, d]$, and $v \in D_j$
D: $\{D_1, D_2, \cdots, D_d\}$, t: number of elements in the output class
d: number of features that constitutes $\vec{x}(= |D|)$
q: maximum bit length of values being processed

• Output

XT: $t \times d$ array whose elements $XT[i][j]$ have $T_{i,j}(x_j)$=$K \cdot Pr[X_j = x_j|C = c_i]$

```
function Selection($XT$, $U$, $T$, $D$, $t$, $d$,$q$)
    //For multi-threading, $t$-threads start instead of for $i$
    for $i \leftarrow 1$ to $t$ do
        $offset \leftarrow 0$
        for $j \leftarrow 1$ to $d$ do
            $len \leftarrow |D_j|$
            //offset: first slot's position used to express $x_j$ in $D_j$
            //len: number of slots used to express $x_j$
            Select($XT[i][j]$,$U$,$T_{i,j}$,$i$,$j$,$offset$,$len$,$d$,$q$)
            $offset \leftarrow offset + len$
        end for
    end for
end function
```

Selector module

• Input

$T_{i,j}$: set of ciphertexts containing the integer values $K \cdot \log Pr[X_j = v|C = c_i]$ for every $v \in D_j$, where $i \in [1, t]$ and $j \in [1, d]$.
U, $offset$, len, d, q: same as in the description of Selection module.
i: value in $[1, t]$, j: value in $[1, d]$.

• Output

$XT[i][j]$: ciphertext of $K \cdot Pr[X_j = x_j|C = c_i]$

```
function Selector($XT[i][j]$, $U$, $T_{i,j}$, $i$, $j$, $offset$, $len$, $d$,$q$)
    // $TM[1..len]$ is initialized to 0
    // MaskGen returns an array of ciphertext $TM[]$ where $TM[k] = MAX$
    // if $(offset+k)$'s slot contains 1 in $U$, and $TM[k] = 0$ otherwise.
    MaskGen($TM[]$,$U$,$offset$,$len$,$q$)
    for $k \leftarrow 1$ to $len$ do
        $TM[k] \leftarrow TM[k] \cdot T_{i,j}[k]$        ▷ $T_{i,j}[k] = K \cdot \log Pr[X_j = v_{j,k}|C = c_i]$
    end for
    $XT[i][j] \leftarrow TM[1] \oplus TM[2] \oplus \cdots \oplus TM[k]$
end function
```

MaskGen module

• Input

$offset$: start position of the slots used to represent x_j in D_j
len: $|D_j|$

• Output

$TM[1..len]$: array of ciphertext $TM[]$, where $TM[k] = MAX$ if $(offset + k)$'s slot contains 1 in U, and $TM[k] = 0$ otherwise

```
function MaskGen($TM[]$, $U$, $offset$, $len$, $q$)
    for $l \leftarrow offset + 1$ to $offset + len$ do
        Assign($TM[i - offset]$,$U$,$l$)
        $TM[i - offset] \leftarrow TM[i - offset] <<_S (i - 1)$
        //SlotCopy spreads the bit in the lowest slot in $TM[i - offset]$ into all of the other slots
        SlotCopy($TM[i - offset]$,$q$)
    end for
end function
```

Assign module

• Input

Y: ciphertext that contains the bit value to be copied to X in the y-th slot
y: position of the slot that contains the bit value to be copied in Y

• Output

X: ciphertext whose y-th slot bit-value is updated to Y's bit-value in the same slot

```
function Assign($X$, $Y$, $y$)
    // Assuming $ptxt$ is set to zero in all slots
    // Setting $y$-th bit-value of $ptxt$ to 1
    $ptxt[y] \leftarrow 1$
    // Performing nullified-encryption with $ptxt$, assuming that a suitable $pk$ is provided to $Encrypt$
    $mask \leftarrow Encrypt(ptxt)$
    $X \leftarrow X \oplus (mask \cdot Y)$
end function
```

SlotCopy module

• Input

X: ciphertext that contains either 0 or 1 in the lowest slot

• Output

X: updated ciphertext, where all of the next $q - 1$ slots' bit values from the lowest slot become the same as the lowest slot's bit value

```
function SlotCopy($X$, $q$)
    for $i \leftarrow 1$ to $q - 1$ do
        // $Y[1..q - 1]$ is an array of ciphertexts
        $Y[i] \leftarrow X >>_R i$
    end for
    $X \leftarrow X \oplus Y[1] \oplus \cdots \oplus Y[q - 1]$
end function
```

Appendix B. Classifier circuit and its subroutines

The section describes the **Classifier** circuit and its subroutines Fig. B.21.

Appendix C. Privacy proof of the proposed protocol

As mentioned in Section 4, the proposed protocol must satisfy the privacy-preserving requirement. We provide a formal proof in this section. We can informally list the privacy requirements of the proposed protocol as follows:

- The server is able to access the model information (=w) and cannot derive the user's (i.e. client's) input (=$\vec{x}$) while performing the protocol.
- The user is unable to extract model information from the information he/she can obtain while running the protocol.

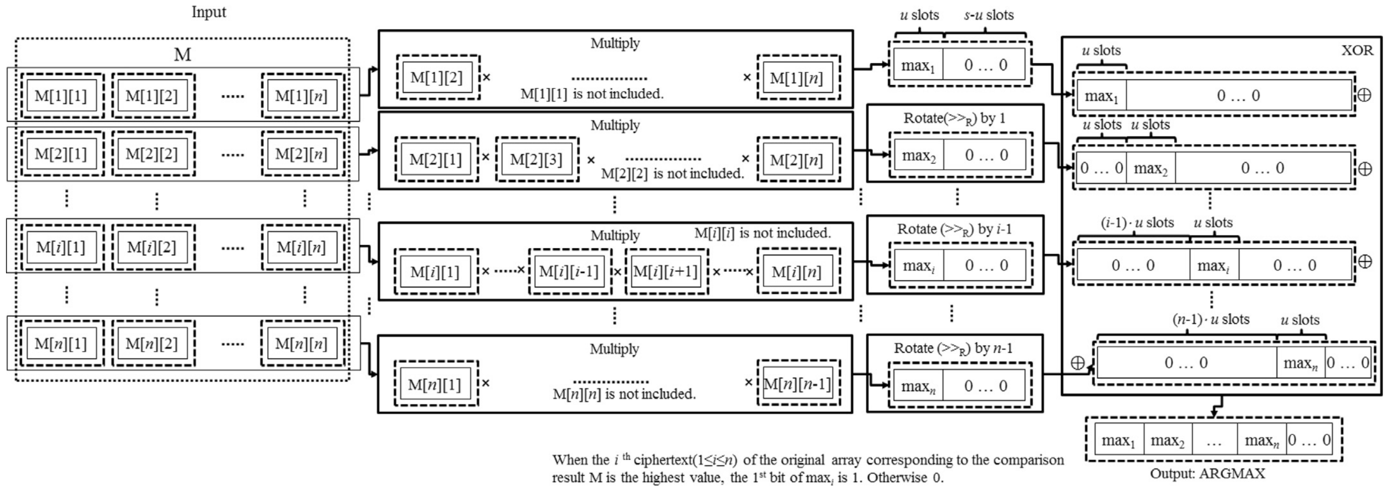


Fig. B.21. Illustration of MAX circuit.

Classifier module

- **Input**

$XT[][]$: $t \times d$ array whose element $XT[i][j]$ has $T_{i,j}(x_j)=K \cdot Pr[X_j = x_j | C = c_i]$. Refer to the **Selection** module.
S: array of ciphertexts for storing the integer value of $K \cdot \log Pr[C = c_i]$ for all $i \in [1, t]$
q: maximum bit length of values being processed

- **Output**

R_1: resulting ciphertext for storing the classification result (refer to R_1 in 6)

```
function Classifier($R_1$, $S$, $XT$, $q$)
    //For multi-threading, $t$-threads start instead of for $i$
    for $i \leftarrow 1$ to $t$ do
        for $j \leftarrow 1$ to $d$ do
            //$IN[1..d+1]$ is an array of ciphertexts
            $IN[j] \leftarrow XT[i][j]$
        end for
        $IN[d+1] \leftarrow S[i]$
        //Adding up all elements in $IN$ and assigning the
    result to $R[i]$
        MultipleAdder($R[i]$,$IN$,$d+1$)
    end for
    //Finding the maximum element in $R[]$
    ArgMax($R_1$, $R[i]$, $t$)
end function
```

ArgMax module

- **Input**

IN: array of ciphertexts that contain q-bit integers
n: size of IN, MAX: ciphertext representing $2^q - 1$, by assigning 1 to the 1st, $(u+1)$-th, $\cdots$, $((q-1)u+1)$-th slots
LSB:ciphertext in which the first slot is set to 1 and the others are set to 0
q: maximum bit length of values being processed

- **Output**

argmax: ciphertext in which the $(u*(k-1)+1)$-th slot is set to 1 and the other is set to 0, where $IN[k]$ contains the maximum integer among all the integers in IN

```
function ArgMax(argmax,$IN$,$n$,$MAX$,$LSB$,$q$)
    // $M[1..n][1..n]$: 2D array of ciphertexts
    // $M[i][j]$ stores the result for comparison between
    // $IN[i]$ and $IN[j]$
    //For multi-threading, $t(t-1)/2$-threads
    //start instead of for $i, j$
    for $i \leftarrow 1$ to $n$ do
        for $j \leftarrow i+1$ to $n$ do
            // Comparator sets $M[i][j]$ as 1 in the 1st slot
            // if $IN[i] > IN[j]$
            Comparator($M[i][j]$,$IN[i]$,$IN[j]$,$MAX$,$LSB$,$q$)
            $M[j][i] \leftarrow LSB \oplus M[i][j]$
        end for
    end for
    //Max let argmax have the index of the largest value
    //in $IN$.
    Max(argmax,$M$,$n$)
end function
```

To describe the above requirements formally, we build up definitions of the privacy requirements. We employ the simulator in our definitions of privacy. The simulator is a hypothetical object for the simulation of participants in a protocol [60]. In a security proof, the simulator simulates a designated participant in a two-party protocol. The simulator runs a two-party protocol with the other participant. As a result of the execution of the protocol, the simulator should provide a valid-looking output to the other participant using only the public information. If this is possible, then we can prove that the other participant cannot derive any private information from the party that is simulated by the simulator.

We provide a formal definition of privacy for the proposed protocol as follows:

Definition 1 (User Input Privacy)**.** If the server cannot distinguish with non-negligible probability [60] whether it is interacting with a user or with a simulator running a two-party protocol using only the public information available when executing the protocol, we can say that the protocol supports the privacy of the user input. The term 'distinguish' refers to computational indistinguishability [60]. The probability is drawn over the choice of both the user and the server in the protocol.

Definition 2 (Server Model Privacy)**.** If the user cannot distinguish with non-negligible probability [60] whether he/she is interacting with a server or a simulator running a protocol using only public information available when executing the protocol, we can say that the protocol supports the privacy of the model. The term 'distinguish' again refers to computational indistinguishability [60]. The probability is drawn over the choice of both the user and the server in the protocol.

We prove that the proposed protocol ensures the privacy of the user input and the model based on the above definitions. We use the code-based security proof technique [61].

Theorem 1. *If the proposed server-centric protocol uses a CPA-secure FHE scheme and the server and user's computational powers are at most the same as a conventional CPA adversary [62], then the user input privacy and server model privacy requirements defined in Definitions 1 and 2, respectively, are satisfied.*

Proof. proof of Theorem] 1 We prove the theorem by constructing simulators to show that the proposed protocol satisfies Definitions 1 and 2.

We first explain the real execution environment shown as World AA in Fig. C.22. In the server-centric protocol, the server has the model *w* and the user has the input $\vec{x}$, and the public key of the user pk_u is shared between them. The private key of the user sk_u is only given to the user. In this setting, the user sends $\vec{x}$ encrypted by his/her public key to the server. The server computes the encrypted $C_w(\vec{x})$ and sends the result to the user. Finally, the user decrypts the result with his/her private key to obtain $C_w(\vec{x})$. In this scenario, the server can access the ciphertexts encrypted by pk_u and the model *w*.

(Proof of Definition 1) To prove that the proposed server-centric protocol satisfies Definition 1, we build a simulator that acts as the real user but that only has the public information, pk_u. This is shown as World AB in Fig. C.22. The simulator creates a random input vector $\vec{y}$, encrypts it with pk_u, and sends it to the server. After the server's classification computation, the simulator receives the encrypted $C_w(\vec{y})$. We now run a hypothetical game. As a setup, we uniformly randomly select the world from either World AA or World AB, and run the server-centric protocol. In this case, we claim that the probability of the server determining which world it is working in is negligible. Our claim holds if the underlying FHE scheme supports CPA security: the server cannot computationally distinguish whether it receives $c_{\vec{x}}$ or $c_{\vec{y}}$. Otherwise, the CPA-security of the underlying FHE scheme would be broken. Additionally, what the server can actually do with the received ciphertext is limited to what the CPA adversary can do with it. Therefore, our claim holds.

(Proof of Definition 2) Similar to the proof of Definition 1, we build a simulator in World BA that interacts with the user based solely on public information. We prove that the user cannot distinguish whether he/she is running on either World AA or World BA with non-negligible probability. The difference between World AA and World BA is that the user obtains $C_w(\vec{r})$ in World AA, but $C'_w(\vec{r})$ in World BA, where w' is

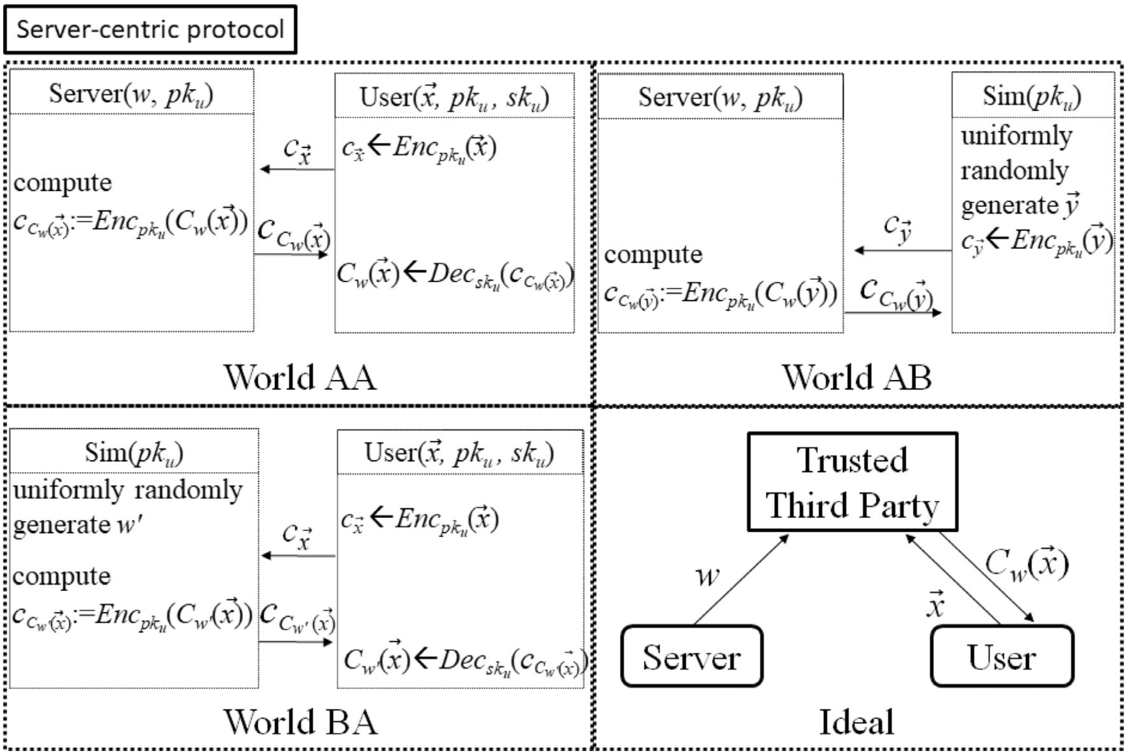

Server-centric protocol
Server(w, pk_u)
User($\vec{x}$, pk_u, sk_u)
$c_{\vec{x}}$
$c_{\vec{x}} \leftarrow Enc_{pk_u}(\vec{x})$
compute
$c_{C_w(\vec{x})} := Enc_{pk_u}(C_w(\vec{x}))$
$c_{C_w(\vec{x})}$
$C_w(\vec{x}) \leftarrow Dec_{sk_u}(c_{C_w(\vec{x})})$
World AA
Server(w, pk_u)
Sim(pk_u)
uniformly randomly generate $\vec{y}$
$c_{\vec{y}} \leftarrow Enc_{pk_u}(\vec{y})$
$c_{\vec{y}}$
compute
$c_{C_w(\vec{y})} := Enc_{pk_u}(C_w(\vec{y}))$
$c_{C_w(\vec{y})}$
World AB
Sim(pk_u)
uniformly randomly generate w'
compute
$c_{C_{w'}(\vec{x})} := Enc_{pk_u}(C_{w'}(\vec{x}))$
User($\vec{x}$, pk_u, sk_u)
$c_{\vec{x}} \leftarrow Enc_{pk_u}(\vec{x})$
$c_{\vec{x}}$
$c_{C_{w'}(\vec{x})}$
$C_{w'}(\vec{x}) \leftarrow Dec_{sk_u}(c_{C_{w'}(\vec{x})})$
World BA
Trusted Third Party
w
$C_w(\vec{x})$
$\vec{x}$
Server
User
Ideal

Fig. C.22. Simulator construction for proof of Theorem 1.

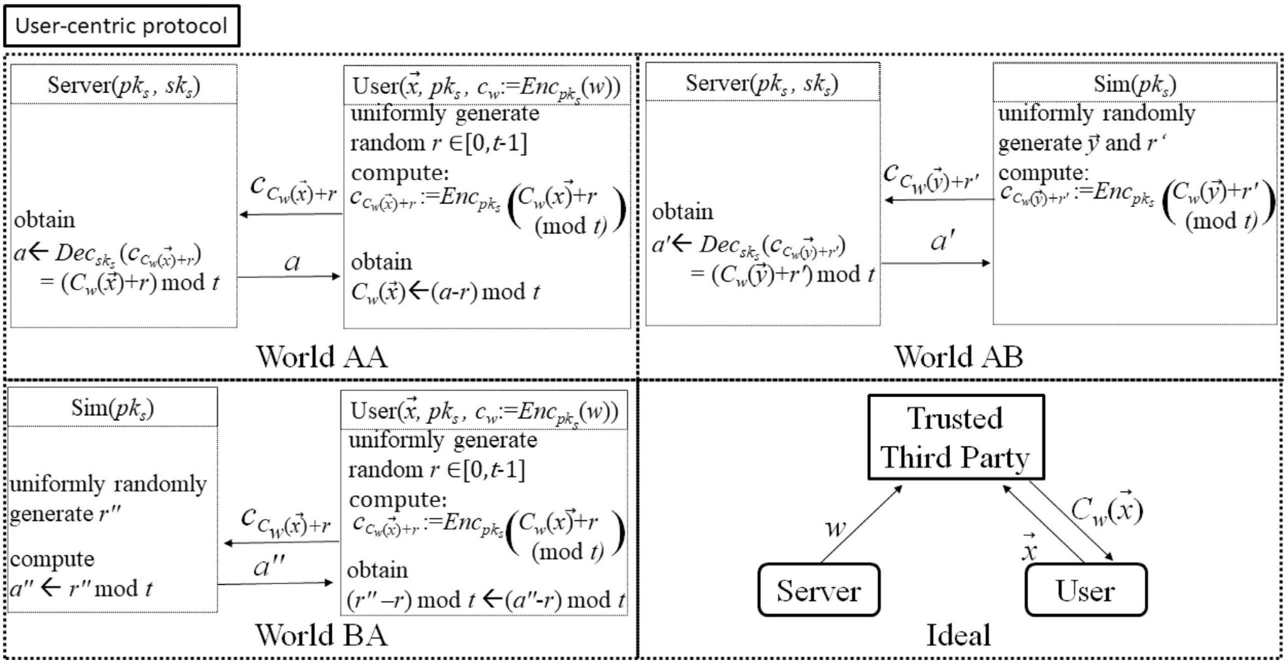

User-centric protocol
Server(pk_s, sk_s)
User($\vec{x}$, pk_s, $c_w := Enc_{pk_s}(w)$)
uniformly generate random $r \in [0, t\text{-}1]$
compute:
$c_{C_w(\vec{x})+r} := Enc_{pk_s}\left(C_w(\vec{x})+r \pmod t\right)$
$c_{C_w(\vec{x})+r}$
obtain
$a \leftarrow Dec_{sk_s}(c_{C_w(\vec{x})+r}) = (C_w(\vec{x})+r) \bmod t$
a
obtain
$C_w(\vec{x}) \leftarrow (a\text{-}r) \bmod t$
World AA
Server(pk_s, sk_s)
Sim(pk_s)
uniformly randomly generate $\vec{y}$ and r'
compute:
$c_{C_w(\vec{y})+r'} := Enc_{pk_s}\left(C_w(\vec{y})+r' \pmod t\right)$
$c_{C_w(\vec{y})+r'}$
obtain
$a' \leftarrow Dec_{sk_s}(c_{C_w(\vec{y})+r'}) = (C_w(\vec{y})+r') \bmod t$
a'
World AB
Sim(pk_s)
uniformly randomly generate r''
compute
$a'' \leftarrow r'' \bmod t$
User($\vec{x}$, pk_s, $c_w := Enc_{pk_s}(w)$)
uniformly generate random $r \in [0, t\text{-}1]$
compute:
$c_{C_w(\vec{x})+r} := Enc_{pk_s}\left(C_w(\vec{x})+r \pmod t\right)$
$c_{C_w(\vec{x})+r}$
a''
obtain
$(r'' - r) \bmod t \leftarrow (a''\text{-}r) \bmod t$
World BA
Trusted Third Party
w
$C_w(\vec{x})$
$\vec{x}$
Server
User
Ideal

Fig. C.23. Simulator construction for proof of Theorem 2.

a randomly generated model. In this case, the user can only determine the same as in the ideal model of the proposed protocol, as shown in the bottom-right of Fig. C.22. Therefore, if the user is able to extract any information about w in the proposed protocol, then the user can also extract that information on w in the ideal model. Therefore, we can conclude that the proposed protocol meets the best level of security that can be achieved in the setting. □

Theorem 2. *If the proposed user-centric protocol uses a CPA-secure FHE scheme and the server and user have computational power that is at most the same as a conventional CPA adversary [62], then the user input privacy and server model privacy requirements defined in Definitions 1 and 2, respectively, are satisfied.*

Proof of Theorem 2. We follow the same approach as for the proof of Theorem 1. To prove the theorem, we build up simulators in Worlds AB and BA of Fig. C.23, respectively.

(Proof of Definition 1) If the server cannot distinguish whether it is working in World AA or World AB in Fig. C.23, the theorem holds. For the server to distinguish between the two, the distribution of $((C_w(\vec{x}) + r) \mod t)$ should be different from that of $((C_w(\vec{y}) + r'') \mod t)$, where $\vec{y}$ is a uniformly randomly generated input vector and r and r'' are uniformly randomly generated numbers in $[0, t-1]$. As the range of the function $C_w()$ is $[0, t-1]$, r and r'' have the same distribution. Therefore, the proposed user-centric protocol satisfies Definition 1.

(Proof of Definition 2) Similar to the proof of Definition 1, if the user cannot distinguish whether he/she is in World AA or World BA in Fig. C.23 with non-negligible probability, the protocol satisfies Definition 2. If we look carefully at Fig. C.23, the difference between the worlds is that the user obtains a in world AA, but a' in world BA. As we can easily derive that both are drawn from $[0, t-1]$ uniformly, if the user obeys the protocol, we can conclude that he/she cannot distinguish between a and a' with non-negligible probability. Therefore, the theorem holds. □

Supplementary material

Supplementary material associated with this article can be found, in the online version, at 10.1016/j.csi.2017.12.004

References

[1] R. Bost, R.A. Popa, S. Tu, S. Goldwasser, Machine learning classification over encrypted data., NDSS, 2015, doi:10.14722/ndss.2015.23241.
[2] B. David, R. Dowsley, R. Katti, A.C. Nascimento, Efficient unconditionally secure comparison and privacy preserving machine learning classification protocols, in: International Conference on Provable Security, Springer, 2015, pp. 354–367, doi:10.1007/978-3-319-26059-4_20.
[3] M. De Cock, R. Dowsley, C. Horst, R. Katti, A. Nascimento, W.-S. Poon, S. Truex, Efficient and private scoring of decision trees, support vector machines and logistic regression models based on pre-computation, IEEE Trans. Dependable Secure Comput. (2017), doi:10.1109/TDSC.2017.2679189.
[4] X. Liu, R. Lu, J. Ma, L. Chen, B. Qin, Privacy-preserving patient-centric clinical decision support system on naive bayesian classification, IEEE J. Biomed. Health Informat. 20 (2) (2016) 655–668, doi:10.1109/JBHI.2015.2407157.
[5] M. Brambilla, G. Guglielmetti, C. Tziviskou, Asynchronous web services communication patterns in business protocols, in: International Conference on Web Information Systems Engineering, Springer, 2005, pp. 435–442, doi:10.1007/11581062_33.
[6] C. Gentry, et al., Fully homomorphic encryption using ideal lattices., in: STOC, 9, 2009, pp. 169–178, doi:10.1145/1536414.1536440.
[7] C. Schurink, P. Lucas, I. Hoepelman, M. Bonten, Computer-assisted decision support for the diagnosis and treatment of infectious diseases in intensive care units, Lancet Infect. Dis. 5 (5) (2005) 305–312, doi:10.1016/S1473-3099(05)70115-8.
[8] N.P. Smart, F. Vercauteren, Fully homomorphic simd operations, Designs, Codes Cryptograp. (2014) 1–25, doi:10.1007/s10623-012-9720-4.
[9] C. Gentry, S. Halevi, N.P. Smart, Homomorphic evaluation of the aes circuit, in: Advances in Cryptology–CRYPTO 2012, Springer, 2012, pp. 850–867, doi:10.1007/978-3-642-32009-5_49.
[10] S. Halevi, V. Shoup, Algorithms in helib, in: International Cryptology Conference, Springer, 2014, pp. 554–571, doi:10.1007/978-3-662-44371-2_31.
[11] S. Halevi, V. Shoup, Bootstrapping for helib, in: Annual International Conference on the Theory and Applications of Cryptographic Techniques, Springer, 2015, pp. 641–670, doi:10.1007/978-3-662-46800-5_25.
[12] I. Kononenko, Machine learning for medical diagnosis: history, state of the art and perspective, Artif. Intell. Med. 23 (1) (2001) 89–109, doi:10.1016/S0933-3657(01)00077-X.
[13] N. Lavrač, I. Kononenko, E. Keravnou, M. Kukar, B. Zupan, Intelligent data analysis for medical diagnosis: using machine learning and temporal abstraction, AI Commun. 11 (3, 4) (1998) 191–218, doi:10.1007/978-1-4615-6059-3.
[14] K. Bache, M. Lichman, Uci machine learning repository(2013).
[15] C.C. Aggarwal, S.Y. Philip, A general survey of privacy-preserving data mining models and algorithms, in: Privacy-preserving data mining, Springer, 2008, pp. 11–52, doi:10.1007/978-0-387-70992-5_2.
[16] R. Agrawal, R. Srikant, Privacy-preserving data mining, in: ACM Sigmod Record, 29, ACM, 2000, pp. 439–450, doi:10.1145/342009.335438.
[17] D. Agrawal, C.C. Aggarwal, On the design and quantification of privacy preserving data mining algorithms, in: Proceedings of the twentieth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems, ACM, 2001, pp. 247–255, doi:10.1145/375551.375602.
[18] A. Machanavajjhala, D. Kifer, J. Gehrke, M. Venkitasubramaniam, l-diversity: privacy beyond k-anonymity, ACM Trans. Knowl. Discovery Data (TKDD) 1 (1) (2007) 3, doi:10.1145/1217299.1217302.
[19] A.C. Yao, Protocols for secure computations, in: Foundations of Computer Science, 1982. SFCS'08. 23rd Annual Symposium on, IEEE, 1982, pp. 160–164, doi:10.1109/SFCS.1982.38.
[20] O. Goldreich, S. Micali, A. Wigderson, How to play any mental game, in: Proceedings of the nineteenth annual ACM symposium on Theory of computing, ACM, 1987, pp. 218–229, doi:10.1145/28395.28420.
[21] W. Henecka, A.-R. Sadeghi, T. Schneider, I. Wehrenberg, et al., Tasty: tool for automating secure two-party computations, in: Proceedings of the 17th ACM conference on Computer and communications security, ACM, 2010, pp. 451–462, doi:10.1145/1866307.1866358.
[22] D. Malkhi, N. Nisan, B. Pinkas, Y. Sella, et al., Fairplay-secure two-party computation system., USENIX Security Symposium, 4, San Diego, CA, USA, 2004 http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.130.1264.
[23] A. Ben-David, N. Nisan, B. Pinkas, Fairplaymp: a system for secure multi-party computation, in: Proceedings of the 15th ACM conference on Computer and communications security, ACM, 2008, pp. 257–266, doi:10.1145/1455770.1455804.
[24] J.H. Ziegeldorf, J. Metzke, M. Henze, K. Wehrle, Choose wisely: a comparison of secure two-party computation frameworks, in: Security and Privacy Workshops (SPW), 2015 IEEE, IEEE, 2015, pp. 198–205, doi:10.1109/SPW.2015.9.
[25] I. Damgard, M. Geisler, M. Kroigard, A correction to'efficient and secure comparison for on-line auctions', Int. J. Appl. Cryptograp. 1 (4) (2009) 323–324, doi:10.1504/IJACT.2009.028031.
[26] D.J. Wu, T. Feng, M. Naehrig, K. Lauter, Privately evaluating decision trees and random forests, Proc. Privacy Enhancing Technol. 2016 (4) (2016) 335–355, doi:10.1515/popets-2016-0043.
[27] T. Graepel, K. Lauter, M. Naehrig, Ml confidential: machine learning on encrypted data, in: International Conference on Information Security and Cryptology, Springer, 2012, pp. 1–21, doi:10.1007/978-3-642-37682-5_1.
[28] J.W. Bos, K. Lauter, M. Naehrig, Private predictive analysis on encrypted medical data, J. Biomed. Informat. 50 (2014) 234–243, doi:10.1016/j.jbi.2014.04.003.
[29] W. Lu, S. Kawasaki, J. Sakuma, Using fully homomorphic encryption for statistical analysis of categorical, ordinal and numerical data., NDSS, 2017, doi:10.14722/ndss.2017.23119.
[30] C. Liu, X.S. Wang, K. Nayak, Y. Huang, E. Shi, Oblivm: a programming framework for secure computation, in: Security and Privacy (SP), 2015 IEEE Symposium on, IEEE, 2015, pp. 359–376, doi:10.1109/SP.2015.29.
[31] A. Khedr, G. Gulak, V. Vaikuntanathan, Shield: scalable homomorphic implementation of encrypted data-classifiers, IEEE Trans. Comput. 65 (9) (2016) 2848–2858, doi:10.1109/TC.2015.2500576.
[32] A. McCallum, K. Nigam, et al., A comparison of event models for naive bayes text classification, in: AAAI-98 workshop on learning for text categorization, 752, Citeseer, 1998, pp. 41–48. http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.65.9324.
[33] J. Chen, H. Huang, S. Tian, Y. Qu, Feature selection for text classification with naïve bayes, Expert Syst. Appl. 36 (3) (2009) 5432–5435, doi:10.1016/j.eswa.2008.06.054.
[34] I. Rish, An empirical study of the naive bayes classifier, in: IJCAI 2001 workshop on empirical methods in artificial intelligence, 3, IBM New York, 2001, pp. 41–46. http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.330.2788.
[35] R. Bellazzi, B. Zupan, Predictive data mining in clinical medicine: current issues and guidelines, International journal of medical informatics 77 (2) (2008) 81–97, doi:10.1016/j.ijmedinf.2006.11.006.
[36] J.L. Hellerstein, T. Jayram, I. Rish, et al., Recognizing end-user transactions in performance management, IBM Thomas J. Watson Research Division, 2000 http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.500.7777.
[37] Z. Brakerski, C. Gentry, V. Vaikuntanathan, (leveled) fully homomorphic encryption without bootstrapping, ACM Trans. Comput. Theory (TOCT) 6 (3) (2014) 13, doi:10.1145/2633600.
[38] E. Crockett, C. Peikert, $\lambda o \lambda$: functional lattice cryptography, in: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, ACM, 2016, pp. 993–1005, doi:10.1145/2976749.2978402.
[39] Y. Doröz, Y. Hu, B. Sunar, Homomorphic aes evaluation using the modified ltv scheme., Des. Codes Cryptography 80 (2) (2016) 333–358, doi:10.1007/s10623-015-0095-1.
[40] I. Chillotti, N. Gama, M. Georgieva, M. Izabachéne, Improving tfhe: faster packed homomorphic operations and efficient circuit bootstrapping (2017). http://www.eprint.iacr.org/2017/430.
[41] M.R. Albrecht, On dual lattice attacks against small-secret lwe and parameter choices in helib and seal, in: Annual International Conference on the Theory and Applications of Cryptographic Techniques, Springer, 2017, pp. 103–129, doi:10.1007/978-3-319-56614-6.
[42] S. Halevi, V. Shoup, Design and implementation of a homomorphic-encryption library, IBM Res. (Manuscript) 6 (2013) 12–15. http://researcher.ibm.com/researcher/files/us-shaih/he-library.pdf.
[43] P. Kim, Y. Lee, H. Yoon, Sorting method for fully homomorphic encrypted data using the cryptographic single-instruction multiple-data operation, IEICE Trans. Commun. 99 (5) (2016) 1070–1086, doi:10.1587/transcom.2015EBP3359.
[44] V. Lyubashevsky, C. Peikert, O. Regev, On ideal lattices and learning with errors over rings, in: Annual International Conference on the Theory and Applications of Cryptographic Techniques, Springer, 2010, pp. 1–23, doi:10.1007/978-3-642-13190-5_1.
[45] O. Goldreich, Foundations of Cryptography: Volume 2, Basic Applications, Cambridge University Press, 2009, doi:10.1017/CBO9780511721656.
[46] M. Fredrikson, S. Jha, T. Ristenpart, Model inversion attacks that exploit confidence information and basic countermeasures, in: Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, ACM, 2015, pp. 1322–1333, doi:10.1145/2810103.2813677.
[47] G. Loukides, J.C. Denny, B. Malin, The disclosure of diagnosis codes can breach research participants' privacy, J. Am. Med. Inform. Assoc. 17 (3) (2010) 322–327, doi:10.1136/jamia.2009.002725.

[48] S. Tschiatschek, F. Pernkopf, On bayesian network classifiers with reduced precision parameters, IEEE Trans. Pattern Anal. Mach. Intell. 37 (4) (2015) 774–785, doi:10.1109/TPAMI.2014.2353620.

[49] S. Tschiatschek, P. Reinprecht, M. Mücke, F. Pernkopf, Bayesian network classifiers with reduced precision parameters, Mach. Learn. Knowl. Discovery Databases (2012) 74–89, doi:10.1007/978-3-642-33460-3_10.

[50] K. Seo, P. Kim, Y. Lee, Implementation and performance enhancement of arithmetic adder for fully homomorphic encrypted data, J. Korea Inst. Inf. Secur. Cryptol. 27 (2) (2017), doi:10.13089/JKIISC.2017.27.3.413.

[51] S. Knowles, A family of adders, in: Computer Arithmetic, 2001. Proceedings. 15th IEEE Symposium on, IEEE, 2001, pp. 277–281, doi:10.1109/ARITH.2001.930129.

[52] G.S. Çetin, Y. Doröz, B. Sunar, E. Savaş, Depth optimized efficient homomorphic sorting(2015) 61–80. doi: 10.1007/978-3-319-22174-8_4.

[53] P. Paillier, Public-key cryptosystems based on composite degree residuosity classes, in: International Conference on the Theory and Applications of Cryptographic Techniques, Springer, 1999, pp. 223–238, doi:10.1007/3-540-48910-X_16.

[54] S. Goldwasser, S. Micali, Probabilistic encryption, J. Comput. Syst. Sci. 28 (2) (1984) 270–299, doi:10.1016/0022-0000(84)90070-9.

[55] D. Boneh, R. Venkatesan, Breaking rsa may not be equivalent to factoring, Adv. Cryptol.–EUROCRYPT'98 (1998) 59–71, doi:10.1007/BFb0054117.

[56] N. Gilboa, Two party rsa key generation, in: Annual International Cryptology Conference, Springer, 1999, pp. 116–129, doi:10.1007/3-540-48405-1_8.

[57] G. Asharov, Y. Lindell, T. Schneider, M. Zohner, More efficient oblivious transfer and extensions for faster secure computation, in: Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security, ACM, 2013, pp. 535–548, doi:10.1145/2508859.2516738.

[58] D. Boneh, The decision diffie-hellman problem, Algo. Num. Theory (1998) 48–63, doi:10.1007/BFb0054851.

[59] V. Lyubashevsky, C. Peikert, O. Regev, On ideal lattices and learning with errors over rings, in: Annual International Conference on the Theory and Applications of Cryptographic Techniques, Springer, 2010, pp. 1–23, doi:10.1007/978-3-642-13190-5_1.

[60] O. Goldreich, Foundations of Cryptography: Volume 1, Basic Tools, Cambridge University Press, 2001, doi:10.1017/CBO9780511546891.

[61] M. Bellare, P. Rogaway, The security of triple encryption and a framework for code-based game-playing proofs, in: Annual International Conference on the Theory and Applications of Cryptographic Techniques, Springer, 2006, pp. 409–426, doi:10.1007/978-3-540-34547-3_36.

[62] O. Goldreich, Foundations of Cryptography: Volume 2, Basic Applications, Cambridge University Press, 2001, doi:10.1017/CBO9780511721656.