Non-interactive privacy-preserving naïve Bayes classifier from leveled fully homomorphic encryption

Jingwei Chen[1,2], Yong Feng[1,2], Yang Liu[3*], Wenyuan Wu[1,2] and Guanci Yang[4]

[1]Chongqing Key Laboratory of Automated Reasoning and Cognition, Chongqing Institute of Green and Intelligent Technology, CAS, Chongqing, 400714, China.
[2]Chongqing College, University of Chinese Academy of Sciences, Chongqing, 400714, China.
[3*]College of Information Science and Engineering, Chongqing Jiaotong University, Chongqing, 400074, China.
[4]Key Laboratory of Advanced Manufacturing Technology of Ministry of Education, Guizhou University, Guiyang, 400074, China.

*Corresponding author(s). E-mail(s): liuyang13@cqjtu.edu.cn;

Abstract

In this paper, we propose a privacy-preserving naïve Bayes classifier based on leveled fully homomorphic encryption schemes. The classifier runs on a server that is also the owner of the model, with input as encrypted data from a client. The classifier produces encrypted classification results, which can only be decrypted by the client, whereas the model is only accessible to the server itself. Therefore, the classifier does not leak private information on either the server's model or the client's data and results. More importantly, the classifier does not require any interactions between the server and the client during the classification phase. The main technical ingredient is an algorithm to compute the maximum index of an encrypted array homomorphically, which does not require any interactions. The proposed classifier is implemented using HElib. Experiments show the accuracy and efficiency of our classifier. For instance, the average cost can achieve about **34**ms per sample for a real data set in UCI Machine Learning Repository with the security parameter about **100** and accuracy about **97%**.

Keywords: privacy-preserving machine learning, naïve Bayes classifier, fully homomorphic encryption, BGV, HElib

1 Introduction

Over the past decade, Machine Learning as a Service has been involved in various fields, from academia to industry. A typical application scenario is that the model vendor uses a large amount of user data to train the model and then uses the trained model to infer/predict some results based on data supplied by clients. However, as security incidents such as data breaches continue to occur, the demand for privacy-preserving MLaaS is rapidly increasing. On the one hand, the model owner is unwilling to leak information about the model. On the other hand, the data owner is reluctant to leak information about the data as well. To resolve this contradiction, privacy-preserving machine learning is proposed.

In this paper, we consider the framework presented in [1] for privacy-preserving classifiers. As shown in Fig. 1, each shaded box indicates private data that should be accessible to only one party: the model to the server and the sample data and classification result to the client. The framework in Fig. 1 happens quite often in practice. For instance, the server might be a big-data service provider with a predictive model for a specific disease, and the client might be a hospital that needs to diagnose many potential cases every day. Following the framework, the hospital first sends the encrypted samples to the service provider for a diagnosis. The service provider runs the classifier with input as its model and the received encrypted samples to obtain an encrypted diagnosis result, and sends it to the hospital. The hospital decrypts it to disclose the diagnosis result.

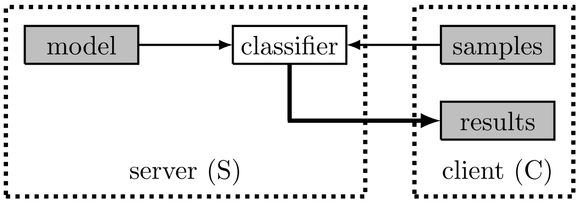

Fig. 1 Framework of privacy-preserving classifiers

In particular, we present a privacy-preserving naïve Bayes classifier (Protocol 1) based on leveled fully homomorphic encryption schemes. The leveled fully homomorphic encryption schemes used in Protocol 1 allow us to evaluate functions with a bounded multiplicative depth on encrypted data, such as BGV [2] and BFV [3, 4]. These schemes are based on the Learning With Errors over Rings (RLWE) assumption [5] and hence thought to be post-quantum safe. Based on the RLWE assumption and the underlying leveled fully homomorphic encryption schemes, we prove that Protocol 1 is correct and secure in the honest-but-curious (semi-honest or passive) model.

Protocol 1 is a minimally interactive protocol. The sample owner (client) encrypts its data x to be predicted as a ciphertext c and sends c to the model owner (server). After receiving the ciphertext c, the server evaluates the model using the client's public key, with input as the encrypted data c. The server sends the resulting ciphertext to the client, and the client decrypts the ciphertext using itself's secret key to specify in which class x lies. Thanks to an algorithm that computes a ciphertext of the maximum index (i.e., the index of the maximum) of an encrypted array (Algorithm 3), no interaction between the client and the server happens during the classification phase (Step 3 in Protocol 1).

We implement Protocol 1 using the C++ homomorphic encryption library HElib [6] and test for the Iris and Wisconsin Breast Cancer (WBC) data set in UCI Machine Learning Repository [7]. Experiments show that Protocol 1 is comparable to existing privacy-preserving naïve Bayes classifiers in literature. For example, with security parameter 100, the average cost of our classifier is 214ms and 34ms per sample for the Iris and WBC data sets, respectively.

Related work

It seems challenging to list all literature on privacy-preserving protocols for classifiers. We refer to [1, 8, 9] for good surveys. Here we only focus on those privacy-preserving naïve Bayes classifiers based on homomorphic encryption.

Naïve Bayes classifiers is a simple but powerful algorithm to predict the category label of unclassified samples; see, e.g., [10]. Bost *et al.* proposed in [1, Sec. VI] the first efficient privacy-preserving protocols for naïve Bayes classifier based on the Quadratic Residuosity (QR) [11] and Paillier [12] cryptosystems, which are known to be broken by quantum computers. Li *et al.* proposed in [13] a secure naïve Bayes classifier for four parties, without experimental results reported. Later on, Kim *et al.* [14] adapted Li *et al.*'s framework using the homomorphic encryption scheme presented by Brakerski, Gentry, and Vaikuntanathan (BGV) [2]. Yasumura *et al.* [15] and Sun *et al.* [16] also gave privacy-preserving protocols for naïve Bayes classification based on BGV. In addition, Wood *et al.* presented in [17] a private naïve Bayes classifier based on a private fully homomorphic encryption scheme proposed by Gribov, Kahrobaei, and Shpilrain [18]. While all of these privacy-preserving naïve Bayes classifiers require interactions among participants during the classification phase (the classifier in Fig. 1), Protocol 1 presented in this paper does not require any interactions at all. For most of them, interactions are needed to compute the maximum index of an encrypted array. Instead, we present a non-interactive algorithm (Algorithm 3), which makes our protocol non-interactive. Furthermore, being different from those protocols based on non-quantum-resistant assumptions, our used leveled fully homomorphic cryptosystem is BGV or BFV, which are based on the RLWE assumption [5] and hence thought to be post-quantum safe.

This paper was presented in part at the 4th EAI International Conference on Security and Privacy in New Computing Environments [19]. In this paper, a rigorous proof for the security of Protocol 1 and an extensive experimental study on the performance are supplemented.

Road-map

In Section 2, we give a brief introduction to the naïve Bayes classifier, homomorphic encryption, and adversarial model. We present several building blocks in Section 3 for our classifier, including the main technical ingredient, Algorithm 3. In Section 4, we propose a privacy-preserving naïve Bayes classifier and prove its correctness and its security in the passive (or honest-but-curious [20]) model. In Section 5, we report extensive experimental results on our implementation of Protocol 1.

2 Preliminaries

In this section, we give some backgrounds useful for the rest of this paper.

2.1 Naïve Bayes classifier

Naïve Bayes classifier is based on the assumption that all features are conditional independent. Consider a data set with s categories $1, \cdots, s$ and n features $X_1, \cdots, X_n$, where each feature X_k has at most t different values $1, 2, \cdots, t$. Under the conditional independence assumption, the classification of a sample $\boldsymbol{x} = (x_1, \cdots, x_n)$ is

$$s^* = \arg \max_{i=1,\ldots,s} \Pr[Y = i] \prod_{k=1}^{n} \Pr[X_k = x_k | Y = i],$$

where $\Pr[Y = i]$ is the probability that each class i occurs, i.e., the *prior probability*, and $\Pr[X_k = x_k | Y = i]$ is the probability of the kth feature X_k to be $x_k \in \{1, 2, \cdots, t\}$ when $\boldsymbol{x}$ belongs to category i, i.e., the *likelihood*. As in [1], we only deal with the case that the domain of the feature values (the x_i's) is discrete and finite, so the $\Pr[X_k = x_k | Y = i]$'s are probability masses.

In addition, it is a common choice to use the Laplacian correction (see, e.g., [33, page 162]) for smoothing the probability estimation.

2.2 The RLWE assumption

The security of many efficient FHE schemes, including BGV [2], BFV [3, 4], CKKS [21], depends on the RLWE assumption or its variants. For more details, we refer the reader to [5].

Definition 1 (RLWE) For security parameter λ, let $\Phi_m(x)$ be the m-th cyclotomic polynomial with degree $n = \varphi(m)$. Let $R = \mathbb{Z}[x]/\langle \Phi_m(x) \rangle$ and let $R_q = R/qR$. Let χ be a distribution over R. The $\mathsf{RLWE}_{m,q,\chi}$ problem is to distinguish between the following two distributions: In the first distribution, one samples (a_i, b_i) uniformly from R_q^2. In the second distribution, one first draw $s \leftarrow R_q$ uniformly and then sample $(a_i, b_i) \in R_q^2$ by sampling $a_i \leftarrow R_q$ uniformly, $e_i \leftarrow \chi$, and setting $b_i = a_i \cdot s + e_i$. The $\mathsf{RLWE}_{m,q,\chi}$ assumption is that the $\mathsf{RLWE}_{m,q,\chi}$ problem is infeasible.

2.3 Leveled fully homomorphic encryption

Fully homomorphic encryption schemes allow arithmetic circuits to be evaluated directly on ciphertexts [22, 23]. Since Gentry's seminal work [23], multiple HE schemes have been designed, such as BGV [2], BFV [3, 4], CKKS [21], FHEW [24], TFHE [25]. Each of them has its features. For instance, BGV and BFV are good at performing large vectorial arithmetic operations, CKKS supports floating-point computations, and FHEW and TFHE run bootstrapping for one bit extremely fast but slow for arithmetic operations. As naïve Bayes classifiers require many integer arithmetic operations, we choose BGV or BFV as the leveled fully homomorphic encryption schemes, with parameters supporting integer arithmetic circuits of a certain bounded depth.

For completeness, we briefly describe the leveled fully homomorphic encryption schemes used in this paper. We restrict to those RLWE-based schemes. In such schemes, the plaintext space is $R_p = \mathbb{Z}_p[x]/\langle\Phi_m(x)\rangle$ and the ciphertext space is $R_q = \mathbb{Z}_q[x]/\langle\Phi_m(x)\rangle$, where $\Phi_m(x)$ is the m-th cyclotomic polynomial, p is a prime number, and $q \gg p$ is an integer. Both of the BGV [2] and the BFV [3, 4] schemes have this structure.

Typically, a leveled fully homomorphic encryption scheme FHE can be described by the following randomized algorithms:

- $\mathsf{FHE.Setup}(1^\lambda)$. Given a security parameter λ as input, outputs parms.
- $\mathsf{FHE.KeyGen}(\mathsf{parms})$. Output a secret key $\mathsf{sk} = \boldsymbol{s}$ and the corresponding public key pk. (For convenience, we let pk also include one or more evaluation keys ek.)
- $\mathsf{FHE.Enc}_{\mathsf{pk}}(b)$. Given a message $b \in R_p$, outputs a ciphertext $c \in R_q$.
- $\mathsf{FHE.Dec}_{\mathsf{sk}}(\boldsymbol{c})$. Given a ciphertext $c \in R_q$, outputs a message $b \in R_p$.
- $\mathsf{FHE.Eval}_{\mathsf{pk}}(C, (c_1, \cdots, c_k))$. Given an arithmetic circuit C of a function f with k input wires, and input $c_1, \cdots, c_k$ with $c_i \leftarrow \mathsf{FHE.Enc}_{\mathsf{pk}}(b_i)$, outputs a ciphertext c such that $\Pr[\mathsf{FHE.Dec}_{\mathsf{sk}}(c) \neq f(b_1, \cdots, b_k)] = \mathrm{negl}(\lambda)$.

FHE is said to be *compact* if the size of the output of $\mathsf{FHE.Eval}$ is not more than polynomial in λ and is independent of f. FHE is said to be *secure* if it is IND-CPA secure and *weakly circular* secure, which means that the scheme remains secure even if the adversary is given encryptions of the bits of the secret key. We say that FHE achieves *circuit privacy* if the distribution of the outputs of any fixed homomorphic evaluation is indistinguishable from the distribution of fresh encryptions of the plaintext outputs.

2.3.1 Homomorphic evaluation

Let c_1 and c_2 be two ciphertexts of two plaintexts b_1 and b_2 under the same secret key sk. Suppose that the noise of c_1 and c_2 is bounded from above by B. The addition ($\mathsf{FHE.Add}$) of the two ciphertexts is typically $c_+ = c_1 + c_2$, which is a ciphertext of $b_1 + b_2$ under the secret key sk. The noise of c_+ is at most $2B$. For multiplication ($\mathsf{FHE.Mul}$), $c_\times = c_1 \otimes c_2$ is typically a ciphertext of $b_1 \cdot b_2$ under a new secret key $\mathsf{sk} \otimes \mathsf{sk}$ with larger dimension, where $\otimes$ is the usual tensor product. The noise of $c_\times$ can only be bounded from above by B^2. To keep the dimension of the secret key and to decrease the noise of evaluated ciphertext, a refresh procedure $\mathsf{FHE.Refresh}$ (consisting of key switching and modulus switching) follows every homomorphic addition and multiplication. Of course, one can call $\mathsf{FHE.Refresh}$ only if necessary for efficiency. Note that the public key pk of FHE also includes all keys for $\mathsf{FHE.Refresh}$. Theoretically, the cost of each homomorphic addition or multiplication increases fast as L grows, where L is the circuit depth of the function f to be evaluated. Besides, FHE also supports plaintext-ciphertext addition ($\mathsf{FHE.AddConst}$) and plaintext-ciphertext multiplication ($\mathsf{FHE.MulConst}$).

2.3.2 Batching

Recall the plaintext space $R_p = \mathbb{Z}_p[x]/\langle\Phi_m(X)\rangle$. Let d be the multiplicative order of p modulo m, and $\phi(m)$ be the Euler's totient function. Then d divides $\phi(m)$ and $R_p \cong \mathbb{F}_{p^d}^\ell$ with $\ell = \phi(m)/d$. Therefore each plaintext can be seen as a packed message with ℓ slots. From this view, each homomorphic operation on a ciphertext is equivalent to the same operation on all slots independently and simultaneously. This batching technique [26, 27] significantly decreases the *amortized* cost (i.e., the total cost divided by ℓ) of homomorphic encryption schemes based on RLWE. For batching, FHE usually supports data packing ($\mathsf{FHE.Encode}$), data rotating ($\mathsf{FHE.Rotate}$), and data shifting ($\mathsf{FHE.Shift}$). Based on these operations, one can build some advanced functions. For instance, $\mathsf{FHE.TotalSum}$ converts a ciphertext that encrypts $(z_1, \cdots, z_t)$ into a ciphertext that encrypts $(y, \cdots, y)$ with $y = \sum_{i=1}^t z_i$.

2.4 Adversarial model

Our protocol only involves the client and the server, labeled as parties C and S, respectively. To show Protocol 1 preserves the privacy of both parties, we work in the honest-but-curious (semi-honest or passive) model as described in [20, Sec. 7.2]. The materials presented here are mainly taken from the full version of [1].

Let $f = (f_C, f_S)$ be a (probabilistic) polynomial function and Π a protocol computing f. C and S want to compute $f(a, b)$ where a is C's input and b is S's input, using Π and with the security parameter λ. The *view* of party C during the execution of Π is the tuple $V_C(\lambda, a, b) = (1^\lambda;\ a;\ r^C;\ m_1^C, \cdots, m_t^C)$ where r is C's random tape and $m_1^C, \cdots, m_t^C$ are the messages received by C. We define the view of S similarly. The outputs of parties C and S for the execution of Π on input (a, b) as $\mathsf{Output}_C^\Pi(\lambda, a, b)$ and $\mathsf{Output}_S^\Pi(\lambda, a, b)$, which is implicit in the party's own view of the execution, and the global output as $\mathsf{Output}^\Pi(\lambda, a, b) = (\mathsf{Output}_C^\Pi(\lambda, a, b), \mathsf{Output}_S^\Pi(\lambda, a, b))$.

To ensure security, we have to show that whatever C can compute from its interactions with S can be computed from its own input and output, which leads us to the following security definition.

Definition 2 ([20, Def. 7.2.1]) The two-party protocol Π securely computes the function f if there exist two probabilistic polynomial-time algorithms S_C and S_S such that for every possible input a, b of f,

$$\begin{aligned}&\{S_C(1^\lambda, a, f_A(a, b)), f(a, b)\}\\ \equiv_c&\{V_C(\lambda, a, b), \mathsf{Output}^\Pi(\lambda, a, b)\}\end{aligned}$$

and

$$\begin{aligned}&\{S_S(1^\lambda, a, f_B(a, b)), f(a, b)\}\\ \equiv_c&\{V_S(\lambda, a, b), \mathsf{Output}^\Pi(\lambda, a, b)\}\end{aligned}$$

where $\equiv_c$ means computational indistinguishability against probabilistic polynomial time adversaries with negligible advantage in the security parameter λ.

To simplify the notation and proof, we omit the security parameter. As we only consider deterministic functions f, we can simplify the distributions we want to show being indistinguishable: when f is deterministic, to prove the security of Π that computes f, we only have to show that

$$\begin{aligned}S_C(a, f_A(a, b)) &\equiv_c V_C(a, b),\\ S_S(b, f_B(a, b)) &\equiv_c V_S(a, b).\end{aligned} \tag{1}$$

3 Building blocks

We now describe a few necessary building blocks that will be used to build our classifier. Note that all the following algorithms will be executed on the server and that the owner of pk (the public key) in these algorithms is not the server but the client since we follow the framework given in Fig. 1. For simplicity, we use the functions without explicitly showing the name of the scheme FHE in the rest of this paper. For instance, we use Add to replace FHE.Add.

3.1 Plaintext matrix-encrypted vector multiplication

Matrix-vector multiplication is reasonably common in practice. Here we focus on plaintext matrix-encrypted vector multiplication. Given a plaintext matrix $\boldsymbol{A} \in \mathbb{Z}^{s\times t}$ and ciphertexts of a vector $\boldsymbol{z} \in \mathbb{Z}^t$, our goal is to obtain ciphertexts of $\boldsymbol{Az}$. We present two methods based on different ways to encode a vector.

3.1.1 Naïve encoding

To encrypt a vector $\boldsymbol{z} = (z_i)_{i\le t} \in \mathbb{Z}^t$, one can encrypt each entry z_i of $\boldsymbol{z}$ to a ciphertext. The encryption of $\boldsymbol{z}$ is a vector $\boldsymbol{c}' = (c_i')_{i\le t} \in R_q^t$ of ciphertexts, whose i-th entry c_i' is a ciphertext of z_i. Algorithm 1 computes a vector in R_q^s as the encryption of $\boldsymbol{Az}$. Obviously, Algorithm 1 costs no multiplicative depth.

Algorithm 1 Naïve plaintext matrix-encrypted vector multiplication

```
Input: $\boldsymbol{c}' = (c_i')_{i\le t} \in R_q^t$ ($c_i'$ encrypts the $i$th
    entry of $\boldsymbol{z} = (z_i)_{i\le t}$), and public key $\mathsf{pk}$;
    $\boldsymbol{A} = (a_{i,j}) \in \mathbb{Z}^{s\times t}$.
Output: $(c_i)_{i\le s}$ with $c_i = \mathsf{Enc}_{\mathsf{pk}}(\sum_{j=1}^t a_{i,j} z_j)$.
1. For $i = 1, \cdots, s$ do the following:
  (a) $c_i \leftarrow \mathsf{Enc}_{\mathsf{pk}}(0)$;
  (b) For $j = 1, \cdots, t$
     (i) Update
           $c_i := \mathsf{Add}_{\mathsf{pk}}(c_i, \mathsf{MulConst}_{\mathsf{pk}}(a_{i,j}, c_j'))$.
2. Return $(c_i)_{i\le s}$.
```

3.1.2 Packed encoding

Instead of the above element-wise method, we can pack the vector $\boldsymbol{z} \in \mathbb{Z}^t$ into t slots of one plaintext for batching in Sec. 2.3.2, and encrypt it to only one ciphertext $c \in R_q$, which leads to Algorithm 2. Note that FHE.TotalSum costs no multiplicative depth since it uses only FHE.Rotate and FHE.Add, and FHE.Rotate costs no multiplicative depth (see, e.g., [28]). Thus, Algorithm 2 costs no multiplicative depth as well.

Algorithm 2 Pakced plaintext matrix-encrypted vector multiplication

```
Input: $c \in R_q$ that encrypts $u \in R_p$ with $u = \mathsf{Encode}(\boldsymbol{z})$, and public key $\mathsf{pk}$; $\boldsymbol{A} = (a_{i,j}) \in \mathbb{Z}^{s\times t}$.
Output: $(c_i)_{i\le s}$ with $c_i = \mathsf{Enc}_{\mathsf{pk}}(\sum_{j=1}^{t} a_{i,j} z_j)$.
1. For $i = 1, \cdots, s$ do the following:
  (a) Encode the $i$-th row $\boldsymbol{a}_i$ of $\boldsymbol{A}$ as $v_i = \mathsf{Encode}(\boldsymbol{a}_i)$;
  (b) Compute
        $c_i = \mathsf{TotalSum}_{\mathsf{pk}}(\mathsf{MulConst}_{\mathsf{pk}}(v_i, c)).$
2. Return $(c_i)_{i\le s}$.
```

3.2 Argmax of an encrypted array

We first recall a recent comparator presented by Iliashenko and Zucca in [29], which supports comparison operations for BGV and BFV, and then present our method to compute the index of the maximum of an encrypted array.

3.2.1 Comparison

Essentially, the comparison method for encrypted arrays presented in [29] homomorphically evaluates the Lagrange interpolated polynomial of the *less-than* function over $S = [0, (p-1)/2]$ defined as follows:

$$\mathsf{LT}_S(x, y) = \begin{cases} 1, & \text{if } 0 \le x < y \le (p-1)/2, \\ 0, & \text{if } 0 \le y \le x \le (p-1)/2. \end{cases}$$

It can be interpolated by the following polynomial over $\mathbb{F}_p$ of degree $p-1$ by [29, Thm. 3]: $\frac{p+1}{2}(x-y)^{p-1} + \sum_{i=1,odd}^{p-2}\left(\sum_{a=1}^{\frac{p-1}{2}} a^{p-1-i}\right)\cdot(x-y)^i$. This polynomial can be evaluated within

$$\sqrt{p-3} + \frac{3}{2}\log_2(p-3) + O(1) \qquad (2)$$

multiplicative depth.

3.2.2 Argmax

Based on the less-than function LT, we now present an algorithm (Algorithm 3) to compute the maximum index of an encrypted array, denoted $\arg\max$.

For a given array $\boldsymbol{z} = (z_1, \cdots, z_s)$, Algorithm 3 firstly computes a *comparison matrix* $\boldsymbol{L} = (\ell_{i,j})$ with

$$\ell_{i,j} = \begin{cases} 1 - \mathsf{LT}(z_i, z_j) & \text{if } i < j, \\ 1 & \text{if } i = j, \\ \mathsf{LT}(z_j, z_i) & \text{if } i > j. \end{cases}$$

For instance, if $\boldsymbol{z} = (7, 6, 2, 4, 5)$ the matrix $\boldsymbol{L} \in \{0,1\}^{5\times 5}$ is given by

$$\begin{pmatrix} 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 \end{pmatrix}.$$

It is easy to see that there exists only one row with all entries one, and the index of that row is $\arg\max_i(\boldsymbol{z})$. Equivalently, we have

$$\arg\max_i (z_i)_{1\le i\le s} = \sum_{j=1}^{s} j \cdot \prod_{k=1}^{s} \ell_{j,k},$$

which results in Algorithm 3.

Algorithm 3 Encrypted maximum index of an encrypted array

```
Input: $\boldsymbol{c} = (c_1, \cdots, c_s) \in R_q^t$ ($c_i$ encrypts the $i$-th entry of $\boldsymbol{z} = (z_1, \cdots, z_s)$) and public key $\mathsf{pk}$.
Output: A ciphertext $c \in R_q$ that encrypts $\arg\max(\boldsymbol{z})$.
1. Set $c \leftarrow \mathsf{Enc}_{\mathsf{pk}}(0)$.
2. For $i = 1, \cdots, s$ do the following:
  (a) Set $c'$ to be
        $\prod_{k=1,k\ne j}^{s} \mathsf{AddConst}_{\mathsf{pk}}(1, \mathsf{MulConst}_{\mathsf{pk}}(-1, \mathsf{LT}_{\mathsf{pk}}(c_j, c_k))).$
  (b) Update $c := \mathsf{Add}_{\mathsf{pk}}(c, \mathsf{MulConst}_{\mathsf{pk}}(j, c'))$.
3. Return $c$.
```

Note that it requires at most $s(s-1)/2$ comparisons to construct the comparison matrix $\boldsymbol{L}$. Furthermore, one would better use some recursive methods in practice to compute the encrypted product in Step 22a of Algorithm 3 for saving multiplicative depth. From Eq. (2), Algorithm 3 costs at most

$$\lceil \log_2 s \rceil + \sqrt{p-3} + \frac{3}{2}\log_2(p-3) + O(1)$$

multiplicative depth to compute the arg max of an encrypted array.

4 Privacy-preserving naïve Bayes classification

In this section, we present our privacy-preserving naïve Bayes classifier and prove its correctness and security.

4.1 Preparing the model

If the domain of the feature values is continuous, we first find a bound B on the values and then discretize them by splitting $[-B, B]$ into several equal intervals. For example, if the domain of the kth feature X_k is continuous on $[-1, 1]$, then one can discretize X_k as $X_k = 0$ if $X_k \in [-1, 0)$ and $X_k = 1$ if $X_k \in [0, 1]$. This discretization technique enables our classifier to deal with continuous features as well, possibly at the cost of decreasing the prediction accuracy.

For convenience, we limit the values of features $x_1, \cdots, x_n$ in $\{1, 2, \cdots, t\}$. Furthermore, for numerical stability, we work with the logarithm of the probability:

$$s^* = \arg\max_{i=1,\dots,s} \left\{ \log \Pr[Y = i] \right. \\ \left. + \sum_{k=1}^{n} \log \Pr[X_k = x_k | Y = i] \right\}, \tag{3}$$

where $x_k \in \{1, \cdots, t\}$. Another convenient simplification is to take the numbering of the s classes as contiguous integers from 1 to s. Then s^* is precisely the index of the maximum over the s values in (3).

Additionally, since the BGV encryption scheme works with integers, one needs to convert each logarithm of probability in (3) to an integer by multiplying it with a certain number $K > 0$ and rounding it to the closest integer. A similar *shifting* technique is already used and analyzed in, e.g., [1, 30].

In summary, for a data set with s categories and n features (each feature has at most t different values), the prior probability in the model will be converted into a vector $\boldsymbol{b} = (b_1, \cdots, b_s) \in \mathbb{Z}^s$, where b_i is obtained by rounding $K \cdot \log(\Pr[Y = i])$ for an appropriate scaling integer K. The likelihoods will be converted into n matrices $\boldsymbol{A}_k \in \mathbb{Z}^{s \times t}$ for $k = 1, \cdots, n$, where the (i, j)-entry of $\boldsymbol{A}_k$ is derived by rounding $K \cdot \log \Pr[X_k = j | Y = i]$ with the same integer K.

4.2 Privacy-preserving Naïve bayes Classifier

To resolve the privacy concerns, the client should only obtain the classification result s^* without learning any information about the prior probability and likelihood, and the server should learn nothing about the client's data $\boldsymbol{x}$.

The client has data $\boldsymbol{x} = (x_1, \cdots, x_n)$ with $x_k \in \{1, \cdots, t\}$ and wants the server to predict which class $\boldsymbol{x}$ is in using a naïve Bayes classifier without leaking any information about $\boldsymbol{x}$. One choice of the client is to encrypt $\boldsymbol{x}$ using himself's public key. However, since $\boldsymbol{x}$ is encrypted, the server cannot decide which entry of $\boldsymbol{A}_k$ should be chosen. For instance, the first feature of $\boldsymbol{x}$ is x_1, i.e., $X_1 = x_1$. To access the information about $\Pr[X_1 = x_1 | Y = i]$ in $\boldsymbol{A}_1$, we need to select the (i, x_1) entry of $\boldsymbol{A}_1$. However, as the first entry of $\boldsymbol{x}$, x_1 is only available in encrypted form on the server-side. To get around this obstacle, one can encode the sample $\boldsymbol{x}$ as a 0-1 matrix

$$\boldsymbol{X} = (\boldsymbol{e}_{x_1}, \cdots, \boldsymbol{e}_{x_n}) \in \{0, 1\}^{t \times n}, \tag{4}$$

where $\boldsymbol{e}_j$ is the t-dimensional vector whose jth entry is one and all others are zero. Now, to select the x_k-th row of a matrix $\boldsymbol{A}_k \in \mathbb{Z}^{s \times t}$ is just to compute $\boldsymbol{A}_k \boldsymbol{e}_{x_k}$. If $\boldsymbol{e}_{x_k}$ is in encrypted form, this is a plaintext matrix-encrypted vector multiplication discussed in Section 3.

Now we are ready to present our privacy-preserving naïve Bayes classifier as Protocol 1, assuming that FHE achieves circuit privacy. Clearly, the classification phase (Step 3) of Protocol 1 does not require any interactions between the server and the client.

We prove the security of our protocol using the secure two-party computation framework for passive adversaries. Roughly speaking, a passive adversary tries to learn as much private information as possible from the other party; however, this adversary faithfully follows the prescribed protocol.

Proposition 1 *Protocol 1 is correct and secure in the honest-but-curious model.*

Protocol 1 Privacy-preserving naïve Bayes classifier

Input of the client: A sample $\boldsymbol{x} = (x_1, \cdots, x_n)$ to be classified, the secret and public key sk and pk.

Input of the server: The model consisting of the likelihood information $(\boldsymbol{A}_k)_{k\le n}$ and the prior information $\boldsymbol{b} = (b_i)_{i\le s}$, and the client's pk.

1. The client encodes $\boldsymbol{x}$ to a matrix $\boldsymbol{X}$ as in (4).
2. The client encrypts the column vectors $\boldsymbol{e}_{x_k}$ of $\boldsymbol{X}$ for $k = 1, \cdots, n$ and sends the ciphertexts to the server.
3. The server does the following:
   (a) For $i = 1, \cdots, s$, set $c_i \leftarrow \mathsf{Enc}_{\mathsf{pk}}(0)$ and update $c_i := \mathsf{AddConst}_{\mathsf{pk}}(c_i, b_i)$.
   (b) For $k = 1, \cdots, n$, calling Algorithm 1 or 2 with input as $\boldsymbol{A}_k$, pk, and the ciphertexts of $\boldsymbol{e}_{x_k}$ received from the client, outputs $(c'_i)_{i\le s}$.
   (c) Update $c_i := \mathsf{Add}_{\mathsf{pk}}(c_i, c'_i)$ for $i = 1, \cdots, s$.
   (d) Calling Algorithm 3 with input as $\boldsymbol{c} = (c_i)_{i\le s}$ and pk returns c.
4. The server sends c to the client.
5. The client decrypts c to $y = \mathsf{Dec}_{\mathsf{sk}}(c)$ and outputs y.

Proof The correctness follows directly from that what the server does is to evaluate the following procedure homomorphically:

1. Set $\boldsymbol{y} := \boldsymbol{b}$, the information of the prior probability.
2. For $k = 1, \cdots, n$, set $\boldsymbol{y} := \boldsymbol{y} + \boldsymbol{A}_k \cdot \boldsymbol{e}_{x_k}$.
3. Return y as the index of the maximum entry of $\boldsymbol{y} = (y_i)_{0\le i\le s-1}$.

We prove the security by Eq. (1). The client's view is

$$V_C = (\mathsf{pk}, \mathsf{sk}, \boldsymbol{x};\ c, y).$$

The simulator S_C, on input $(\mathsf{pk}, \mathsf{sk}, \boldsymbol{x}, y')$ with

$$y' = \arg\max_{i=1,\ldots,s} \left(\sum_{k=1}^{n} \boldsymbol{A}_k \boldsymbol{e}_{x_k} + \boldsymbol{b} \right),$$

generates a ciphertext $c' = \mathsf{Enc}_{\mathsf{pk}}(z)$, where z is a random integer and outputs $(\mathsf{pk}, \mathsf{sk}, \boldsymbol{x};\ c', y')$. As the integer y that the client receives is its output, and as the given FHE scheme is semantically secure and achieves circuit privacy, the distributions $S_C = (\mathsf{pk}, \mathsf{sk}, \boldsymbol{x};\ c', y')$ and $V_C = (\mathsf{pk}, \mathsf{sk}, \boldsymbol{x};\ c, y)$ are computationally indistinguishable.

The view of the server is

$$V_S = ((\boldsymbol{A}_k)_k, \boldsymbol{b}, \mathsf{pk};\ \boldsymbol{X}', c),$$

where $\boldsymbol{X}'$ is ciphertexts that encrypt $\boldsymbol{X}$. The simulator S_S, on input $((\boldsymbol{A}_k)_k, \boldsymbol{b}, \mathsf{pk})$,

- generates a random 0-1 matrix $\boldsymbol{Y}$ of size $t \times n$ and computes the ciphertexts $\boldsymbol{Y}'$ that encrypt $\boldsymbol{Y}$,
- generates a ciphertext $c' = \mathsf{Enc}_{\mathsf{pk}}(z)$, where z is a random integer,
- outputs $((\boldsymbol{A}_k)_k, \boldsymbol{b}, \mathsf{pk};\ \boldsymbol{Y}', c')$.

The distributions V_S and S_S are computationally indistinguishable, because of the same reason for V_C and S_C. This completes the proof. □

Protocol 1 assumes that FHE is with circuit privacy. For fully homomorphic encryption schemes, circuit privacy can be achieved using, e.g., the techniques of [31]. In practice, the slightly weaker notion of (statistical) *function privacy* [32] suffices, and is easier to achieve in the leveled fully homomorphic encryption setting using *re-randomization* and *noise flooding*, where the server re-randomizes the output ciphertexts by homomorphically adding a ciphertext of zero with a large noise [23, 31].

5 Implementation and experiments

We have implemented three variants of Protocol 1 in C++ using HElib (v2.1.0) [6]. The first two variants come from the different choices in Step 33b of Protocol 1. These two variants deal with only one sample at a time. The third variant comes from the batching technique (see Section 2.3.2), which we call the *batching variant*. It is based on the same encoding scheme as Algorithm 1. Assume that the selected parameters support ℓ plaintext slots. In the batching variant, we pack the information of a sample into one slot to deal with at most ℓ samples at a time.

In this section, we will report the prediction accuracy, communication cost and calculation time of our implementations of Protocol 1. All experiments run serially (using only one thread) on a laptop with a Ubuntu 20.04 OS as Windows Subsystem for Linux, 2.60 GHz Intel Core i7-10750H CPU (64 bit) with 16 GB RAM.

5.1 Data set

The Iris and Wisconsin Breast Cancer (WBC) data sets in UCI Machine Learning Repository [7] were used in this experiment.

The Iris data set has 150 samples classified into three categories (i.e., $s = 3$). Each sample has four features (i.e., $n = 4$), and each feature takes at most five different values (i.e., $t = 5$). Our experiment used 120 samples (80%) for training the model and the remaining 30 samples for prediction.

For WBC, the dataset has 683 effective samples, classified into two categories, i.e., $s = 2$. There are nine features for each sample, and each feature may take at most ten different values, i.e., $n = 9$ and $t = 10$. Among these 683 samples, 478 samples are used for training (70%), and the remaining 205 samples are used to test.

5.2 Parameter setting

For Iris, the scaling factor in Section 4.1 is set to be one, i.e., $K = 1$, which leads that the entries of the rounded logarithm of likelihood $\boldsymbol{A}_k$ for $k = 1, \cdots, 4$ are integers between -4 and 0, and the entries of the rounded logarithm of the prior probability are bounded by 1. Hence the resulting integers to be compared must be at most $4 \cdot 4 + 1 = 17$, which implies that $p = 37$ is enough for our purpose. In addition, m is fixed to 14539. In this setting, each plaintext in R_p has $\ell = 1980$ slots.

For WBC, K is also set to be 1. Hence the entries of the rounded logarithm of likelihood $\boldsymbol{A}_k$ for $k = 1, \cdots, 9$ are integers between -6 and 0, and the entries of the rounded logarithm of the prior probability are bounded by 2. Hence the resulting integers to be compared must be at most $6 \cdot 9 + 2 = 56$, which implies that $p = 113$ is enough for our purpose. In addition, m is fixed to 12883. In this setting, each plaintext in R_p has $\ell = 3960$ slots.

For both data sets, the standard deviation σ of the error distribution in the encryption scheme is fixed to the default value in HElib, i.e., $\sigma = 3.2$, which is an approximation of $8/\sqrt{2\pi}$.

5.3 Accuracy

Our experiment shows that the classification accuracy of our implementation of Protocol 1 based on HElib is about 97% for both Iris and WBC and for all three variants. Note that this accuracy is almost the same as the plaintext (unencrypted) naïve Bayes classifier.

5.4 Communication

Although Protocol 1 does not require interactions between the server and client during the classification phase, it does require one interaction, which consists of that the client sends the encrypted data to the server and that the server sends the encrypted result to the client.

Table 1 counts the communication cost of the batching variant. The column labeled "Data" gives the size of the ciphertext file for all samples to be classified (i.e., 30 samples for Iris and 205 samples for WBC), and the "Result" column gives the size of the resulting ciphertext for all samples. For Iris and WBC, the amortized communication costs of the batching variant are 22.15K and 46.09KB, respectively. Table 2 compares the transferred data sizes among the batching variant of Protocol 1 and some other existing privacy-preserving naïve Bayes classifiers. The star mark $(*)$ means that the data is taken directly from the corresponding reference. For communication cost, Protocol 1 is not as good as the privacy-preserving naïve Bayes classifier presented in [1] (based on the QR and Paillier cryptosystems) but is better than that in [15] (based on BGV).

5.5 Timing

In Step 33b of Protocol 1, there are two choices (Algorithm 1 and Algorithm 2) for plaintext matrix-encrypted vector multiplication. We test them all, together with the batching variant, and record their performance. The row named "naïve" ("packed" resp.) is the performance of Protocol 1 based on Algorithm 1 (Algorithm 2 resp.), and the row named "batching" is the performance of the batching variant of Protocol 1 using SIMD. The columns with "Each sample" give the average execution time for each sample. All timings with the

Table 1 Communication cost (KB) of Protocol 1

	Data	Result	Data per sample	Result per sample
Iris	40,980	2877	1366	96
WBC	180,875	1633	882	8

Table 2 Total communication cost for each sample of WBC

	[1]	[15]	Protocol 1
Transferred data size (KB)	**74***	4096*	890

star mark (∗) are directly taken from the the literature, in which the taken timings are the best ones reported. Note that the framework used in the literature may not be the same as in Fig. 1.

It can be observed from Tables 3 and 4 that the naïve variant is more efficient than the packed one, although the timing for encryption is worse. Overall, the batching variant is the most efficient one among the three variants of Protocol 1. The average cost of the batching variant for each sample of Iris and WBC is about 214ms and 34ms, respectively. Since the "batching" variant can classify $\ell = 1980$ (3960 resp.) samples simultaneously, the amortized cost in our setting can be less than 4ms (2ms resp.) per sample for Iris (WBC resp.).

We also compare our implementation with several existing privacy-preserving naïve Bayes classifiers in Tables 3 and 4. For the Iris data set, Kim *et al.* [14] reported that their proposed privacy-preserving naïve Bayes classifier takes 17h40m on a single CPU core and 5h03m on four CPU cores. For WBC, Bost *et al.* [1] reported that their classifier takes about 0.479s and 14 interactions per sample. The computation time of the modified protocol presented in [15] is about 0.732s per sample. Wood *et al.* reported that their classifier takes about 0.402s for each sample. The classifier presented by Sun *et al.* [16] takes about 0.141s per sample with SIMD for WBC. Note that the classifiers presented in [15] and [16] are also implemented using HElib. Overall, although with a not-so-good total cost, Protocol 1 is comparable to existing protocols, especially when the client has a lot of samples to be classified at a time.

6 Conclusion

In this paper, we attempt to design privacy-preserving classifier protocols in the client-server setting. The server owns the model, which should not be accessible to any other, and the client also needs to preserve the privacy of the data to be predicted. As a result, we propose a privacy-preserving naïve Bayes classifier (Protocol 1) based on leveled fully homomorphic encryption schemes, such as BGV and BFV. We show that the classifier is correct and secure in the honest-but-curious model. The main feature of our classifier is that it does not require any interaction between the client and the server during the classification phase. According to experiments with our implementation based on HElib, the efficiency of Protocol 1 is comparable to existing ones. An intriguing direction is to extend the framework in Fig. 1 to more classifiers such as decision trees, nearest neighbor classifier, etc.

Acknowledgment

This research was supported partly by National Key Research and Development Project of China (2020YFA0712303), Natural Science Foundation of China (61903053), Guizhou Science and Technology Program [2020]4Y056, Chongqing Science and Technology Program (cstc2021-jcyjmsxmX0821, cstc2019yszx-jcyjX0003, cstc2020yszx-jcyjX0005, cstc2021yszx-jcyjX0004, KJQN201900702), and Youth Innovation Promotion Association of CAS (2018419).

Data availability

The datasets generated during and/or analysed during the current study are available in the UCI Machine Learning Repository, http://archive.ics.uci.edu/ml.

Conflict of Interests

The authors declare that they have no conflict of interest.

Table 3 Timing for Iris (s)

	$\log q$	λ	Enc	Argmax	Dec	Total	Each sample
naïve	387	100	16.641	36.457	3.147	93.778	3.126
packed	488	74	4.191	32.299	6.09	405.181	13.506
batching	387	100	0.556	4.571	0.024	6.423	**0.214**
[14]	–	–	–	–	–	–	18,180*

Table 4 Timing for WBC (s)

	$\log q$	λ	Enc	Argmax	Dec	Total	Each sample
naïve	382	100	464.654	338.236	12.555	1332.812	6.502
packed	476	76	54.299	321.292	23.046	1814.817	8.853
batching	382	100	2.376	1.58	0.397	7.033	**0.034**
[1]	–	100	–	–	–	–	0.479*
[15]	–	119	–	–	–	–	0.732*
[17]	–	–	–	–	–	–	0.402*
[16]	–	55	–	–	–	–	0.141*

References

[1] Bost, R., Popa, R.A., Tu, S., Goldwasser, S.: Machine learning classification over encrypted data. In: Proceedings of the 22nd Annual Network and Distributed System Security Symposium (February 8-11, 2015, San Diego, USA). The Internet Society, Reston (2015). https://doi.org/10.14722/ndss.2015.23241

[2] Brakerski, Z., Gentry, C., Vaikuntanathan, V.: (Leveled) fully homomorphic encryption without bootstrapping. ACM Transactions on Computation Theory **6**(3), 13–11336 (2014). https://doi.org/10.1145/2633600

[3] Brakerski, Z.: Fully homomorphic encryption without modulus switching from classical GapSVP. In: Safavi-Naini, R., Canetti, R. (eds.) Advances in Cryptology – Proc CRYPTO 2012 (August 19–23, 2012, Santa Barbara, CA, USA). Lecture Notes in Computer Science, vol. 7417, pp. 868–886. Springer, Heidelberg (2012). http://doi.org/10.1007/978-3-642-32009-5_50

[4] Fan, J., Vercauteren, F.: Somewhat Practical Fully Homomorphic Encryption. Cryptology ePrint Archive https://eprint.iacr.org/2012/144 (2012)

[5] Lyubashevsky, V., Peikert, C., Regev, O.: On ideal lattices and learning with errors over rings. Journal of ACM **60**(6), 43–135 (2013). https://doi.org/10.1145/2535925

[6] HElib:: An implementation of homomorphic encryption. https://github.com/homenc/HElib (Accessed in August, 2021)

[7] Dua, D., Graff, C.: UCI Machine Learning Repository. http://archive.ics.uci.edu/ml (2017)

[8] Sun, X., Yu, F.R., Zhang, P., Xie, W., Peng, X.: A survey on secure computation based on homomorphic encryption in vehicular ad hoc networks. Sensors **20**(15), 4253–131 (2020). https://doi.org/10.3390/s20154253

[9] Wood, A., Najarian, K., Kahrobaei, D.: Homomorphic encryption for machine learning in medicine and bioinformatics. Journal of ACM Computing Surveys **53**(4), 70–135 (2020). https://doi.org/10.1145/3394658

[10] Domingos, P., Pazzani, M.: On the optimality of the simple Bayesian classifier under zero-one loss. Machine Learning **29**(2), 103–130 (1997). https://doi.org/10.1023/A:1007413511361

[11] Goldwasser, S., Micali, S.: Probabilistic encryption & how to play mental poker keeping secret all partial information. In:

Rabin, T. (ed.) STOC '82: Proceedings of the Fourteenth Annual ACM Symposium on Theory of Computing (San Francisco, USA, May 5 - 7, 1982), pp. 365–377. ACM, New York (1982). https://doi.org/10.1145/800070.802212

[12] Paillier, P.: Public-key cryptosystems based on composite degree residuosity classes. In: Stern, J. (ed.) Advances in Cryptology – EUROCRYPT '99 (May 2–6, 1999, Prague, Czech Republic). Lecture Notes in Computer Science, vol. 1592, pp. 223–238. Springer, Heidelberg (1999). https://doi.org/10.1007/3-540-48910-X_16

[13] Li, X., Zhu, Y., Wang, J.: Secure naïve bayesian classification over encrypted data in cloud. In: Chen, L., Han, J. (eds.) Proceedings of the 10th International Conference on Provable Security (Nanjing, China, November 10-11, 2016). Lecture Notes in Computer Science, vol. 10005, pp. 130–150. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-47422-9_8

[14] Kim, S., Omori, M., Hayashi, T., Omori, T., Wang, L., Ozawa, S.: Privacy-preserving naive Bayes classification using fully homomorphic encryption. In: Cheng, L., Leung, A.C.S., Ozawa, S. (eds.) Neural Information Processing – Proceedings of the 25th International Conference on Neural Information Processing (Siem Reap, Cambodia, December 13–16, 2018). Lecture Notes in Computer Science, vol. 11304, pp. 349–358. Springer, Cham (2018). https://doi.org/10.1007/978-3-030-04212-7_30

[15] Yasumura, Y., Ishimaki, Y., Yamana, H.: Secure naïve Bayes classification protocol over encrypted data using fully homomorphic encryption. In: Indrawan-Santiago, M., Pardede, E., Salvadori, I.L., Steinbauer, M., Khalil, I., Anderst-Kotsis, G. (eds.) Proceedings of the 21st International Conference on Information Integration and Web-Based Applications & Services (Munich, Germany, December 2–4, 2019), pp. 45–54. ACM, New York (2019). https://doi.org/10.1145/3366030.3366056

[16] Sun, X., Zhang, P., Liu, J.K., Yu, J., Xie, W.: Private machine learning classification based on fully homomorphic encryption. IEEE Transactions on Emerging Topics in Computing **8**(2), 352–364 (2020). https://doi.org/10.1109/TETC.2018.2794611

[17] Wood, A., Shpilrain, V., Najarian, K., Kahrobaei, D.: Private naive bayes classification of personal biomedical data: Application in cancer data analysis. Computers in Biology and Medicine **105**, 144–150 (2019). https://doi.org/10.1016/j.compbiomed.2018.11.018

[18] Gribov, A., Kahrobaei, D., Shpilrain, V.: Practical private-key fully homomorphic encryption in rings. Groups Complexity Cryptology **10**(1), 17–27 (2018). https://doi.org/10.1515/gcc-2018-0006

[19] Chen, J., Feng, Y., Liu, Y., Wu, W., Yang, G.: Non-interactive privacy-preserving naïve Bayes classifier using homomorphic encryption. In: A, B. (ed.) SPNCE: International Conference on Security and Privacy in New Computing Environments (Qinhuangdao, China, December 10–11, 2021). Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering. Springer, Cham (2022). https://www.arcnl.org/jchen/download/naiveBayes.pdf

[20] Goldreich, O.: Foundations of Cryptography – Basic Applications. Cambridge University Press, Cambridge (2004)

[21] Cheon, J.H., Kim, A., Kim, M., Song, Y.: Homomorphic encryption for arithmetic of approximate numbers. In: Takagi, T., Peyrin, T. (eds.) Proceedings of ASIACRYPT 2017 – 23rd International Conference on the Theory and Applications of Cryptology and Information Security (December 3-7, 2017, Hong Kong, China), Part I. Lecture Notes in Computer Science, vol. 10624, pp. 409–437. Springer, Heidelberg (2017). https://doi.org/10.1007/978-3-319-70694-8_15

[22] Rivest, R., Adleman, L., Dertouzos, M.: On data banks and privacy homomorphisms. In: DeMillo, R.A., Dobkin, D.P., Jones, A.K.,

Lipton, R.J. (eds.) Foundations of Secure Computation, pp. 165–179. Academic Press, Atlanta (1978)

[23] Gentry, C.: Fully homomorphic encryption using ideal lattices. In: Mitzenmacher, M. (ed.) Proceedings of the Forty-first Annual ACM Symposium on Theory of Computing (May 31 - June 2, 2009, Bethesda, USA), pp. 169–178. ACM, New York (2009). https://doi.org/10.1145/1536414.1536440

[24] Ducas, L., Micciancio, D.: FHEW: Bootstrapping homomorphic encryption in less than a second. In: Oswald, E., Fischlin, M. (eds.) Advances in Cryptology - Proceedings of EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Part I (April 26-30, 2015, Sofia, Bulgaria). Lecture Notes in Computer Science, vol. 9056, pp. 617–640. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-46800-5_24

[25] Chillotti, I., Gama, N., Georgieva, M., Izabachène, M.: TFHE: Fast fully homomorphic encryption over the torus. Journal of Cryptology **33**(1), 34–91 (2020). https://doi.org/10.1007/s00145-019-09319-x

[26] Gentry, C., Halevi, S., Smart, N.P.: Homomorphic evaluation of the AES circuit. In: Safavi-Naini, R., Canetti, R. (eds.) Advances in Cryptology – Proc CRYPTO 2012 (August 19–23, 2012, Santa Barbara, USA). Lecture Notes in Computer Science, vol. 7417, pp. 850–867. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-32009-5_49, updated implementation in 2015 is available from http://eprint.iacr.org/2012/099

[27] Smart, N.P., Vercauteren, F.: Fully homomorphic SIMD operations. Designs, Codes and Cryptography **71**(1), 57–81 (2014). https://doi.org/10.1007/s10623-012-9720-4

[28] Halevi, S., Shoup, V.: Design and implementation of HElib: a homomorphic encryption library. Cryptology ePrint Archive https://eprint.iacr.org/2020/1481 (2020)

[29] Iliashenko, I., Zucca, V.: Faster homomorphic comparison operations for BGV and BFV. Proceedings on Privacy Enhancing Technologies **2021**(3), 246–264 (2021). https://doi.org/10.2478/popets-2021-0046

[30] Tschiatschek, S., Reinprecht, P., Mücke, M., Pernkopf, F.: Bayesian network classifiers with reduced precision parameters. In: Flach, P.A., De Bie, T., Cristianini, N. (eds.) Proceedings of ECML PKDD 2012: Joint European Conference on Machine Learning and Knowledge Discovery in Databases (Bristol, UK, September 24-28, 2012). Lecture Notes in Computer Science, vol. 7523, pp. 74–89. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-33460-3_10

[31] Ducas, L., Stehlé, D.: Sanitization of FHE ciphertexts. In: Fischlin, M., Coron, J.-S. (eds.) Advances in Cryptology – EUROCRYPT 2016, Part I (Vienna, Austria, May 8-12, 2016). Lecture Notes in Computer Science, vol. 9665, pp. 294–310. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-49890-3_12

[32] Gentry, C., Halevi, S., Vaikuntanathan, V.: *i*-Hop homomorphic encryption and rerandomizable Yao circuits. In: Rabin, T. (ed.) Advances in Cryptology – CRYPTO 2010 (Santa Barbara, USA, August 15-19, 2010). Lecture Notes in Computer Science, vol. 6223, pp. 155–172. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-14623-7_9