IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received 3 April 2023, accepted 21 April 2023, date of publication 28 April 2023, date of current version 9 May 2023.

Digital Object Identifier 10.1109/ACCESS.2023.3271319

RESEARCH ARTICLE

Verifiable Homomorphic Secret Sharing for Machine Learning Classifiers

XIN CHEN
School of Information Science and Technology, ShanghaiTech University, Shanghai 201210, China
Shanghai Institute of Microsystem and Information Technology, Chinese Academy of Sciences, Shanghai 200050, China
University of Chinese Academy of Sciences, Beijing 101408, China
e-mail: chenxin3@shanghaitech.edu.cn

ABSTRACT When using machine learning classifiers to classify data in cloud computing, it is crucial to maintain data privacy and ensure the correctness of classification results. To address these security concerns, we propose a new verifiable homomorphic secret sharing (VHSS) scheme. Our approach involves distributing the task of executing a polynomial form of the machine learning classifier among two servers who produce partial results on encrypted data. Each server cannot obtain any data information, and the classification result can be reconstructed and verified using a verification key in conjunction with the two partial results. Compared to previous VHSS schemes, our scheme can compute the degree of polynomials as high as the polynomials in the system security parameters while performing comparably to homomorphic secret sharing (HSS) schemes. We implement our proposed scheme and demonstrate its application to decision trees (a type of machine learning classifier). Our experiments show that our scheme is twice as fast as previous VHSS schemes when evaluating decision trees with depths ranging from 2–14.

INDEX TERMS Data privacy, verification, homomorphic secret sharing, decision trees, multi server, cloud computing.

I. INTRODUCTION

Machine learning classification has been widely used in medical predictions, credit investment and financial risk prediction [39], [40]. The typical process of using a machine learning classifier may generally be as follows: a company trains a machine learning classifier, and the users send their data to the company to classify the data. When faced with numerous classification requests, the company needs to maintain certain hardware resources to ensure that the classification requests can be responded to in time. With powerful computing capabilities and popular prices, cloud computing can free the company from heavy computations, and the company no longer needs to own or maintain hardware facilities. In more detail, the users outsource the data $x_1, \ldots, x_n$ to the servers, and the company sends the classifier f to the servers. The servers compute and return the result $y = f(x_1, \ldots, x_n)$ to the users. However, this scenario has significant risks [14], [24], [28], [42]. At first, data is outsourced to the servers in plain text, which can easily lead to data leakage. Secondly, the servers return wrong results when hijacked or choose to execute the computing task $f' \neq f$ with lower computational complexity to reduce costs.

A natural idea to solve the first problem is to use fully homomorphic encryption (FHE) [23]. The company employs existing technology [35], [41] to convert the machine learning classifier into a polynomial. Then, the server can homomorphically compute the ciphertext of y on FHE-encrypted ciphertexts of $x_1, \ldots, x_n$. However, while FHE can safeguard data privacy, it may not be practical in terms of performance [30]. Alternatively, Homomorphic Secret Sharing (HSS) [7] offers an efficient and multi-server version of FHE. Specifically, HSS by BKS [7] eliminates costly key-switching and modulus-reduction steps used in FHE.

Although HSS can protect the data privacy, none of the current HSS schemes can verify the correctness of the results of machine learning classifiers. Currently, there are five works [16], [17], [36], [37], [38] that focus on verifiable HSS (VHSS) schemes. These schemes ensure both data privacy and result verification. However, these VHSS schemes may

The associate editor coordinating the review of this manuscript and approving it for publication was Massimo Cafaro.

not be suitable for machine learning classifiers due to their limitations. The VHSS scheme proposed by Tsaloli et al. [36] has been proven to be insecure [26]. Although Tsaloli et al. proposed two secure VHSS schemes in [37] and [38], they only support computation of linear polynomials which is insufficient for machine learning classifiers that often require higher degree polynomials.

There is also a VHSS scheme [17] that supports computing higher degree polynomials, but its capability is limited by the number of servers. As the degree of the polynomial increases, more servers are required to perform the computation. The VHSS of CZ [16] is based on the HSS of BKS [7]. In order to ensure verifiability, the servers of CZ must create a tag for every computation instruction, resulting in twice the amount of computation performed by BKS's servers. In summary, the current VHSS schemes do not meet the requirements for machine learning classifiers. They fail to fulfill three key criteria: (1) capacity to compute high-degree polynomial form of machine learning classifiers; (2) ability to increase the degree of polynomial without being restricted by server numbers; and (3) capability to achieve performance comparable to that of HSS of BKS [7].

A. OUR CONTRIBUTION

The hypothesis of our research is that a VHSS scheme can meet requirements (1)-(3). We introduce the first VHSS scheme that satisfies these requirements and enables privacy-preserving and verifiable machine learning classification in cloud computing. In our scheme, the users send the ciphertexts of the data to two servers, and the company converts the machine learning classifier into a polynomial and distributes it to both servers. Each server independently execute the polynomial on the ciphertexts without interaction, and sends a partial result to the users for reconstruction and verification of the final classification result. Our contributions are as follows:

- ***Privacy preservation:*** We use *Public-Key Encryption with Nearly Linear Decryption* (PKE-NLD) to encrypt data to protect data privacy. PKE-NLD ensures that any user holding the public key encrypts its data and any single server can not obtain information about the data.
- ***High-degree polynomial:*** With PKE-NLD, our scheme enables $\mathsf{poly}(\lambda)$ (a polynomial function in the security parameter λ) times distributed additions and multiplications between two servers. This enables the computation of polynomials with a degree of $\mathsf{poly}(\lambda)$.
- ***Verification:*** The servers need to compute a tag for the classification result. The correctness of the result can be verified by using the tag and the verification key, and any probabilistic polynomial-time malicious adversary cannot convince the users to accept a wrong result.
- ***Efficient server computing:*** To improve server-side efficiency, we have applied new server-side algorithms to avoid generating a tag for each computation instruction. Experiments show that the computational complexity of our scheme on the server side is only half that of CZ.

B. RELATED WORK

1) DIFFERENT PRIVACY (DP)

DP guarantees that the server can compute the function f on appropriately calibrated noisy data without exposing any confidential information. Despite the existence of various DP models, such as Local DP [19], Metric DP [10], and Shuffled DP [3], DP alone cannot verify the correctness of the result y. Moreover, the primary constraint of DP is that the client can only obtain a *result with noise*, rather than the original result y. Our scheme, however, achieves both data privacy protection and result verification while obtaining *noise-free result*.

2) MULTIPLICATIVE SECRET SHARING (MSS)

The use of MSS [8] enables the sharing of a set of data $x_1, \ldots, x_d$ among multiple servers. Each server can compute a partial result with its share of the data and the sum of all partial results is equal to $\prod_{i=1}^{d} x_i$. The MSS scheme constructed by Barkol et al. [8] can achieve information theory security and have a flexible access structure. At present, there is no literature or technology to show that the MSS scheme can be transformed into a scheme that supports the computation of general polynomials. Compared with our VHSS model, MSS requires a *non-constant number* of servers and only allows the computation of *monomials*.

3) HOMOMORPHIC SIGNATURES/MACs

Homomorphic signatures (HomSigs) [27] enable a user to sign a set of data $x_1, \ldots, x_n$ using a secret key, and a server can then homomorphically compute a function f to obtain both $y = f(x_1, \ldots, x_n)$ and a short signature μ for y. The resulting (μ, y) can be verified by anyone using a public key. Existing HomSigs [12], [22] rely on a single server and expose *the data in clear* to the server. In contrast, Homomorphic MACs [11], [25] are symmetric-key alternatives to HomSigs that use a single server and leave *the data in clear*.

4) VERIFIABLE COMPUTATION (VC)

The VC model consists of two phases [2]. During the offline phase, the company generates an encoding of f to send to the server. In the online phase, the user encodes the data to be processed by the server. The server then responds with an encoding of the result y and a proof, and the user verifies the encoding before reconstructing y. Some single-server VC schemes [2], [5], [13], [43] ensure data privacy, but rely on *low-efficiency FHE* or only support *matrix-vector multiplication*. In contrast, our scheme can compute *polynomial functions without FHE*. Furthermore, while multiple-server VC schemes [1], [15] have been explored, they either leave data unprotected or require complex interactions among servers, which our *non-interactive* scheme avoids.

C. ORGANIZATION

Section II includes the introduction of the participants in our system and the threat model. Section III introduces the notations and cryptographic primitives used in our scheme. In Section IV, we introduce our scheme and analyze its security. In Section V, we implement the proposed scheme and show its performance. Finally, our discussions and conclusions are presented in Sections VI and VII, respectively.

II. PRELIMINARIES AND CRYPTOGRAPHIC PRIMITIVES

We denote with $\lambda \in \mathbb{N}$ a security parameter, use $\mathsf{poly}(\lambda)$ to denote the polynomial function in λ, and use $\mathsf{negl}(\lambda)$ to denote the negligible function in λ. Let PPT denote probabilistic polynomial time.

For any integer $n > 0$, we denote $[n] = \{1, \ldots, n\}$. For any finite set S, we denote by $s \leftarrow S$ the process of sampling s uniformly at random from S. For any real number $a \in \mathbb{R}$, $\lfloor a \rceil \in \mathbb{Z}$ is the integer closest to a, where we round up if the first decimal place of a is ≥ 5. We denote by $R = \mathbb{Z}[X]/(X^N + 1)$ the ring of integer-coefficient polynomials modulo $X^N + 1$. For any integer $p > 0$, we denote $R_p = R/pR = \mathbb{Z}_p[X]/(X^N + 1)$. For any $a = a^{(0)} + \cdots + a^{(N-1)}X^{N-1} \in R$, $\|a\|_\infty = max_{i=0}^{N-1}|a^{(i)}|$ is the *infinity norm* of a.

A. PUBLIC-KEY ENCRYPTION WITH NEARLY LINEAR DECRYPTION (PKE-NLD)

PKE-NLD was introduced by Boyle et al. [7], and its security under the Ring-LWE (RLWE) problem [29]. PKE-NLD is called near-linear decryption because its decryption algorithm first applies a linear function to the ciphertext and then performs rounding.

Compared with a general PKE, a PKE-NLD has two additional algorithms: the *key-dependent message oracle (OKDM)* and the *distributed decryption (DDec)*. OKDM allows anyone to use the public key pk and the data x to compute an encryption of $x \cdot \mathsf{sk}$, without knowing the secret key sk. DDec allows two servers to perform decryptions distributively: by running DDec, each server can use an additive share of sk to decrypt a ciphertext of x to compute an additive share of x, and each server can also use an additive share of $x \cdot \mathsf{sk}$ to decrypt a ciphertext of x' to compute an additive share of $x \cdot x'$.

An RLWE-based PKE-NLD (Figure. 1) may be parameterized by:

- $\kappa \in \mathbb{N}$, a correctness parameter (each Mult instruction in our scheme gives the correct result with probability $\geq 1 - 2^{-\kappa}$).
- $B_{\mathsf{msg}} \in \mathbb{N}$, a power of 2 and a magnitude bound to ensure that the value of any intermediate result in the process of computing $y = f(x_1, \ldots, x_n)$ does not exceed B_{msg}.
- $R = \mathbb{Z}[X]/(X^N + 1)$, a ring where N is a power of 2.
- $\mathcal{D}_{\mathsf{sk}}$, a distribution over R from which the secret key is sampled.

- $\mathsf{PKE.Gen}(1^\lambda)$: On input a security parameter λ, the key generation algorithm randomly samples $a \leftarrow R_q$, $\hat{s} \leftarrow \mathcal{D}_{\mathsf{sk}}$ and $e \leftarrow \mathcal{D}_{\mathsf{err}}$. Next, it sets $b = a \cdot \hat{s} + e \bmod q$ and $\mathbf{s} = (1, \hat{s}) \in R_q^2$. Finally, it outputs a public key $\mathsf{pk} = (a, b)$ and a secret key $\mathsf{sk} = \mathbf{s}$.
- $\mathsf{PKE.Enc}(\mathsf{pk}, x)$: On input a public key $\mathsf{pk} = (a, b)$ and a message $x \in R_p$, the encryption algorithm samples $u \leftarrow \mathcal{D}_{\mathsf{sk}}$ and $e_1, e_2 \leftarrow \mathcal{D}_{\mathsf{err}}$. Next, it sets $c_1 = bu + e_1 + (q/p) \cdot x \bmod q$ and $c_2 = -au + e_2 \bmod q$. Finally, it outputs a ciphertext $\mathbf{c}^x = (c_1, c_2)$ of message x.
- $\mathsf{PKE.Dec}(\mathsf{sk}, \mathbf{c}^x)$: On input a secret key $\mathsf{sk} = \mathbf{s}$ and a ciphertext $\mathbf{c}^x = (c_1, c_2)$, the decryption algorithm outputs $x = \lfloor (p/q)\langle \mathbf{s}, \mathbf{c}^x \rangle \rceil \bmod p$.
  // $(p/q)\langle \mathbf{s}, \mathbf{c}^x \rangle = (p/q)(1 \cdot c_1 + \hat{s} \cdot c_2)$ is a linear function of sk. Rounding is performed after applying this linear function to the ciphertext. This is called a *nearly linear decryption*.
- $\mathsf{PKE.OKDM}(\mathsf{pk}, x)$: On input a public key $\mathsf{pk} = (a, b)$ and a message $x \in R_p$, the key-dependent message oracle compute $\mathbf{c}^{x \cdot 1} = \mathsf{PKENLD.Enc}(\mathsf{pk}, x)$ and $\mathbf{c}^{x \cdot \hat{s}} = \mathsf{PKE.Enc}(\mathsf{pk}, 0) + (0, (q/p) \cdot x) \bmod q$. Next, it outputs a 2-dimensional vector $\mathbf{C}^x = (\mathbf{c}^{x \cdot 1}, \mathbf{c}^{x \cdot \hat{s}})$ as an encryption of $x \cdot \mathbf{s} = (x \cdot 1, x \cdot \hat{s})$.
- $\mathsf{PKE.DDec}(\mathsf{pk}, x)$: On input an additive share of $x \cdot \mathbf{s} \in (R_p)^2$ and a ciphertext $\mathbf{c}^{x'} = (c_1, c_2) \in R_q^2$ of $x' \in R_p$, the distributed decryption algorithm outputs an additive share $(\lfloor (p/q) \cdot (c_1 \cdot t_b + c_2 \cdot \widehat{t_b}) \rceil \bmod p) \bmod q$ of $x \cdot x' \in R_p$.
  // If the input $\mathbf{t}_b^x \in R_q^2$ is replaced with an additive share $\mathbf{s}_b \in R_q^2$ of the secret key $\mathsf{sk} = \mathbf{s} \in R_q^2$, then the output will be an additive share of $x' \in R_p$. Although reducing modulo q has no effect on the output value, it is included to indicate that the output of this algorithm is considered as an element of R_q.

FIGURE 1. RLWE-based instantiation of PKE-NLD [7].

- $h_{\mathsf{sk}} \in \mathbb{N}$, a power of 2 and an upper bound on the number of non-zero terms in the secret key sampled from $\mathcal{D}_{\mathsf{sk}}$.
- $B_{\mathsf{sk}} \in \mathbb{N}$, a power of 2 and an upper bound on the infinity norm of the secret key sampled from $\mathcal{D}_{\mathsf{sk}}$.
- $\mathcal{D}_{\mathsf{err}}$, an error distribution over R.
- $B_{\mathsf{err}} \in \mathbb{N}$, an upper bound on the infinity norm of the error sampled from $\mathcal{D}_{\mathsf{err}}$.
- $p, q \in \mathbb{N}$, two powers of 2 such that $p = 2^{\kappa+2} \cdot B_{\mathsf{msg}} \cdot N \cdot h_{\mathsf{sk}} \cdot B_{\mathsf{sk}}$, $q \geq p \cdot 2^{2\kappa+3} \cdot B_{\mathsf{msg}} \cdot N^2 \cdot (2h_{\mathsf{sk}} + 1) \cdot B_{\mathsf{err}}$ and $p|q$.
- $\mathcal{M} = R_p$, a message space.
- $\mathcal{C} = R_q^2$, a ciphertext space.

III. SYSTEM MODEL

First, we introduce VHSS system model that can be used for privacy-preserving and verifiable machine learning

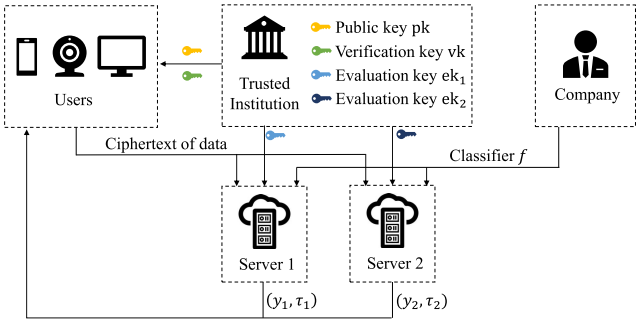

FIGURE 2. An illustration of system.

classification in cloud computing. The syntax of the building blocks of our model is as follows:

- $\mathsf{Gen}(1^\lambda)$: On input a security parameter 1^λ, the *key generation algorithm* outputs a *public key* pk, a *verification key* vk and a pair $(\mathsf{ek}_1, \mathsf{ek}_2)$ of *(private) evaluation keys*.
- $\mathsf{Enc}(\mathsf{pk}, x)$: On input a public key pk and a message $x \in \mathcal{M}$, the *encryption algorithm* outputs a ciphertext $\mathsf{ct} \in \mathcal{C}$, where $\mathcal{C}$ is the ciphertext space.
- $\mathsf{Eval}(b, \mathsf{ek}_b, (\mathsf{ct}_1, \ldots, \mathsf{ct}_n), f)$: On input an evaluation key ek_b, a vector $(\mathsf{ct}_1, \ldots, \mathsf{ct}_n)$ of n ciphertexts and a function $f(x_1, \ldots, x_n)$, the bth ($b \in [2]$) server's *evaluation algorithm* outputs a partial result (y_b, τ_b).
- $\mathsf{Ver}(\mathsf{vk}, ((y_1, \tau_1), (y_2, \tau_2)))$: On input a public verification key pvk and two partial result $(y_1, \tau_1), (y_2, \tau_2)$, the *verification algorithm* outputs a result y (which is believed to be the correct result) or an error symbol $\perp$ (to indicate that at least one of the servers is cheating).

In our model, there are four types of participants: a trusted institution, multiple users, a company with machine learning classifiers and two non-communicating servers $\mathcal{S}_1, \mathcal{S}_2$. We provide an intuitive illustration of the system in Figure 2. In general, our VHSS scheme is executed as follows.

1. The trusted institution runs $\mathsf{Gen}(1^\lambda)$ to generate the keys $(\mathsf{pk}, \mathsf{vk}, (\mathsf{ek}_1, \mathsf{ek}_2))$. The public key pk is sent to all participants. The evaluation key ek_b is sent to the server $\mathcal{S}_b$. The verification key vk is sent to the users.
2. Every user encrypts its data x into a ciphertext $\mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{pk}, x)$ and uploads ct to both servers. And, the company converts the machine learning classifier into a polynomial function f, and sends f to both servers.
3. Each server $\mathcal{S}_b$ locally computes an additive share y_b (i.e. $y = y_1 + y_2 \mod q$) and its authentication τ_b of $y = f(x_1, \ldots, x_n)$ using the evaluation key ek on the ciphertext $\mathsf{ct}_1, \ldots, \mathsf{ct}_n$.
4. Each user uses y_1 and y_2 returned by the servers to reconstruct y, and uses the verification key vk and τ_1, τ_2 to verify the correctness of y.

In our scheme, the trusted institution and the users are considered to be honest and not colluding with any other participants. Our scheme needs to rely on a trusted authority to distribute keys, which may imply a single point of failure. In future work, we will explore models that do not rely on trusted authorities. Users want to get correct classification results, then they have no incentive to break the normal operation of the scheme. Two non-communicating servers are considered malicious. Each server has the incentive to obtain information about the data. The servers may return wrong results for some economic reasons and internal or external attacks. Furthermore, we adopt the same assumption in related works [4], [6], [7], [16], [20], [21], [31], [33] that the two servers do not collude. It is feasible to consider two non-colluding servers in reality, and the servers can be rented from two different cloud server providers to ensure that the two servers do not collude.

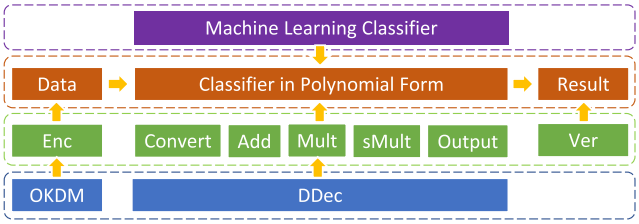

FIGURE 3. The framework of our scheme.

IV. OUR SCHEME

In this section, we introduce a new VHSS scheme for privacy-preserving and verifiable machine learning classification in cloud computing and perform a security analysis of our scheme.

A. THE CONSTRUCTION OF OUR SCHEME

We show the framework of our scheme in Figure 3 to give a concise overview. At first, the machine learning classifier is equivalently transformed into a multivariate polynomial function by existing techniques [35], [41]. In order to protect the privacy of data, we use OKDM algorithm of PKE-NLD to encrypt data. The computation of a polynomial over the ciphertext of data can be composed of five basic instructions: Conversion (Convert), Addition (Add), Multiplication (Mult), Scalar Multiplication (sMult) and Output (Output). These instructions are based on the DDec algorithm of PKE-NLD. Finally, the classification result is verified by the Verification (Ver) algorithm.

As mentioned in Section III, we divided our scheme into four stages: key generation, encryption, evaluation, and verification. Below we describe these four stages in detail.

1) KEY GENERATION

This stage is mainly responsible by trusted institution, including *parameter generation, key generation and key distribution.*

- *Parameter generation.* First, the trusted institution chooses the upper bound B_{msg} (powers of 2) to ensure that the value of any intermediate result in the process of computing $y = f(x_1, \ldots, x_n)$ does not exceed B_{msg}. Next, the trusted institution selects the appropriate

parameter for PKE-NLD according to B_{msg}. The parameters for PKE-NLD include distributions $\mathcal{D}_{\mathsf{sk}}$, $\mathcal{D}_{\mathsf{err}}$, bounds $B_{\mathsf{sk}}, B_{\mathsf{err}} \in \mathbb{N}$, the number h_{sk} of non-zero terms in a secret key, a polynomial ring $R = \mathbb{Z}[X]/(X^N + 1)$, a message space $\mathcal{M} = R_p$, a ciphertext space $\mathcal{C} = R_q^2$, where $B_{\mathsf{sk}}, h_{\mathsf{sk}}, N, p, q$ are powers of $2, p = 2^{\kappa+2} \cdot B_{\mathsf{msg}} \cdot N \cdot h_{\mathsf{sk}} \cdot B_{\mathsf{sk}}, q \geq p \cdot 2^{2\kappa+3} \cdot B_{\mathsf{msg}} \cdot N^2 \cdot (2h_{\mathsf{sk}} + 1) \cdot B_{\mathsf{err}}$.

- *Key generation.* First, the trusted institution generates a key pair $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{PKE.Gen}(1^\lambda)$, where $\mathsf{sk} = s = (1, \widehat{s}) \in R_q^2$ and $\|\widehat{s}\|_\infty \leq B_{\mathsf{sk}}$. Next, the trusted institution randomly chooses $s_1 \leftarrow R_q^2$ and computes $s_2 = \mathsf{sk} - s_1 \bmod q$. Then, the trusted institution randomly chooses $\alpha = (\alpha^{(0)}, \ldots, \alpha^{(N-1)}) \in R$ and $\|\alpha\|_\infty \leq 1$, and lets verification key $\mathsf{vk} = \alpha$. After that, the trusted institution computes $C^\alpha \leftarrow \mathsf{PKE.OKDM}(\mathsf{pk}, \alpha)$, where C^α is a ciphertext of $\alpha \cdot s = (\alpha, \alpha \cdot \hat{s})$. Finally, the trusted institution draws a key $K \leftarrow \mathcal{K}$ for a pseudorandom function $\mathsf{PRF} : \mathcal{K} \times \{0,1\}^* \to R_q^2$ and sets the evaluation key $\mathsf{ek}_b = (s_b, C^\alpha, K)$ for $b = 1, 2$.
- *Key distribution.* First, the trusted institution distributes the public key pk to all participants. Next, the trusted institution distributes the evaluation key ek_b to the server $\mathcal{S}_b$ for $b = 1, 2$. Finally, the trusted institution distributes the verification key vk to the users.

2) ENCRYPTION

At first, each user encrypts its data x as $C^x \leftarrow \mathsf{PKE.OKDM}(\mathsf{pk}, x)$, where C^x is a ciphertext of $x \cdot s = (x, x \cdot \hat{s})$. Next, each user sends the ciphertext C^x to the servers $\mathcal{S}_1$ and $\mathcal{S}_2$. Given C^{x_i} for $i \in [n]$, each server $\mathcal{S}_b$ $(b \in [2])$ cannot obtain any information about x_i, and each server can use an evaluation key ek_b to locally compute an additive share of $y = P(x_1, \ldots, x_n)$ with C^{x_i} for $i \in [n]$.

3) EVALUATION

The evaluation algorithm of our scheme is divided into the following five instructions: conversion $\mathsf{Convert}$, addition Add, multiplication Mult, scalar multiplication sMult and output Output. Each server will parse polynomial form f as a sequence of these 5 instructions, which are sorted according to a unique identifier $\mathsf{id} \in \{0,1\}^*$. Each server will execute these instructions in order of identifier id. Next, we will describe these instructions in detail.

- *Conversion* $\mathsf{Convert}$. The instruction $\mathsf{Convert}$ can convert a ciphertext $C^x = (c^x, c^{x \cdot \hat{s}})$ of $(x, x \cdot \hat{s})$ into an additive share of $(x, x \cdot \hat{s})$. $\mathsf{Convert}$ is essentially based on the DDec algorithm in PKE-NLD, which allows each server uses an additive share s_b of sk to decrypt a ciphertext of $(x, x \cdot \hat{s})$ to obtain an additive share of $(x, x \cdot \hat{s})$. $\mathsf{Convert}$ is a basic instruction. The remaining four algorithms all take additive shares as one of the inputs. The formal definition of the $\mathsf{Convert}$ is as follows.
  $\mathsf{Convert}(\mathsf{id}, \mathsf{ek}_b, C^x)$: the server $\mathcal{S}_b$ computes
$$t_b^x \leftarrow (\mathsf{DDec}(s_b, c^x), \mathsf{DDec}(s_b, c^{x \cdot \hat{s}})) + (2b-3) \cdot \mathsf{PRF}(K, \mathsf{id}) \bmod q,$$
  where t_b^x is an additive share of $(x, x \cdot \hat{s})$, i.e. $(x, x \cdot \hat{s}) = t_1^x + t_2^x \mod q$.
- *Addition* Add. The essence of Add is to use the property of additive share to add t_b^x and $t_b^{x'}$ to obtain an additive share $t_b^{x+x'}$ of $(x + x', (x + x') \cdot \hat{s})$. The formal definition of Add is as follows.
  $\mathsf{Add}(\mathsf{id}, \mathsf{ek}_b, t_b^x, t_b^{x'})$: the server $\mathcal{S}_b$ computes
$$t_b^{x+x'} \leftarrow t_b^x + t_b^{x'} + (2b-3) \cdot \mathsf{PRF}(K, \mathsf{id}) \bmod q \mod q,$$
  where $t_b^{x+x'}$ is an additive share of $(x + x', (x + x') \cdot \hat{s})$, i.e. $(x + x', (x + x') \cdot \hat{s}) = t_1^{x+x'} + t_2^{x+x'} \mod q$.
- *Multiplication* Mult. Mult is essentially based on the DDec algorithm in PKE-NLD, which allows each server can use an additive share t_b^x of $(x, x \cdot \hat{s})$ to decrypt a ciphertext $C^{x'} = (c^{x'}, c^{x' \cdot \hat{s}})$ of $(x', x' \cdot \hat{s})$ to obtain an additive share $t_b^{x \cdot x'}$ of $(x', x' \cdot \hat{s})$. The formal definition of Mult is as follows.
  $\mathsf{Mult}(\mathsf{id}, \mathsf{ek}_b, t_b^x, C^{x'})$: the server $\mathcal{S}_b$ computes
$$t_b^{x \cdot x'} \leftarrow (\mathsf{DDec}(t_b^x, c^{x'}), \mathsf{DDec}(t_b^x, c^{x' \cdot \hat{s}})) + (2b-3) \cdot \mathsf{PRF}(K, \mathsf{id}) \bmod q,$$
  where $t_b^{x \cdot x'}$ is an additive share of $(x, x \cdot \hat{s})$, i.e. $(x \cdot x', x \cdot x' \cdot \hat{s}) = t_1^{x \cdot x'} + t_2^{x \cdot x'} \mod q$.
  The 1st component of $t_b^{x \cdot x'}$ is an additive share of $x \cdot x'$. Note that the 2nd component of $t_b^{x \cdot x'}$ does not contribute to the reconstruction of $x \cdot x'$. However, it is required to enable further multiplications. For example, decrypting $C^{x''}$ with $t_b^{x \cdot x'}$ will give an additive share of $x \cdot x' \cdot x'' \cdot s$.
- *Scalar Multiplication* sMult. sMult can obtain an additive share of $(c \cdot x, (c \cdot x) \cdot \hat{s})$ from a constant $c \in R_p$ and an additive share of $(x, x \cdot \hat{s})$. The formal definition of sMult is as follows. $\mathsf{sMult}(\mathsf{id}, \mathsf{ek}_b, c, t_b^x)$: the server $\mathcal{S}_b$ computes
$$t_b^{c \cdot x} \leftarrow c \cdot t_b^x + (2b-3) \cdot \mathsf{PRF}(K, \mathsf{id}) \mod q,$$
  where $t_b^{c \cdot x}$ is an additive share of $(c \cdot x, c \cdot x \cdot \hat{s})$, i.e. $(c \cdot x, c \cdot x \cdot \hat{s}) = t_1^{c \cdot x} + t_2^{c \cdot x} \mod q$.
- *Output* Output. The above instructions finally result in each server $\mathcal{S}_b$ has $t_b^y = (t_b^y, \widehat{t_b^y})$ which is an additive share of $(y, y \cdot \hat{s})$. In particular, $t_b^y \in R_q$ is an additive share of $y \in R_p$. For result reconstruction, the Output instruction only needs to return t_b^y. For verification, each server $\mathcal{S}_b$ uses t_b^y, an additive share of $y \cdot s$, and $C^\alpha = (c^\alpha, c^{\alpha \cdot \hat{s}})$, a ciphertext of $(\alpha, \alpha \cdot \hat{s})$, to compute an additive share $\boldsymbol{\tau}_b = (\tau_b, \widehat{\tau_b})$ of $\alpha \cdot y \cdot s$. In particular, τ_1 and τ_2 are additive shares of $\alpha \cdot y$ in the ring R_q, i.e. $\tau_1 + \tau_2 = \alpha \cdot y \bmod q$. The formal definition of Output is as follows.
  $\mathsf{Output}(\mathsf{id}, \mathsf{ek}_b, t_b^y)$: the server $\mathcal{S}_b$ computes
$$\boldsymbol{\tau}_b \leftarrow (\mathsf{DDec}(t_b^y, c^\alpha), \mathsf{DDec}(t_b^y, c^{\alpha \cdot \hat{s}})) (2b-3) \cdot \mathsf{PRF}(K, \mathsf{id}) \bmod q.$$
  Next, the server $\mathcal{S}_b$ parses $t_b^y = (t_b^y, \widehat{t_b^y})$ and $\boldsymbol{\tau}_b = (\tau_b, \widehat{\tau_b})$. Finally, the server $\mathcal{S}_b$ outputs (t_b^y, τ_b).

4) VERIFICATION

The users receive (t_1^y, τ_1) and (t_2^y, τ_2) from the two servers respectively. To reconstruct y, the users compute $y = t_1^y + t_2^y \mod q$. To verify y, the company uses the verification key $\mathsf{vk} = \alpha$ to check $\mathsf{vk} \cdot y \stackrel{?}{=} \tau_1 + \tau_2 \mod q$.

B. SECURITY ANALYSIS

Our scheme focuses on two security properties: privacy preservation and verification. We will now examine how our scheme achieves these objectives.

1) CORRECTNESS

The property of correctness requires that the company always outputs the correct result y, if the scheme is faithfully executed. We prove that the subroutines of the evaluation algorithm and the verification algorithm maintain correctness as follows:

- Consider $\mathsf{Convert}(\mathsf{id}, \mathsf{ek}_b, C^x)$ for $b \in [2]$, we have $t_1^x + t_2^x \equiv (\mathsf{DDec}(s_1, c^x), \mathsf{DDec}(s_1, c^{x\cdot\hat{s}})) - \mathsf{PRF}(K, \mathsf{id}) + (\mathsf{DDec}(s_2, c^x), \mathsf{DDec}(s_2, c^{x\cdot\hat{s}})) + \mathsf{PRF}(K, \mathsf{id}) \equiv (\mathsf{DDec}(s, c^x), \mathsf{DDec}(s, c^{x\cdot\hat{s}})) \equiv (x, x\cdot\hat{s}) \pmod q$.
- Consider $\mathsf{Add}(\mathsf{id}, \mathsf{ek}_b, t_b^x, t_b^{x'})$ for $b \in [2]$, we have $t_1^{x+x'} + t_2^{x+x'} \equiv t_1^x + t_1^{x'} - \mathsf{PRF}(K, \mathsf{id}) + t_2^x + t_2^{x'} + \mathsf{PRF}(K, \mathsf{id}) \equiv (x + x', (x + x')\cdot\hat{s}) \pmod q$.
- Consider $\mathsf{Mult}(\mathsf{id}, \mathsf{ek}_b, t_b^x, C^{x'})$ for $b \in [2]$, we have $t_1^{x\cdot x'} + t_2^{x\cdot x'} \equiv (\mathsf{DDec}(t_1^x, c^{x'}), \mathsf{DDec}(t_1^x, c^{x'\cdot\hat{s}})) - \mathsf{PRF}(K, \mathsf{id}) + (\mathsf{DDec}(t_2^x, c^{x'}), \mathsf{DDec}(t_2^x, c^{x'\cdot\hat{s}})) + \mathsf{PRF}(K, \mathsf{id}) \equiv (\mathsf{DDec}(x\cdot s, c^{x'}), \mathsf{DDec}(x\cdot s, c^{x'\cdot\hat{s}})) \equiv (x\cdot x', (x\cdot x')\cdot\hat{s}) \pmod q$.
- Consider $\mathsf{sMult}(\mathsf{id}, \mathsf{ek}_b, c, t_b^x)$ for $b \in [2]$, we have $t_1^{c\cdot x} + t_2^{c\cdot x} \equiv c\cdot t_1^x - \mathsf{PRF}(K, \mathsf{id}) + c\cdot t_2^x + \mathsf{PRF}(K, \mathsf{id}) \equiv c\cdot(t_1^x + t_1^x) \equiv (c\cdot x, c\cdot x\cdot\hat{s}) \bmod q$.
- Consider $\mathsf{Output}(\mathsf{id}, \mathsf{ek}_b, t_b^y)$ for $b \in [2]$, we have $\boldsymbol{\tau}_1 + \boldsymbol{\tau}_2 \equiv (\mathsf{DDec}(t_1^y, c^\alpha), \mathsf{DDec}(t_1^y, c^{\alpha\cdot\hat{s}})) - \mathsf{PRF}(K, \mathsf{id}) + (\mathsf{DDec}(t_2^y, c^\alpha), \mathsf{DDec}(t_2^y, c^{\alpha\cdot\hat{s}})) + \mathsf{PRF}(K, \mathsf{id}) \equiv (\mathsf{DDec}(y\cdot s, c^{x'}), \mathsf{DDec}(y\cdot s, c^{x'\cdot\hat{s}})) \equiv (y\cdot\alpha, (y\cdot\alpha)\cdot\hat{s}) \pmod q$.

From above all, for the output (t_b^y, τ_b) of server $\mathcal{S}_b$, it holds $\tau = \tau_1 + \tau_2 = \alpha\cdot(t_1^y + t_2^y) = \alpha\cdot y$. As the equality $\tau = \alpha\cdot y$ is always satisfied, the verification algorithm will output $y = f(x_1, \ldots, x_n)$.

2) PRIVACY PRESERVATION

The property of privacy preservation requires that each server can not learn any information about the users' data. In our scheme, two non-communicating servers perform evaluations based on the ciphertext of the data, the classifier in polynomial form and the evaluation key, respectively. Like related works [4], [6], [7], [16], [20], [21], [31], [33], we assume that the two servers are not colluding. This assumption is reasonable because the two servers do not need to interact at all, and each server performs evaluations independently.

In more detail, the ciphertext C^x held by each server is obtained by encrypting x with the OKDM algorithm in PKE-NLD. C^x is actually an encryption of $x\cdot\mathsf{sk} = (x, x\cdot\hat{s})$. Boyle et al. [7] have proved that any PPT adversary cannot obtain any information about x and sk from C^x.

Besides C^x, the server $\mathcal{S}_b$ also holds the public key pk, the evaluation key $\mathsf{ek}_b = (s_b, C^\alpha, K)$, and the polynomial function f. Obviously, the information of x and sk cannot be obtained from $\mathsf{pk}, C^\alpha, K, f$. s_b is an additive share of sk that is randomly distributed over R_q^2. Therefore, no individual server can use s_b to get sk. In addition, the server $\mathcal{S}_b$ holding s_b cannot execute the decryption algorithm on C^x, but can only execute the distributed decryption algorithm DDec. The output of DDec is an additive share of $x\cdot\mathsf{sk}$, which is a random value on R_q^2. Therefore, each server cannot obtain information about x and sk through s_b.

We can conclude that each server cannot get any information about the data.

3) VERIFICATION

Verification requires that a malicious server has no ability to convince the company to accept a wrong result $y' \neq y$. We consider two servers not colluding.

In our scheme, without knowing α, a malicious server cannot persuade the company to accept $y' \neq y$, except with a negligible probability. We explain this fact by taking the example that the server $\mathcal{S}_2$ is malicious. Denote $t_2^{y'}$ and τ_2' as the modified result of the server $\mathcal{S}_2$. Let $\Delta_y = y - y' = (t_1^y + t_2^y) - (t_1^y + t_2^{y'}) = t_2^y - t_2^{y'}$, $\Delta_\tau = \tau - \tau' = (\tau_1 + \tau_2) - (\tau_1 + \tau_2') = \tau_2 - \tau_2'$. Server $\mathcal{S}_2$ succeeds iff $\Delta_y \neq 0$ and $\tau_1 + \tau_2' = \alpha(t_1^y + t_2^{y'})$, i.e. $\alpha\Delta_y = \Delta_\tau$. Without knowing α, the probability that $\mathcal{S}_2$ can fool the company by choosing Δ_y and Δ_τ is negligible.

Next, we analyze that a malicious server will not obtain information about α. In our scheme, a malicious server will obtain the ciphertext C^α of $\alpha\cdot s$. But the security of the OKDM algorithm guarantees that the malicious server cannot obtain information about α from the ciphertext C^α. Even if the server tries to decrypt C^α with s_b it will only get an additive share of α, which is a random value and does not provide information about α. Therefore, a malicious server can not get α. Furthermore, if the malicious server wants to convince the company to accept the wrong result, it needs to guess α. Because $\alpha \in R$ and $\|a\|_\infty \leq 1$, it has 2^N ($N > \lambda$, see Table 1) possible values. That is to say, the probability that a malicious server guesses α is $1/2^N < 1/2^\lambda \leq \mathsf{negl}(\lambda)$.

V. PERFORMANCE EVALUATION

In this section, we first benchmark each algorithm in the following PKE-NLD based schemes: the non-verifiable scheme of BKS [7], the verifiable scheme of CZ [16] and our scheme. Next, we compare the performance of these schemes for machine learning classifiers in polynomial form.

Platforms: The trusted institution is realized by a virtual machine with 4GB RAM and an Intel Core i7-6700 CPU. The company is realized by a virtual machine with 4GB RAM and an Intel Core i7-7700K CPU. The users are realized

TABLE 1. The parameters and security level of the PKE-NLD based scheme.

B_{msg}	N	$\log p$	$\log q$	Security
2	8192	63	148	206.6
2^{16}	8192	77	176	168.0
2^{32}	8192	93	208	138.5
2^{64}	16384	126	275	221.3
2^{128}	16384	190	403	142.1
2^{256}	32768	319	662	177.3
2^{512}	65536	576	1177	201.9

TABLE 2. The performance of Gen, Enc, Dec, Ver algorithms in BKS [7], CZ [16] and our scheme.

B_{msg}	Gen (s)			Enc (s)			Dec (ms)	Ver (ms)	
	BKS	CZ	Our	BKS	CZ	Our	BKS	CZ	Our
2	0.18	0.25	0.49	0.27	0.27	0.27	1.7	3.9	3.9
2^{16}	0.21	0.29	0.51	0.30	0.30	0.30	1.8	8.1	8.0
2^{32}	0.22	0.34	0.57	0.34	0.34	0.35	1.8	8.3	8.2
2^{64}	0.56	0.80	1.48	0.88	0.88	0.87	4.2	17.5	17.5
2^{128}	0.75	1.09	1.96	1.16	1.17	1.16	4.3	18.7	18.8
2^{256}	2.53	3.60	6.492	4.09	4.22	4.11	10.8	42.9	43.2
2^{512}	9.58	13.08	23.82	14.32	14.58	14.42	30.7	123.7	123.8

by a virtual machine with 4GB RAM and an Intel Core i5-8500 CPU. Each server is realized by a virtual machine with 32GB RAM and an Intel Xeon E-2286G CPU. All virtual machines in our experiment run Ubuntu Desktop operating system (Ver. 20.04 LTS) and are configured with 100Mbps upload and 80Mbps download network bandwidth settings. We limit the frequency of the CPU of the trusted institution, the company and the users to 800MHZ to simulate scenarios with limited computing resources.

Tools: We use the C++ library NTL 11.5.1 [34] to realize large number operations. Our experiments are all single-threaded. We repeat ten times for different experiments and take the mean of the timings.

A. BENCHMARKS

This section tests the running time of the key generation algorithm, encryption algorithm, decryption/verification algorithm, and each subroutine in evaluation algorithm.

1) EXPERIMENT SETTINGS

To test the benchmark running time of BKS, CZ and our scheme at $B_{\mathsf{msg}} = 2, 2^{16}, 2^{32}, 2^{64}, 2^{128}, 2^{256}, 2^{512}$, we first select the parameters B_{msg}, N, p and q (Table 1) for PKE-NLD according to [7] such that these schemes achieve at least 128-bit security. Next, we implement these schemes with the parameters of Table 1 and show their performance in Tables 2 and 3.

In Table 2, the Gen of our scheme takes more time than BKS and CZ. Fortunately, the Gen is executed only once, and the running time of the Gen of our scheme is at most 2.7× that of BKS and CZ. The running time of Enc in all schemes are close to each other for the same B_{msg}. Regarding Dec/Ver, it takes more time in our scheme and CZ to verify the results than BKS. Our scheme and CZ use the same method to verify the results. Therefore, the running time of their Ver is the same.

TABLE 3. The performance of instructions in Eval in BKS [7], CZ [16] and our scheme.

B_{msg}	Convert (ms)		Add (ms)	
	BKS, Our	CZ	BKS, Our	CZ
2	49.8	101.2	1.1	2.3
2^{16}	54.0	109.8	1.1	2.2
2^{32}	60.9	123.9	1.1	2.4
2^{64}	153.3	309.7	2.5	5.2
2^{128}	204.6	411.3	2.6	5.3
2^{256}	712.4	1440.8	6.6	13.7
2^{512}	2880.4	5781.7	20.8	42.3

B_{msg}	sMult (ms)		Mult (ms)		Output (ms)		
	Our	CZ	BKS, Our	CZ	BKS	CZ	Our
2	1.2	100.1	49.8	99.9	0.8	1.7	49.9
2^{16}	2.2	115.3	54.0	109.8	0.8	1.8	54.3
2^{32}	2.3	124.7	61.0	127.2	0.9	1.8	61.1
2^{64}	4.9	311.9	152.0	308.3	1.7	3.4	150.0
2^{128}	5.3	408.7	205.0	411.5	1.7	3.4	205.9
2^{256}	13.8	1329.3	712.1	1414.2	3.7	6.7	714.7
2^{512}	43.6	5778.2	2779.2	5378.4	8.5	16.3	2719.8

Table 3 shows that Convert, Add, and Mult in CZ [16] take twice more time as that in BKS [7] and our scheme. This is because CZ needs to compute a tag for each of these instructions to enable verification, and the computation of the tag takes as much time as the corresponding instruction. BKS does not support scalar multiplications (sMult). Although CZ supports sMult, their algorithm is obtained by modifying Mult and as slow as Mult. On the contrary, sMult of our scheme is not based on Mult and is 50-100× faster. The Output of our scheme is the slowest. However, Output can be performed only 1 time for each polynomial function f and has a very limited impact on efficiency.

B. EVALUATION OF CLASSIFIERS IN POLYNOMIAL FORM

In this section, we demonstrate the efficiency of BKS, CZ and our scheme with decision trees, a fundamental and widely used machine learning classifier.

1) EXPERIMENT SETTINGS

We use the real and sensitive data set: activity recognition with healthy older people using a batteryless wearable sensor data set [18]. We use the Python library scikit-learn [32] to train the decision tree. We trained decision tree models with depth $d = 2, 4, \ldots, 14$ respectively on the data sets. We use the technique of Bost et al. [9] to convert a depths d decision tree into a degree d polynomial. Raw data is encoded as a 32-bit integer to apply BKS, CZ and our scheme. We choose $B_{\mathsf{msg}} = 2^{64}, B_{\mathsf{msg}} = 2^{128}, B_{\mathsf{msg}} = 2^{256}, B_{\mathsf{msg}} = 2^{512}$ such that BKS, CZ and our scheme can evaluate the polynomials of degree $d = 2$, $d = 4$, $d = 6, 8$, $d = 10, 12, 14$ for 32-bit inputs, respectively.

According to the settings described above, we implemented BKS, CZ, and our proposed scheme. We compared the computational overhead, ciphertext size, and partial result size of each scheme as follows.

2) COMPUTATION OVERHEAD

We denote by t_s^1, t_s^2, and t_s^3 the time taken by each server in BKS, CZ and our scheme, respectively. Figure 4a shows

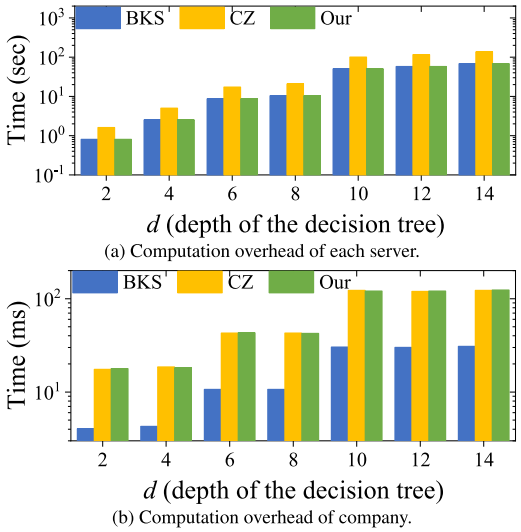

FIGURE 4. The computation overhead of BKS [7], CZ [16] and our scheme.

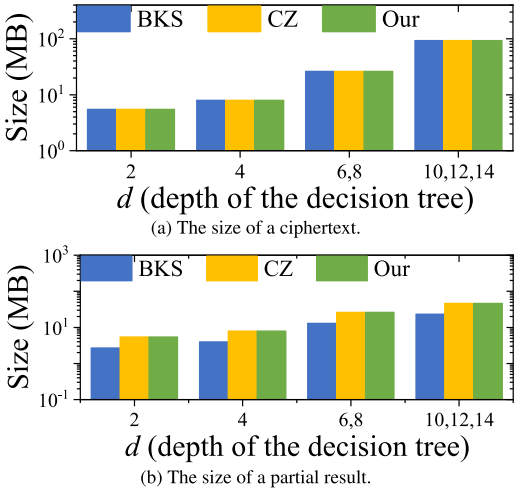

FIGURE 5. The size of a ciphertext, a partial result of BKS [7], CZ [16] and our scheme.

$(t_s^1, t_s^2)/t_s^3 \approx (1, 2)$. CZ is constructed on the basis of BKS. In order to achieve verifiability, CZ needs to compute a tag for instructions in Eval, and the computation of the tag is the same as the corresponding instruction. This results in about 2 times the computation of CZ on the server side than BKS. Our scheme does not need to compute a tag. Although the computation of Output of our scheme is twice that of Output of BKS, Output is only executed once in our scheme. Therefore, our scheme computation on the server-side is almost equal to that of BKS. We denote by t_c^1, t_c^2 and t_c^3 taken by each company in the three schemes, respectively. Figure 4b shows the fact: t_c^1 is the smallest, and t_c^2 and t_c^3 are almost equal. For BKS, the computations of the reconstruction of the classification result is dominated of 1 addition over R_q. For CZ and our scheme, the computations of the reconstruction and verification of the classification result are dominated by 2 additions and 1 multiplication over R_q, meaning that both CZ and our scheme require the same amount of computation for reconstructing the correct result.

3) CIPHERTEXT SIZE AND PARTIAL RESULT SIZE

The ciphertext in the three schemes is an element of $(R_q^2)^2$. Therefore, BKS, CZ and our scheme have ciphertexts of the same size. The partial results generated by every server in BKS and CZ/our scheme are uniformly distributed over R_q and R_q^2, respectively. Therefore, CZ and our scheme have the same size partial results, and and twice as large as BKS. Figure 4 reflects these facts.

VI. DISCUSSION

Currently, there exist five works [16], [17], [36], [37], [38] that concentrate on VHSS schemes. However, not all of these schemes are secure or efficient. The scheme of [36] is insecure. The schemes of [37] and [38] only support linear polynomial computations. The scheme of [17] requires a non-constant number of servers. The scheme of [16] perform worse than HSS scheme of BKS [7]. Compared with these works, our scheme uses *two servers* that can support computing polynomials whose degree can be a *polynomial of security parameters*, and the performance is *comparable to HSS* of BKS.

FHE, DP, MSS can only protect the privacy of data, while Homsig, Homomorphic MACs can only verify the correctness of the results. Our VHSS scheme can protect the privacy of data and the correctness of classification results at the same time. Some VC schemes can also solve these two problems, but these schemes either rely on the less efficient FHE, or only support computing matrix-vector multiplication, or require complex information interaction between servers. However, our *non-interactive* VHSS scheme can compute polynomials whose degree can be a *polynomial of security parameters*, and *does not depend on FHE*.

VII. CONCLUSION

Previous VHSS schemes either only supported computing linear polynomials, required a non-constant number of servers, or performed worse than HSS scheme. These shortcomings make VHSS unsuitable for data classification using polynomial form machine learning classifiers in cloud computing. To solve this problem, this paper constructs a new VHSS scheme. Our scheme uses *two servers* and PKE-NLD that can support computing polynomials whose degree can be a *polynomial of security parameters*. Experiments show that the server-side performance of our scheme is *comparable to HSS* and 2× faster than the existing VHSS scheme CZ.

In our scheme, the machine learning classifier is transformed into a polynomial, and the polynomial is sent to the servers in plaintext. Machine learning classifiers are trade secrets to a company. Once the model of the classifier is leaked, the company cannot profit from it. In our scheme, the company can encrypt the coefficients of the polynomial and send it to the servers to protect the privacy of the model, but this will also increase the amount of computations on the servers. How to preserve the privacy of the classifier while maintaining the performance is an interesting future work.

One significant drawback of VHSS schemes is that it necessitates either modifying applications or using dedicated and specialized client-server applications to ensure its functional operation. Additionally, the methodology does not allow one to conduct ad-hoc/discovery-based queries. As future work, designing a VHSS scheme that supports discovery-based queries would be an interesting challenge.

REFERENCES

[1] P. Ananth, N. Chandran, V. Goyal, B. Kanukurthi, and R. Ostrovsky, "Achieving privacy in verifiable computation with multiple servers—Without FHE and without pre-processing," in *Proc. Int. Conf. Public-Key Cryptogr.*, 2014, pp. 149–166.

[2] B. Applebaum, Y. Ishai, and E. Kushilevitz, "From secrecy to soundness: Efficient verification via secure computation," in *Proc. Int. Colloq. Automat. Lang. Program.*, 2010, pp. 152–163.

[3] B. Balle, J. Bell, A. Gascón, and K. Nissim, "Private summation in the multi-message shuffle model," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Oct. 2020, pp. 657–676.

[4] E. Boyle, G. Couteau, N. Gilboa, Y. Ishai, and M. Orrù, "Homomorphic secret sharing: Optimizations and applications," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Oct. 2017, pp. 2105–2122.

[5] M. Barbosa and P. Farshim, "Delegatable homomorphic encryption with applications to secure outsourcing of computation," in *Proc. Conf. CT-RSA*, 2012, pp. 296–312.

[6] E. Boyle, N. Gilboa, and Y. Ishai, "Group-based secure computation: Optimizing rounds, communication, and computation," in *Proc. Annu. Int. Conf. Theory Appl. Cryptol. Techn.*, 2017, pp. 163–193.

[7] E. Boyle, L. Kohl, and P. Scholl, "Homomorphic secret sharing from lattices without FHE," in *Proc. EUROCRYPT*, 2019, pp. 3–33.

[8] O. Barkol, Y. Ishai, and E. Weinreb, "On d-multiplicative secret sharing," *J. Cryptol.*, vol. 23, no. 4, pp. 580–593, Oct. 2010.

[9] R. Bost, R. A. Popa, S. Tu, and S. Goldwasser, "Machine learning classification over encrypted data," in *Proc. Netw. Distrib. Syst. Secur. Symp.*, 2015.

[10] K. Chatzikokolakis, M. E. Andrés, N. E. Bordenabe, and C. Palamidessi, "Broadening the scope of differential privacy using metrics," in *Proc. Int. Symp. Privacy Enhancing Technol. Symp.*, 2013, pp. 82–102.

[11] D. Catalano and D. Fiore, "Practical homomorphic message authenticators for arithmetic circuits," *J. Cryptol.*, vol. 31, no. 1, pp. 23–59, Jan. 2018.

[12] D. Catalano, D. Fiore, and B. Warinschi, "Homomorphic signatures with efficient verification for polynomial functions," in *Proc. 34th Annu. Cryptol. Conf.*, 2014, pp. 371–389.

[13] K. M. Chung, Y. T. Kalai, and S. P. Vadhan, "Improved delegation of computation using fully homomorphic encryption," in *Proc. CRYPTO*, 2010, pp. 483–501.

[14] Y. Cheng, J. Ma, Z. Liu, Y. Wu, K. Wei, and C. Dong, "A lightweight privacy preservation scheme with efficient reputation management for mobile crowdsensing in vehicular networks," *IEEE Trans. Depend. Sec. Comput.*, early access, Mar. 31, 2022, doi: 10.1109/TDSC.2022.3163752.

[15] R. Canetti, B. Riva, and G. N. Rothblum, "Practical delegation of computation using multiple servers," in *Proc. ACM Conf. CCS*, Oct. 2011, pp. 445–454.

[16] X. Chen and L. F. Zhang, "Two-server verifiable homomorphic secret sharing for high-degree polynomials," in *Proc. Int. Conf. Inf. Secur.*, 2020, pp. 75–91.

[17] X. Chen, L. F. Zhang, and J. Liu, "Verifiable homomorphic secret sharing for low degree polynomials," *IEEE Trans. Depend. Sec. Comput.*, early access, Jul. 27, 2022, doi: 10.1109/TDSC.2022.3194321.

[18] D. Dua and C. Graff. (2017). *UCI Machine Learning Repository*. [Online]. Available: http://archive.ics.uci.edu/ml

[19] J. C. Duchi, M. I. Jordan, and M. J. Wainwright, "Local privacy and statistical minimax rates," in *Proc. IEEE 54th Annu. Symp. Found. Comput. Sci.*, Oct. 2013, pp. 429–438.

[20] I. Dinur, N. Keller, and O. Klein, "An optimal distributed discrete log protocol with applications to homomorphic secret sharing," *J. Cryptol.*, vol. 33, no. 3, pp. 824–873, Jul. 2020.

[21] N. Fazio, R. Gennaro, T. Jafarikhah, and W. E. Skeith, "Homomorphic secret sharing from Paillier encryption," in *Proc. Int. Conf. Provable Secur.*, 2017, pp. 381–399.

[22] D. Fiore, A. Mitrokotsa, L. Nizzardo, and E. Pagnin, "Multi-key homomorphic authenticators," *IET Inf. Secur.*, vol. 13, no. 6, pp. 618–638, Nov. 2019.

[23] C. Gentry, "Fully homomorphic encryption using ideal lattices," in *Proc. 41st Annu. ACM Symp. Theory Comput.*, May 2009, pp. 169–178.

[24] J. Guo, X. Li, Z. Liu, J. Ma, C. Yang, J. Zhang, and D. Wu, "TROVE: A context-awareness trust model for VANETs using reinforcement learning," *IEEE Internet Things J.*, vol. 7, no. 7, pp. 6647–6662, Jul. 2020.

[25] R. Gennaro and D. Wichs, "Fully homomorphic message authenticators," in *Proc. Int. Conf. Theory Appl. Cryptol. Inf. Sec.*, 2013, pp. 301–320.

[26] Y. He and L. F. Zhang, "Cheater-identifiable homomorphic secret sharing for outsourcing computations," *J. Ambient Intell. Hum. Comput.*, vol. 11, no. 11, pp. 5103–5113, Nov. 2020.

[27] R. Johnson, D. Molnar, D. Song, and D. Wagner, "Homomorphic signature schemes," in *Topics in Cryptology—CT-RSA 2002*. 2002, pp. 244–262.

[28] Z. Liu, J. Ma, J. Weng, F. Huang, Y. Wu, L. Wei, and Y. Li, "LPPTE: A lightweight privacy-preserving trust evaluation scheme for facilitating distributed data fusion in cooperative vehicular safety applications," *Inf. Fusion*, vol. 73, pp. 144–156, Sep. 2021.

[29] V. Lyubashevsky, C. Peikert, and O. Regev, "On ideal lattices and learning with errors over rings," in *Proc. Annu. Int. Conf. Theory Appl. Cryptograph. Techn.*, 2010, pp. 1–23.

[30] P. Martins, L. Sousa, and A. Mariano, "A survey on fully homomorphic encryption: An engineering perspective," *ACM Comput. Surv.*, vol. 50, no. 6, pp. 1176–1185, 2018.

[31] C. Orlandi, P. Scholl, and S. Yakoubov, "The rise of Paillier: Homomorphic secret sharing and public-key silent OT," in *Proc. EUROCRYPT*. 2021, pp. 678–708.

[32] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and É. Duchesnay, "Scikit-learn: Machine learning in Python," *J. Mach. Learn. Res.*, vol. 12 no. 10, pp. 2825–2830, 2012.

[33] L. Roy and J. Singh, "Large message homomorphic secret sharing from DCR and applications," *IACR Cryptol. ePrint Arch.*, 2021.

[34] V. Shoup. (Mar. 2020). *NTL: A Library for Doing Number Theory*. [Online]. Available: https://www.shoup.net/ntl/

[35] X. Sun, P. Zhang, J. K. Liu, J. Yu, and W. Xie, "Private machine learning classification based on fully homomorphic encryption," *IEEE Trans. Emerg. Topics Comput.*, vol. 8, no. 2, pp. 352–364, Apr./Jun. 2020.

[36] G. Tsaloli, B. Liang, and A. Mitrokotsa, "Verifiable homomorphic secret sharing," in *Proc. Int. Conf. Provable Secur.*, 2018, pp. 40–55.

[37] G. Tsaloli, G. Banegas, and A. Mitrokotsa, "Practical and provably secure distributed aggregation: Verifiable additive homomorphic secret sharing," *Cryptography*, vol. 4, no. 3, p. 25, Sep. 2020.

[38] G. Tsaloli and A. Mitrokotsa, "Sum it up: Verifiable additive homomorphic secret sharing," in *Proc. Int. Conf. Inf. Secur. Cryptol.*, 2019, pp. 115–132.

[39] H. C. Tanuwidjaja, R. Choi, S. Baek, and K. Kim, "Privacy-preserving deep learning on machine learning as a service—A comprehensive survey," *IEEE Access*, vol. 8, pp. 167425–167447, 2020.

[40] P. Wang, X. Chen, F. Ye, and Z. Sun, "A survey of techniques for mobile service encrypted traffic classification using deep learning," *IEEE Access*, vol. 7, pp. 54024–54033, 2019.

[41] A. Wood, K. Najarian, and D. Kahrobaei, "Homomorphic encryption for machine learning in medicine and bioinformatics," *ACM Comput. Surv.*, vol. 53, no. 4, pp. 1–35, 2020.

[42] D. Zissis and D. Lekkas, "Addressing cloud computing security issues," *Future Gener. Comput. Syst.*, vol. 28, no. 3, pp. 583–592, Mar. 2012.

[43] L. F. Zhang and R. Safavi-Naini, "Protecting data privacy in publicly verifiable delegation of matrix and polynomial functions," *Des. Codes Cryptogr.*, vol. 88, no. 4, pp. 677–709, Apr. 2020.

XIN CHEN received the bachelor's degree in applied mathematics from Xinjiang University. He is working pursuing the Ph.D. degree with the School of Information Science and Technology, ShanghaiTech University. His main research interests include homomorphic encryption, outsourced computing, and verifiable computation.