IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received March 23, 2022, accepted April 6, 2022, date of publication April 11, 2022, date of current version April 18, 2022.

Digital Object Identifier 10.1109/ACCESS.2022.3166523

Secure and Evaluable Clustering Based on a Multifunctional and Privacy-Preserving Outsourcing Computation Toolkit

JIALIN LI, PENGHAO LU, AND XUEMIN LIN, (Fellow, IEEE)
School of Software Engineering, East China Normal University, Shanghai 200062, China

Corresponding author: Jialin Li (katleesh@outlook.com)

This work was supported in part by the National Natural Science Foundation of China under Grant 62172161, Grant 62132005, and Grant 62172162, and in part by the Natural Science Foundation of Shanghai under Grant 20ZR1418400.

ABSTRACT Although tremendous revolution has been made in the emerging cloud computing technologies over digital devices, privacy gradually becomes a big concern in outsourcing computation. Homomorphic encryption has been proposed to facilitate the preservation of data privacy while computational tasks being executed on ciphertext. However, many existing studies only support limited homomorphic calculation functions which barely satisfy complex computing tasks such as machine learning with massive computing resources and rich types of function. To address this problem, a novel multifunctional and privacy-preserving outsourcing computation toolkit is proposed in this paper, which supports several homomorphic computing protocols including division and power on ciphertext of integers and floating point numbers. Specifically, we first implement the homomorphic mutual conversion protocol between integer and floating point ciphertext to balance the efficiency and feasibility, considering the high-precision ciphertext operation on floating point numbers costs 100x computational overhead than that on integers. Second, we implement a homomorphic K-means algorithm based on our proposed toolkit for clustering and design the homomorphic silhouette coefficient as the evaluation index, thereby providing an informative cluster assessment for local users with limited resources. Then, we simulate the protocols of our proposed toolkit to explore the parameter sensitivity in terms of computational efficiency. Last, we report security analysis to prove the security of our toolkit without privacy leakage to unauthorized parties. Comprehensive experiments further demonstrate the efficiency and utility of our toolkit.

INDEX TERMS Privacy-preserving, secure outsourcing computation, K-means, silhouette coefficient.

I. INTRODUCTION

Cloud computing is a third party provider delivering on-demand services of data storage, computing resources and running functions without direct management by local users. It exceeds local servers in massive storage, unlimited computational resources and increasing availability. Therefore, individuals and companies with limited resources resort to the cloud services by outsourcing their computational tasks to the cloud. Meanwhile, sensitive information exists in countless domains, such as medical health monitoring [1], [2], financial transaction records and location services [3]. Though powerful and flexible, the misgiving of privacy preservation on the cloud remains as a prior concern, which may be deteriorating the functionality and performance of the cloud services. For example, the personal information of over 77 million of accounts in Sony PlayStation Network outage were stolen due to an illegal and unauthorized intrusion, which damaged the reputation of the enterprise and caused huge economic loss. The breach of photographs of the celebrity from iCloud service in Apple company raised widespread condemnation from the public which forced the company to improve the data protection policy as well as technologies. As a consequence, the security issues exposed in such incidents have aroused an upsurge of research in both industry and academia.

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

A. MOTIVATIONS

In general, the privacy preservation approaches rely on the following assumptions: the cloud servers are honest-but-curious, and they do not act maliciously against the users and will follow the protocols as expected. However, the cloud server may not always be trusted. On the one hand, they might be vulnerable to the attackers who intend to gain access

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/

to the sensitive information from the outsourced data on the cloud. On the other hand, some intelligent cloud servers can be curious and sometimes even malicious who may deduce the information of the original data through the model and intermediate calculation results [4]. Therefore, how to design effective and suitable privacy-preserving protocols to achieve secure outsourcing computation on the cloud becomes an opening challenge.

In recent years, machine learning technology has been widely applied in natural language processing, computer vision, intrusion detection and etc. Consider a clustering task requested by a local user who uploaded the resource-demanding task to the cloud. The privacy concerns of this user are in threefold. First, the private training data has potential risk of leakage through the deduction from model parameters or predictions. Second, existing machine learning algorithms may involve complex operations such as division, power, exponential and logarithmic, which few secure outsourcing computation protocols are able to support. Previous frameworks [5], [6] can only support homomorphic addition, multiplication and comparison on ciphertext of floating point numbers(FPN) which cannot meet the requirement of clustering. Third, the secure outsourcing computation protocols on integer ciphertext are not capable of ensuring the convergence and precision of these machine learning algorithms. If running the entire task on the ciphertext of floating-point numbers is a must, it will cause huge computational overhead and time cost since the computation overhead on FPN is hundreds of times higher than that on integers. Furthermore, some of the previous outsourcing clustering algorithms [7]–[9] only implemented the clustering algorithm on ciphertext, and they did not consider how to evaluate the performance of clustering on ciphertext. In practice, the data owners are usually not aware of the number of classes that need to be clustered, which is necessary to be compared through multiple experimental evaluations.

Based on the above problems, we summarize the main contributions of this paper in the next section.

B. MAIN CONTRIBUTIONS

In this paper, we propose a multifunctional and privacy-preserving outsourcing computation toolkit and we have implemented a secure and evaluable homomorphic clustering algorithm. The main contributions of our paper are summarized as follows:

- A multifunctional and privacy-preserving outsourcing computation toolkit (**MPOCT**) is proposed in this paper which provides several protocols both on ciphertext of integers and floating point numbers(FPNs). Specifically, our protocols achieve homomorphic operations such as division, power, logarithm, etc.
- In **MPOCT**, mutual ciphertext conversion protocols between FPNs and integers are provided to balance calculation efficiency and precision. During outsourcing computations, when encountering high-precision calculation tasks, we can transform the integer ciphertext to FPN ciphertext for calculation. If tasks do not have high requirements for precision, we can use integer ciphertext for calculations to improve calculation efficiency and reduce computational overhead.
- Based on **MPOCT**, we implement the homomorphic algorithm of K-means. Furthermore, the homomorphic silhouette coefficient is designed on the ciphertext of clustering results, providing the informative clustering assessment to the local data owners.

The rest of this paper is organized as follows. We discuss the related works in Section II. Then we introduce the preliminaries of this paper in Section III. Our toolkit is described in Section IV and its application on clustering is introduced in Section V. We analyze the security of our toolkit in Section VI.The experimental evaluation is reported in Scetion VII. Section VIII concludes the paper and outlines the future work.

II. RELATED WORK

In this section, we briefly review the most relevant research on homomorphic encryption, secure outsourcing computation frameworks and previous secure outsourcing computation solutions for K-means.

Cloud computing drives the transition of local on-demanding tasks to the cloud servers for storage and calculation [10]. In recent years, privacy-preserving in cloud security has gradually become an increasing important concern [11]. The emergence of homomorphic encryption allows people to perform simple calculations on the ciphertext which is a powerful cryptographic primitive to secure outsourcing computations against an untrusted third-party provider. For example, Paillier cryptosystem [12] which is widely used in industry supports additive homomorphism, while Elgamal [13] supports multiplicative homomorphism. However, partially homomorphic cryptosystem only supports the homomorphic operation of addition or multiplication, so the application scenarios are very limited. Gentry [14] first proposed a fully homomorphic encryption (FHE) based on ideal lattice from a theoretical perspective in 2009. However, FHE cannot be applied in real world due to its large computational overhead. To address with the efficiency problem of FHE, many researchers have proposed more efficient FHE schemes like BGV [15], BFV [16], CKKS [17]. Based on the above-mentioned cryptographic schemes, many scholars have also proposed secure outsourcing computation frameworks. In 2016, Liu not only designed operations such as homomorphic multiplication and comparison through dual servers and Paillier cryptosystem, but also proposed secure outsourcing computation frameworks that support rational numbers [6] and floating-point numbers [5]. Liu *et al.* [18] implemented a secure k-nearest neighbor based multi-label classification scheme based on Paillier cryptosystem, which both guarantees the security of the training data and classification model. Liu *et al.* [19] proposed a secure outsourcing computation framework based on FHE and key packaging technology, and this framework only required a single server

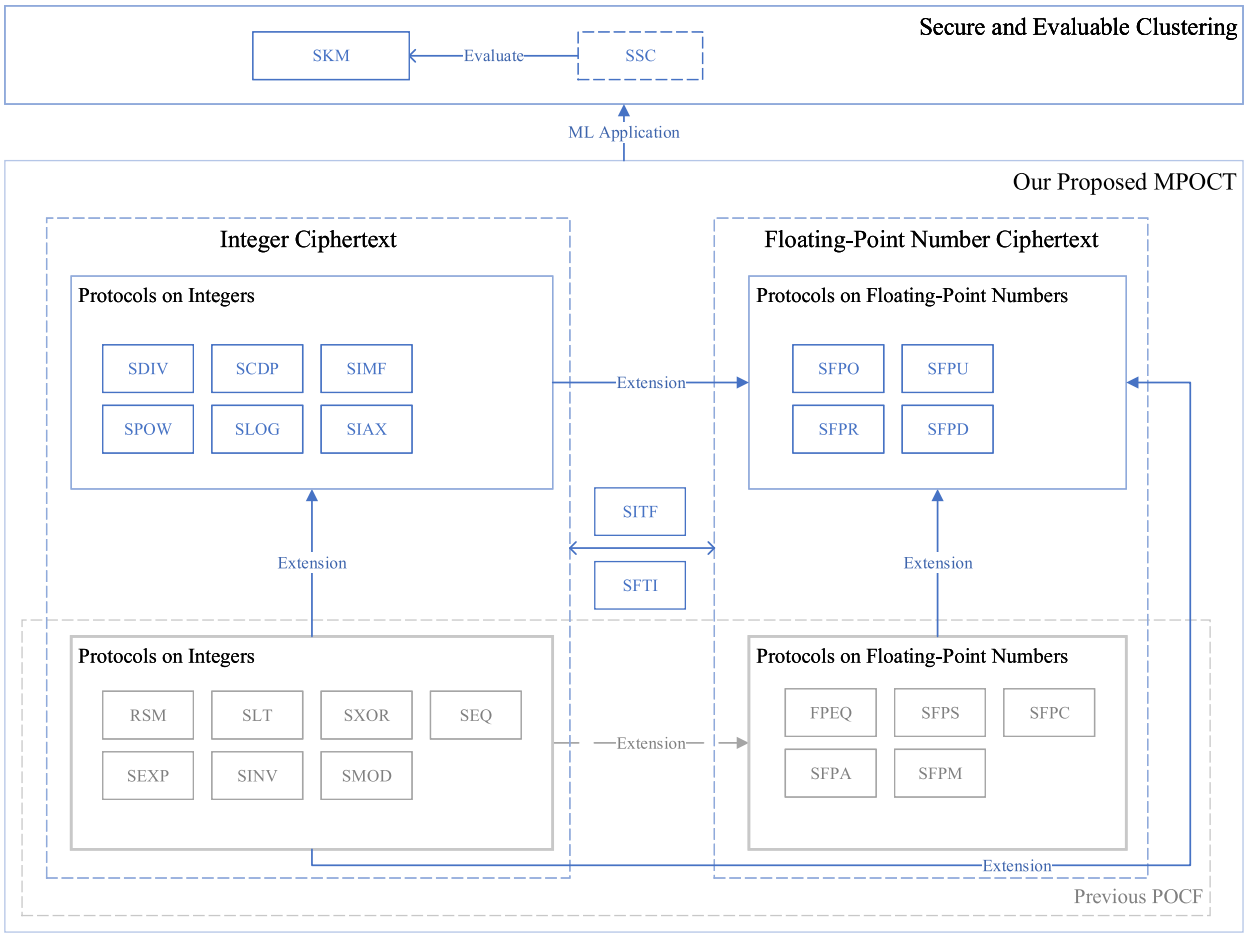


FIGURE 1. A Call Graph of MPOCT. Blue: our implemented protocols. Grey: previous protocols in [5].

TABLE 1. Comparison between different solutions under untrusted server scenario.

Characteristic	[7]	[9]	[8]	[22]	[23]	[24]	Ours
Without Trusted 3rd Party	✓	×	×	✓	✓	✓	✓
Only One-round Interaction	✓	✓	×	×	×	✓	✓
Precision Not Reduced	✓	✓	✓	×	✓	✓	✓
Supporting Division	×	×	✓	✓	✓	✓	✓
Supporting FPNs	×	×	×	✓	✓	✓	✓
Supporting Evaluation	×	×	×	✓	×	×	✓

to perform oursourced computing tasks. Li *et al.* [20] proposed a novel homomorphic encryption framework over non-abelian rings and defined the homomorphism operations in ciphertexts space which support real numbers encryption and privacy-preserving for machine learning training and classification in data ciphertexts environment. Zong *et al.* [21] investigated secure outsourcing computation of matrix determinant based on BGV and proposed an efficient matrix encoding technique which packs a matrix into a single ciphertext and can be easily applied as a submodule in the high-level applications. Nevertheless, above works more or less suffered from low efficiency, accuracy degradation, lack of scalability or ciphertext expansion issues.

As a common data mining algorithm, K-means [25] has abundant application scenarios in daily lives. Therefore, many scholars have studied how to perform clustering on cloud servers under the premise of protecting data privacy. Rao *et al.* [7] implemented the outsourcing computation of K-means based on Paillier in 2015. Then, Rong *et al.* [9] improved the computing efficiency by adding a trusted third party with limited computing capabilities. Almutairi *et al.* [8] also implemented the outsourcing computation of K-means based on homomorphic encryption. This solution has greatly improved the computing efficiency through a third party and limited interaction between user and server. However, neither of the above solutions provided evaluation indicators on clustering ciphertext. Therefore, the user can only get the clustering result, but cannot get the evaluation of them. Besides, the emergence of differential privacy [26] opened up new areas of secure outsourcing computation. Xia *et al.* [22] could efficiently complete outsourcing clustering tasks through local differential privacy technology. However, the choice of privacy budget had a great impact on the results, which means a better privacy may lead to a poorer precision. At the same time, the frequent interaction between user and server also puts a lot of pressure on the user side. Mohassel *et al.* [23]

presented a scalable privacy-preserving clustering algorithm and design a modular approach for multi-party clustering. Although this scheme is computationally efficient, it relies on frequent collaboration and interaction between sender and receiver. Zou *et al.* [24] proposed a highly secure privacy-preserving outsourced k-means clustering under multiple keys in cloud computing, which applied both AES encryption and BCP encryption to provide privacy preservation against semi-honest adversaries. While the scheme guarantees strong security, it does not provide users with an evaluation metric for clustering. Table 1 summarizes the characteristics of different solutions to the aforementioned studies.

III. PRELIMINARIES

In this section, we will introduce the preliminaries of this paper. Our toolkit is mainly based on privacy-preserving outsourcing computation framework (POCF) [5], and we have an expansion and improvement on it. The following will introduce the overview of our system model, problem statement, the basic framework of POCF, and its protocols on integers and floating-point numbers.

A. SYSTEM MODEL

In our system, we mainly focus on how the cloud server can fulfill the computing tasks requested by the users while meeting the requirements of privacy protection. As shown in Figure 2, the simple system comprises a Key Generation Center(KGC), a Cloud Platform(CP), a Computation Service Provider(CSP) and Data Owners(DOs), while it can be extended to a distributed system with multiple pairs of CP and CSP. In this paper, we only use the simple model. The participants in this model are shown below.

- **KGC**: KGC is a trusted third party whose task is to distribute and manage the keys in the system.
- **DOs**: DOs own the data and they are the requesers of the computing tasks. DOs can encrypt their data by the public key(pk). Then they outsourced the encrypted data to CP for storage and requester CP to do some calculations on the outsourced encrypted data. In addition, DOs can use the private key(sk) to decrypt the encrypted result calculated by CP and CSP.
- **CP**: CP has 'unlimited' data storage space. CP can store and manage the outsourced data of all registered data owners. CP has the public key(pk) and the partially private key sk_1. CP always stores the encrypted results of the computational tasks.
- **CSP**: CSP assists CP in completing computing tasks requested by the DOs. CSP owns the public key(pk) and the partially private key sk_2. CSP can partially decrypt the ciphertext sent by CP and perform some operations on the decrypted results. Then CSP encrypts the calculated results and returns them to CP.

B. PROBLEM STATEMENT

Consider DOs have a lot of data and a clustering task, and their limited local resources make it impossible to complete the clustering task. The data therefore needs to be encrypted before being uploaded to the CP for storage. DOs can submit the input parameters required for the clustering task to CP. After CP and CSP cooperate to complete the calculation task, the clustering result and its evaluation are returned to DOs. In this process, we have the following challenges:

- Clustering algorithms and evaluation indicators involve division and comparison operations, but many homomorphic encryption schemes cannot support division homomorphism and comparison homomorphism. Therefore, more calculation protocols need to be constructed.
- Traditional K-means outsourcing computation schemes based on homomorphic encryption always normalize the original data and enlarge it. Then they encrypt the data after rounding. However, simply calculating the magnification of the plaintext is likely to exceed the limit of the plaintext space after a finite number of multiplications.
- Through previous experiments, we have learned that the calculation efficiency of floating-point number ciphertext is low and the communication overhead is high. However, in the process of clustering and evaluation, we may encounter situations where floating-point numbers must be used. If the whole process is based on floating-point ciphertext, it will take a lot of time to complete the clustering task.

C. ATTACK MODEL

In our attack model, DOs, CP and CSP are honest and curious. They will strictly implement the protocols, but they are also interested in data uploaded. We assume that CP and CSP are cooperative and not colluding. Meanwhile we assume that the communication between KGC and all parties are secure. Here, we introduce the adversary $\mathcal{A}^*$ in our model. The purpose of adversary $\mathcal{A}^*$ is to use the following capabilities to decrypt the ciphertext of challenger DO:

1) Adversary $\mathcal{A}^*$ may eavesdrop on all communication channels to obtain encrypted data transmissed.
2) Adversary $\mathcal{A}^*$ may compromise CP, thereby guessing the plaintext corresponding to encrypted data outsourced from DO. In addition, it can perform interactive protocols with CSP to guess the plaintext corresponding to ciphertext sent from CSP.
3) Adversary $\mathcal{A}^*$ may compromise CSP. Therefore, it can perform interactive protocols with CP to guess the plaintext corresponding to ciphertext sent from CP.
4) Adversary $\mathcal{A}^*$ may compromise DOs to obtain their decryption capabilities, with the exception of the challenger DO. Then adversary $\mathcal{A}^*$ may try to guess all the plaintext belonging to challenger DO through the decryption capabilities of other DOs.

However, Adversary $\mathcal{A}^*$ cannot compromise CP and CSP at the same time. Moreover, adversary $\mathcal{A}^*$ is restricted from compromising challenger DO. The above restrictions are common in adversary models in cryptographic protocols [27].

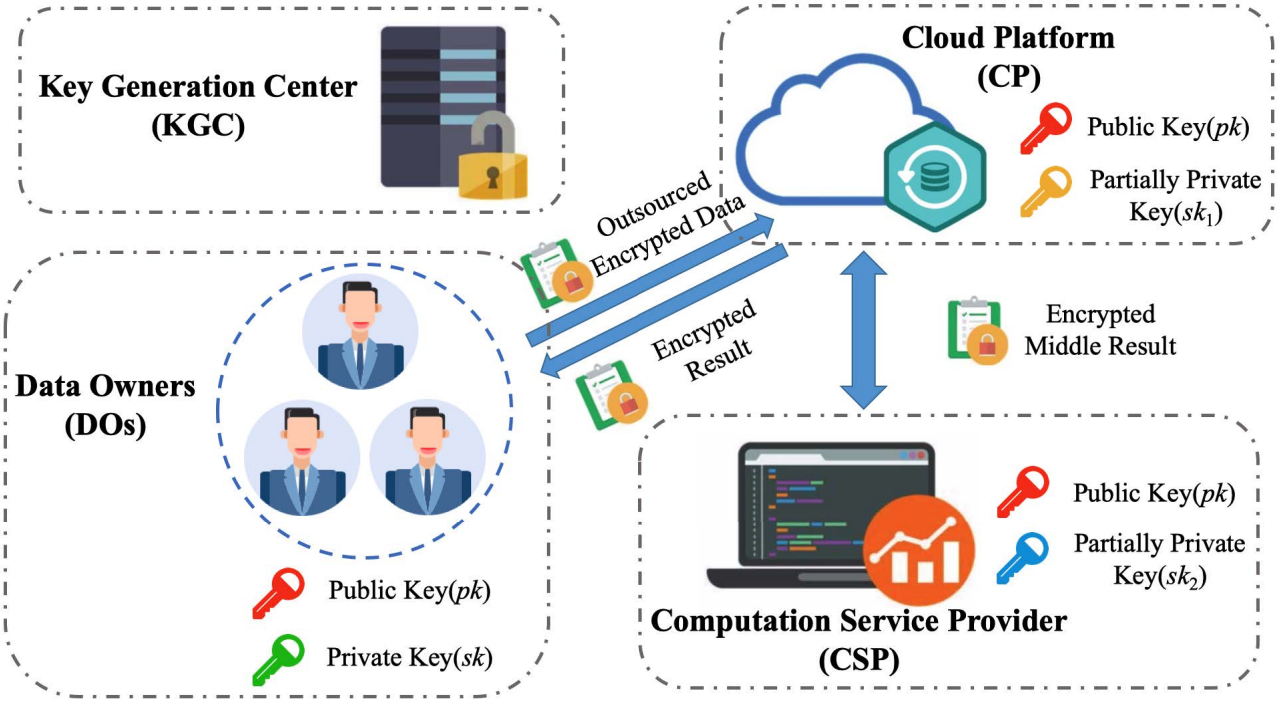


FIGURE 2. The overview of our system model.

D. BASIC FRAMEWORK

The toolkit proposed in this paper is based on the Paillier Cryptosystem with Partial Decryption (**PCPD**) [5]. The basic functions and key splitting algorithm (**KeyS**) of Paillier have been described in **PCPD**, the description is omitted here. In this paper, we specify the following symbols: $[x]$ represents the ciphertext of integer x and $\langle x \rangle$ represents the ciphertext of floating-point number (FPN) x, where $\langle x \rangle = ([s_x], [m_x], [t_x])$. s_x represents the sign of x, m_x represents η-significant digits, t_x represents the exponent of base β. For the sake of convenience, we will use $\eta = 16$ and $\beta = 10$ for protocols on FPNs in this paper. Given $[x]$, $[y]$ and public key N, we have the following properties:

$[x + y] = [x] \cdot [y]$; $[-x] = [x]^{N-1}$; $[x - y] = [x] \cdot [y]^{N-1}$.

E. PREVIOUS WORK ON INTEGERS & FLOATING-POINT NUMBERS

Before presenting our toolkit, we briefly review the previous study [5] of secure outsourcing computational protocols on integers and floating-point numbers respectively. Besides, the construction of floating point number ciphertext is also introduced. Note that our toolkit is an extension based on this previous work.

For simplicity, here we define '@' as 'performed by'. For example, '@CP' means 'performed by CP'.

1) PROTOCOLS ON INTEGERS

Protocols on integers enable the cloud platform CP to cooperate with the computation service provider CSP to implement multiplication, comparison, xor and other operations of integer plaintext on the ciphertext.

a: REVISED SECURE MULTIPLICATION PROTOCOL (RSM)

It enables CP and CSP to cooperate to perform secure integer plaintext multiplication on ciphertext, i.e. $[f] = \mathbf{RSM}([x], [y])$, where $f = x \cdot y$. Details are as follows:

- **Step-1 (@CP)**: CP randomly selects two random numbers $r_x, r_y \in \mathbb{Z}_N^*$ and compute:
  $X = [x + r_x] = [x] \cdot [r_x]$;
  $Y = [y + r_y] = [y] \cdot [r_y]$;
  $X_1 = PD_{sk^1}(X)$; $Y_1 = PD_{sk^1}(Y)$;
  After computation, CP will send X, Y, X_1 and Y_1 to CSP.
- **Step-2 (@CSP)**: CSP partially decrypts X_1 and Y_1 transmitted from CP based on CSP's owned partially private key sk_2, namely:
  $h_x = x + r_x = PD_{sk_2}(X, X_1)$;
  $h_y = y + r_y = PD_{sk^2}(Y, Y_1)$;
  $h = h_x \cdot h_y = (x + r_x) \cdot (y + r_y)$;
  Then, CSP encrypts h based on public key pk to get ciphertext $H = [h]$ and sends H to CP.
- **Step-3 (@CP)**: When CP receives H, CP first calculates:
  $[S_1] = [r_x \cdot r_y]^{N-1} = [-r_x \cdot r_y]$;
  $[S_2] = [x]^{N-r_y} = [-r_y \cdot x]$; $[S_3] = [y]^{N-r_x} = [-r_x \cdot y]$;
  Last, CP can compute the result of $[x \cdot y]$: $[x \cdot y] = H \cdot [S_1] \cdot [S_2] \cdot [S_3] = [h - r_x \cdot r_y - r_y \cdot x - r_x \cdot y]$.

b: SECURE LESS THAN Protocol(SLT)

It performs secure integer plaintext comparison on ciphertext, i.e. $[f] = \mathbf{SLT}([x], [y])$, if $x \geq y$, then $f = 0$; otherwise, $f = 1$. Details are as follows:

- **Step-1 (@CP)**: CP computes:
  $[x_1] = ([x])^2 \cdot [1] = [2x + 1]$; $[y_1] = ([y])^2 = [2y]$;
  Then, CP randomly tosses a coin $s \in 0, 1$ and generates a random number r, where r satisfies $r \neq 0$ and $\|r\| \leq \frac{\|N\|}{4}$. If $s = 1$, then CP computes: $[l] = ([x_1] \cdot (y_1)^{N-1})^r = [r(x_1 - y_1)]$. If $s = 0$, then CP computes: $[l] = ([y_1] \cdot (x_1)^{N-1})^r = [r(y_1 - x_1)]$;
  After that, CP uses its partially private key sk_1 to compute $K = PD_{sk_1}([l])$, and then send it to CSP.
- **Step-2 (@CSP)**: CSP decrypts K with its partially private key sk_2 to get l. If $\|l\| \leq \frac{\|N\|}{2}$, CSP sets u to 1; otherwise, set u to 0. Next, CSP encrypts u with the public key pk to get $[u]$, and sends $[u]$to CP.
- **Step-3 (@CP)**: When CP receives $[u]$, CP will compute $[f]$ according to the value of s. If $s = 0$, CP will compute $[f] = [1] \cdot ([u])^{N-1} = [1 - u]$; if $s = 1$, CP will refresh the value of $[u]$, i.e. $[f] = \mathbf{CR}([u])$.

c: SECURE EXCLUSIVE OR CALCULATION Protocol(SXOR)

It performs secure exclusive or result of two integer plaintext on ciphertext, i.e. $[f] = \mathbf{SXOR}([x], [y])$, where $x, y \in \{0, 1\}$. If $x = y, f = 0$; otherwise, $f = 1$. **CP** and **CSP** jointly calculate as follows:

$[f_1] = \mathbf{RSM}([1] \cdot [x]^{N-1}, [y]) = [(1 - x)y]$;
$[f_2] = \mathbf{RSM}([x], [1] \cdot [y]^{N-1}) = [(1 - y)x]$;
$[f] = [x \oplus y] = [f_1] \cdot [f_2] = [x + y - 2xy]$.

d: SECURE EQUIVALENT TESTING Protocol(SEQ)

It performs secure equivalent between two integer plaintext on ciphertext, i.e. $[f] = \mathbf{SEQ}([x], [y])$. If $x = y$, then $f = 0$; otherwise, $f = 1$. **CP** and **CSP** jointly calculate as follows:

$[u_1] = \mathbf{SLT}([x], [y])$; $[u_2] = \mathbf{SLT}([y], [x])$;
$[f] = [u_1 \oplus u_2] = \mathbf{SXOR}([u_1], [u_2])$.

e: SECURE EXPONENT CALCULATION Protocol(SEXP)

It performs secure integer plaintext exponent on ciphertext, i.e. $[f] = \mathbf{SEXP}(x, [y])$ where x is a public integer, $x > 0$, $y \geq 0$ and $f = x^y$. Details are as follows:

- **Step-1 (@CP)**: CP selects a random number $r \in Z_N^*$ and computes:
  $[y_1] = [y] \cdot [r] = [y + r]$; $S = (x^r)^{-1} \mod N$;
  Then, CP uses its own partially private key sk_1 to compute $Y = PD_{sk_1}([y_1])$, and sends $[y_1]$ and Y to CSP.
- **Step-2 (@CSP)**: CSP decrypts Y using its partially private key sk_2 to obtain y_1. Then, CSP computes $h = x^{y_1} \mod N$ and encrypts it into $[h]$ using the public key pk, and sends $[h]$ to CP. Obviously, there exists $h = x^{y_1+r} \mod N$.
- **Step-3 (@CP)**: When CP receives $[h]$, it will compute $[x^y]$ by:
  $[f] = [h]^S = [x^{y_1+r} \cdot (x^r)^{-1}] = [x^y]$.

f: SECURE INVERSE CALCULATION Protocol(SINV)

It performs secure integer plaintext inverse on ciphertext, i.e. $[f] = \mathbf{SINV}([x])$ where $x \neq 0$ and $f = x^{-1} \mod N$. Details are as follows:

- **Step-1 (@CP)**: CP selects a random number $r \in Z_N^*$ and computes $[y_1] = [y] \cdot [r] = [y + r]$.
  Then, CP computes $Y = PD_{sk_1}([y_1])$ using its partially private key sk_1 and sends $[y_1]$ and Y to CSP.
- **Step-2 (@CSP)**: CSP decrypts Y using its partially private key sk_2 to obtain y_1. Then, CSP computes $h = y_1 \mod x$ and encrypts it as $[h]$ using the public key pk. Then it sends $[h]$ to CP. Obviously, there exists $h = (y + r) \mod x$.
- **Step-3 (@CP)**: When CP receives $[h]$, it will compute $[y \mod x]$ by the following process:
  $[U] = [h] \cdot [r \mod x]^{N-1} = [(y + r) \mod x] \cdot [r \mod x]^{N-1} = [((y + r) \mod x) - (r \mod x)]$;
- **Step-4 (@the collaboration of CP and CSP)**: Through the above formula, we can get $-x \leq U \leq x$. But $((y + r) \mod x) - (r \mod x)$ and $y \mod x$ are not always equal. Therefore, the following two cases are discussed. If $-x < U < 0$, then $f = U + x$. If $0 \leq U < x$, then $f = U$. So it will only need to compare the size of $U$and 0:
  $[k_0] = \mathbf{SLT}([U], [0])$; $[f] = [U] \cdot [k_u]^x = [U + x \cdot k_u] = [((y + r) \mod x) - (r \mod x) + x \cdot k_u] = [y \mod x]$.

g: SECURE MODULAR CALCULATION Protocol(SMOD)

It performs secure integer plaintext modular on ciphertext, i.e. $[f] = \mathbf{SMOD}([x], y)$ where y is a public integer and $y \ll N$ and $f = x \mod y$. Details are as follows:

- **Step-1 (@CP)**: CP selects a random number $r \in Z_N^*$ and computes $[x_1] = [x]^r = [r \cdot x]$. Then, CP computes $X = PD_{sk_1}([x_1])$ using its partially private key sk_1 and sends $[x_1]$ and X to CSP.
- **Step-2 (@CSP)**: CSP decrypts X using its partially private key sk_2 to obtain x_1. Then, CSP computes $h = (x_1)^{-1} \mod N$ and encrypts it as $[h]$ using the public key pk. Then it sends $[h]$ to CP. Obviously, there exists $h = (r \cdot x)^{-1} \mod N$.
- **Step-3 (@CP)**:When CP receives $[h]$, it will compute $[x^{-1} \mod N]$ by the following process:
  $[f] = [h]^r = [(r \cdot x)^{-1} \mod N]^r = [(r \cdot x)^{-1} \cdot r \mod N] = [x^{-1} \mod N]$;

2) CONSTRUCTION OF FLOATING-POINT NUMBER CIPHERTEXT(FPN)

In computer, floating-point number is expressed as follows. Given base β and extreme value index e_{min}, e_{max}, there exists at least one triple (s, m, t), so that $x = (-1)^s \cdot m \cdot \beta^t$. In order to facilitate the storage of floating-point ciphertext and the construction of secure outsourcing computation protocols, it is stipulated that all floating-point numbers in the system should use the same significant digit η, base β, sign bit

$s \in \{0, 1\}$, the significant digit m is a ηbit integer, and the exponential bit t satisfies $(e_{min} - \eta + 1) \leq t \leq (e_{max} - \eta + 1)$.

For example, when $\eta = 5, \beta = 10$ are fixed in the system, -31 can be expressed as $(-1)^1 \cdot 31000 \cdot 10^{-3}$, i.e. begin stored as $(1, 31000, -3)$. 1.024 can be expressed as $(-1)^0 \cdot 10240 \cdot 10^{-4}$, i.e. begin stored as $(0, 10240, -4)$. Through the above example, it can be found that this triple expression is uniquely determined when a finite floating-point number is given. When the integer triplet (s_x, m_x, t_x) representing the floating point number x is uploaded to the cloud platform, it is necessary to separately encrypt s_x, m_x, t_x. On the cloud platform, the ciphertext storage form of floating point number x is $([s_x], [m_x], [t_x])$.

3) PROTOCOLS ON FLOATING-POINT NUMBERS

Based on the above two parts(i.e. protocols on integers and construcion of FPN ciphertext), we now introduce the protocols on floating-point numbers [5] which drives the collaboration of CP and CSP to implement addition, multiplication, comparison and other operations of floating point number plaintext on ciphertext.

- **Secure Floating Point Numbers Equivalent Protocol(FPEQ)**: It performs secure equivalent between two floating point number plaintext on ciphertext, i.e. $[f] =$ **FPEQ**$(\llbracket x \rrbracket, \llbracket y \rrbracket)$. If $x = y$, then $f = 0$; otherwise, $f = 1$.
- **Secure Floating Point Numbers Sorting With Absolute Value(SFPS)**: It performs secure equivalent between the absolute values of two floating point number plaintext on ciphertext, i.e. $\llbracket A \rrbracket, \llbracket I \rrbracket =$ **SFPS**$(\llbracket x \rrbracket, \llbracket y \rrbracket)$ where $A = \max(|x|, |y|); I = \min(|x|, |y|)$.
- **Secure Floating Point Numbers Comparison(SFPC)**: It performs secure equivalent between two floating point number plaintext on ciphertext, i.e. $[f] =$ **SFPC**$(\llbracket x \rrbracket, \llbracket y \rrbracket)$. If $x \geq y$, then $f = 1$; otherwise $f = -1$.
- **Secure Floating Point Numbers Addition Protocol(SFPA)**: It performs secure addition of two floating point number plaintext on ciphertext, i.e. $\llbracket f \rrbracket =$ **SFPA**$(\llbracket x \rrbracket, \llbracket y \rrbracket)$ where $\llbracket f \rrbracket = \llbracket x + y \rrbracket$.
- **Secure Floating Point Numbers Multiplication Protocol(SFPM)**: It performs secure multiplication of two floating point number plaintext on ciphertext, i.e. $\llbracket f \rrbracket =$ **SFPM**$(\llbracket x \rrbracket, \llbracket y \rrbracket)$ where $\llbracket f \rrbracket = \llbracket x \cdot y \rrbracket$.

IV. OUR TOOLKIT

In this section, we will introduce the protocols in our toolkit concretely. To make the relationship and interdependence of the protocols clearer, a call graph is presented in Fig 1. Here, our implemented protocols are colored in blue, while the previous protocols in [5] are colored in grey. Similar to the secure outsourcing computation protocols proposed in POCF [5], our extended protocols can handle outliers(NaN) in the same way. For the sake of simplicity, we only describe the process that excludes these outliers. Below we will describe the details of these secure computation protocols.

A. EXTENDED PROTOCOLS ON INTEGERS

1) SECURE INTEGER DIVISION PROTOCOL (SDIV)

Given two encrypted numbers $[x], [y]$, where $0 \leq |x| < \beta_0^{\eta_0}, 0 < |y| < \beta_0^{\eta_0}$, the goal of **SDIV** protocol is to calculate the result $\langle f \rangle$, s.t. $f = \lfloor x \cdot y^{-1} \rfloor$. The idea of SDIV is as follows: we suppose $f = \lfloor |x| \cdot |y|^{-1} \rfloor$. Given the above range of x and y, we can know that $0 \leq \lfloor |x| \cdot |y|^{-1} \rfloor < \beta_0^{\eta_0}$, i.e. $0 \leq f < \beta_0^{\eta_0}$. Therefore, there exists $f' \in [0, \beta_0^{\eta_0})$ s.t. $f' \cdot |y| \leq |x|$. Then f is the maximal f' that satisfies the above inequality.

Based on the above idea, the overall steps of **SDIV** protocol are shown as follows:

Step-1: First, CP and CSP jointly determine whether $x, y \geq 0$,

$[L_x] =$ **SLT**$(\lfloor \frac{N}{2} \rfloor, [x])$; $[1 - L_x] = [1] \cdot [L_x]^{N-1}$;
$[L_y] =$ **SLT**$(\lfloor \frac{N}{2} \rfloor, [y])$; $[1 - L_y] = [1] \cdot [L_y]^{N-1}$;
$[A_x] =$ **RSM**$([L_x], [x]) \cdot$ **RSM**$([1 - L_x], [x]^{N-1})$;
$[A_y] =$ **RSM**$([L_y], [y]) \cdot$ **RSM**$([1 - L_y], [y]^{N-1})$;
Step-2: Initialize $[V] = [0]$.
for$(i = 0$ to $\eta_0 - 1)\{$
  $[u_1] = [0]$; $[U] = [0]$;
  for$(j = 0$ to $\beta_0 - 1)\{$
    $[B_1] = [(\beta_0 - j) \cdot \beta_0^{\eta_0 - 1 - i}]$; $[V_1] = [V] \cdot [B_1]$;
    $[M_1] =$ **RSM**$([A_y], [V_1])$;
    $[L_1] =$ **SLT**$([M_1], [A_x])$;
    $[u_2] =$ **SXOR**$([u_1], [L_1])$;
    $[u_1] = [L_1]$;
    $[M_2] =$ **RSM**$([B_1], [u_2])$; $[U] = [U] \cdot [M_2]$;
  $\}$. Let $[V] = [V] \cdot [U]$.
$\}$
Step-3: Let $[S_1] =$ **SXOR**$([L_x], [L_y])$;
$[S_2] = [1 - S_1] = [1] \cdot [S_1]^{N-1}$; $[V'] = [V]^{N-1}$;
$[f] =$ **RSM**$([S_1], [V']) \cdot$ **RSM**$([S_2], [V])$.

Through the above steps, CP and CSP can jointly compute $[\lfloor f \rfloor] = [\lfloor x \cdot y^{-1} \rfloor] =$ **SDIV**$([x], [y])$.

2) SECURE INTEGER CIPHERTEXT DIVIDED BY INTEGER PLAINTEXT PROTOCOL (SCDP)

Given one encrypted number $[x]$ and a public integer $y (y \neq 0)$, the goal of **SCDP** protocol is to calculate the result $[f]$, s.t. $f = \lfloor x \cdot y^{-1} \rfloor$. The idea of SCDP is as follows: based on the definition of $\lfloor x \cdot y^{-1} \rfloor$, we have:

$$\lfloor \frac{x}{y} \rfloor = \begin{cases} \frac{x - (x \mod y)}{y}, & y > 0 \\ -\frac{x - (x \bmod |y|)}{|y|}, & y < 0 \end{cases} \tag{1}$$

And, $x - (x \mod y)$ can be divided by y, so that we can compute ciphertext of y^{-1} through Secure Inverse Calculation Protocol(SINV), and then compute the result of $[f]$ through Revised Secure Multiplication Protocol (RSM).

The overall steps of **SCDP** protocol is shown as follows:
$[M_1] =$ **SMOD**$(|y|, [x])$; $[M_2] = [x] \cdot [M_1]^{N-1}$;

$[I_1] = \textbf{SINV}([|y|])$;
$[f] = \textbf{RSM}([M_2], [I_1])$; **if**$(y < 0)\{[f] = [f]^{N-1}\}$.
Then, we have $[\lfloor x \cdot y^{-1} \rfloor] = \textbf{SCDP}([x], y)$.

3) SECURE INTEGER CIPHERTEXT MULTIPLY FLOATING POINT NUMBER (SIMF)

Given one encrypted number $[x]$ and a floating point number y, the goal of **SIMF** protocol is to calculate the result $[f]$, s.t. $f = \lfloor x \cdot y \rfloor$. In addition, we need a precision parameter η_1, where η_1 is an integer. The overall process of **SIMF** protocol is shown as follows:

$[Y] = [\lfloor y \cdot 10^{\eta_1} \rfloor]$; $[R] = \textbf{RSM}([x], [Y])$;
Let $[f] = \textbf{SCDP}([R], 10^{\eta_1})$.
Then, we have $[\lfloor x \cdot y \rfloor] = \textbf{SIMF}([x], y)$.

4) SECURE INTEGER POWER PROTOCOL (SPOW)

Given one encrypted number $[x]$ and one public integer y, where $y \geq 0$. The goal of **SPOW** protocol is to calculate the result $[f]$, s.t. $f = \lfloor x^y \rfloor$.

The idea of SPOW is as follows: since y is a positive integer, we can calculate the power within $\mathcal{O}(\log_2 y)$ time complexity through *Exponentiation by Squaring* algorithm. Compared with cumulative multiplication, it can largely reduce the number of calls to RSM, so as to improve the computation efficiency and reduce the communication overhead.

The overall process of **SPOW** protocol is shown as follows:

Step-1:(@CP) Let $B = (b_n \cdots b_1)$ is the binary string of y, where the bit length of y is $n_y = \lfloor \log_2 y \rfloor + 1$.

Step-2: Then, initialize $[f] = [0]$, $[X] = [x]$.

for$(i = 1$ to $n_y)\{$
  if$(b_i == 1)\{[f] = \textbf{RSM}([f], [X])\}$;
  if$(i < n_y)\{[X] = \textbf{RSM}([X], [X])\}$;
$\}$

Through the above steps, CP and CSP can jointly compute $[x^y] = \textbf{SPOW}([x], y)$.

5) SECURE INTEGER LOGARITHM PROTOCOL (SLOG)

Given one encrypted number $[x]$ and a public integer y, where $y \geq 2$ and $0 < x < y^{\delta}$. The goal of **SLOG** protocol is to calculate the result $[f]$, s.t. $f = \lfloor \log_y x \rfloor$. The overall process of **SLOG** protocol is shown as follows:

Initialize $[f] = [0]$, $[u_1] = [0]$.
for$(i = 1$ to $\delta)\{$
  $[u_2] = \textbf{SLT}([x], [y^i])$; $[v] = \textbf{SXOR}([u_1], [u_2])$;
  $[u_1] = [u_2]$; $[f] = [f] \cdot \textbf{RSM}([v], [i])$;
$\}$.
Let $[f] = [f] \cdot [-1]$.
Then, we have $[\lfloor \log_y x \rfloor] = \textbf{SLOG}([x], y)$.

6) SECURE INTEGER ARRAY MAXIMUM (SIAX)

Given an integer ciphertext array $C = \{[x_0], [x_1], \cdots, [x_{n-1}]\}$, where $n \geq 2$. We calculate: $[x_{max}] = [x_0]$; $[i_{max}] = [0]$;

for $(i = 1$ to $n - 1)\{$
  $[r_1] = \textbf{SLT}([x_{max}], [x_i])$; $[r_2] = \textbf{SEQ}([r_1], [1])$;
  $[1 - r_2] = [1] \cdot [r_2]^{N-1}$;
  $[i_{max}] = \textbf{RSM}([i_{max}], [r_2]) \cdot \textbf{RSM}([i], [1 - r_2])$;
  $[x_{max}] = \textbf{RSM}([x_{max}], [r_2]) \cdot \textbf{RSM}([x_i], [1 - r_2])$.
$\}$

Then, we have $[i_{max}], [x_{max}] = \textbf{SIAX}(C)$, where $i_{max} = \arg\max\{x_0, \cdots, x_{n-1}\}$, $x_{max} = \max\{x_0, \cdots, x_{n-1}\}$. Similarly, we can implement Secure Integer Array Maximum(SIAN) protocol, i,e. $[i_{min}], [x_{min}] = \textbf{SIAN}(C)$, where $i_{max} = \arg\min\{x_0, \cdots, x_{n-1}\}$, $x_{min} = \min\{x_0, \cdots, x_{n-1}\}$.

B. EXTENDED PROTOCOLS ON FLOATING-POINT NUMBERS

1) SECURE FLOATING POINT NUMBER OPPOSITE (SFPO)

Given one encrypted FPN $\langle x \rangle = \{[s_x], [m_x], [t_x]\}$, the goal of **SFPO** protocol is to calculate the result $\langle f \rangle$, s.t. $f = -x$. The overall process of **SFPO** protocol can be executed only by **CP**:

$[s_f] = ([s_x])^{N-1}$; $[m_f] = [m_x]$, $[t_f] = [t_x]$; Let $\langle f \rangle = \{[s_f], [m_f], [t_f]\}$.
Then, we have $\langle -x \rangle = \textbf{SFPO}(\langle x \rangle)$.

2) SECURE FLOATING POINT NUMBER SUBTRACTION (SFPU)

Given two encrypted FPNs $\langle x \rangle$ and $\langle y \rangle$, the goal of **SFPU** protocol is to calculate the result $\langle f \rangle$, s.t. $f = x - y$. Since subtraction is the inverse of addition, the overall process of **SFPU** protocol is shown as follows:

- **Step-1(@CP)**: $\langle -y \rangle = \textbf{SFPO}(\langle y \rangle)$;
- **Step-2**: $\langle x - y \rangle = \textbf{SFPA}(\langle x \rangle, \langle -y \rangle)$.

Then, we have $\langle x - y \rangle = \textbf{SFPU}(\langle x \rangle, \langle y \rangle)$.

3) SECURE FLOATING POINT NUMBER RECIPROCAL (SFPR)

Given one encrypted FPN $\langle x \rangle = \{[s_x], [m_x], [t_x]\}$, the goal of **SFPR** protocol is to calculate the result $\langle f \rangle$, s.t. $f = x^{-1}$. Assuming that $f = x^{-1}(x \neq 0)$, f can also be converted to a triple $\{s_f, m_f, t_f\}$. According to the property of reciprocal, we can easily have $s_f = s_x$. Then the main problem is how to determine m_f and t_f. Notice that $\beta^{\eta-1} \leq m_x < \beta^{\eta}$, $\beta^{\eta-1} \leq m_f < \beta^{\eta}$. Therefore, there exists $m \in (\beta^{\eta-1}, \beta^{\eta}]$ s.t. $m_x \cdot m \leq \beta^{2\eta-1}$. Then m_f is the maximal m that satisfies the above inequality.

Next, we will discuss this problem in two cases according to the value of m'_f.

Case I: if $m'_f = \beta^{\eta}$, then the number of significant digits is $\eta + 1$ bit. Since $\beta^{\eta-1} \leq m_f < \beta^{\eta}$, we need to decrease m'_f by one bit, i.e. $m'_f = \beta^{\eta-1}$.

Case II: if $m'_f \in [\beta^{\eta-1}, \beta^{\eta})$, then it is consistent with m_f in terms of value range.

Then we confirm the value of t_f, we have:

$$\begin{aligned}(m_f \cdot \beta^{t_f}) \cdot (m_x \cdot \beta^{t_x}) &= (m_f \cdot m_x) \cdot (\beta^{t_f} \cdot \beta^{t_x}) \\ &= (m_f \cdot m_x) \cdot \beta^{t_f + t_x} \\ &\approx 1 \end{aligned} \tag{2}$$

Consider $m_f \cdot m_x \approx \beta^{2\eta-1}$, we have $t_f + t_x = 1 - 2\eta$, i.e. $t_f = 1 - 2\eta - t_x$.

Based on the above analysis, the overall steps of **SFPR** protocol are shown as follows:

Step-1: CP and CSP first jointly handle the special cases. We determine whether $\langle x\rangle$ is equal to $\langle 0\rangle$, if $\langle x\rangle$ is equal to $\langle 0\rangle$, then the result is $\langle 0\rangle$. This can be achieved by the following calculations:

$[Z] = \mathbf{FPEQ}(\langle x\rangle, \langle 0\rangle)$; $[1-Z] = [1]\cdot[Z]^{N-1}$.

Step-2: Then CP and CSP can jointly evaluate the significance of f. Initialize $[v] = [0]$, $[E_1] = [\beta^{2\eta-1}]$.

for($i = 0$ to η){
  $[E_2] = [\beta^{\eta-i}]$; $[u_1] = [0]$; $[U] = [0]$;
  for($j = 1$ to β){
    $[E_3] = \mathbf{RSM}([j], [E_2])$; $[V_1] = [v]\cdot[E_3]$;
    $[V_2] = \mathbf{RSM}([V_1], [m_x])$; $[u_2] = \mathbf{SLT}([E_1], [V_2])$;
    $[u] = \mathbf{SXOR}([u_1], [u_2])$; $[u_1] = [u_2]$;
    $[U] = [U]\cdot\mathbf{RSM}([u], [j-1])$;
  }. Let $[v] = [v]\cdot\mathbf{RSM}([U], [E_2])$.
}

Step-3: Finally, CP and CSP can jointly evaluate the exponents of f by combining the two cases in the analysis. Then we calculate:

$[v^*] = \mathbf{SMOD}(\beta^\eta, [v])$; $[r] = \mathbf{SEQ}([v^*], [v])$;
$[1-r] = [1]\cdot[r]^{N-1}$; $[B_0] = [\beta^{\eta-1}]$;
$[m_f] = \mathbf{RSM}([r], [B_0])$; $[m_f^*] = \mathbf{RSM}([m_f], [Z])$;
$[s_f] = [s_x]$; $[T_1] = \mathbf{RSM}([r], [Z])$; $[t_1] = [t_x]^{N-1}\cdot[T_1]$;
$[t_2] = \mathbf{RSM}([\eta_{max}], [1-Z])$; $[t_f] = [t_1]\cdot[t_2]$;
$[t_f^*] = [t_f]\cdot[2\eta-1]^{N-1}$; $\langle f\rangle = \{[s_f], [m_f^*], [t_f^*]\}$.

Then, CP and CSP can jointly compute $\langle x^{-1}\rangle = \mathbf{SFPR}(\langle x\rangle)$.

4) SECURE FLOATING POINT NUMBER DIVISION (SFPD)

Given two encrypted FPNs $\langle x\rangle$, $\langle y\rangle$, the goal of **SFPD** protocol is to calculate the result $\langle f\rangle$, s.t. $f = x\cdot y^{-1}$. The overall process of **SFPD** protocol are shown as follows:

$\langle y^{-1}\rangle = \mathbf{SFPR}(\langle y\rangle)$; $\langle f\rangle = \mathbf{SFPM}(\langle x\rangle, \langle y^{-1}\rangle)$.

Then, we have $\langle x\cdot y^{-1}\rangle = \mathbf{SFPD}(\langle x\rangle, \langle y\rangle)$.

C. CIPHERTEXT MUTUAL CONVERSION PROTOCOLS

1) SECURE INTEGER TO FLOATING POINT NUMBER (SITF)

Given one encrypted number $[x]$ where ($|x| < 10^{2\eta}$), the goal of **SITF** protocol is to calculate the result $\langle x\rangle$. First, we need to determine the sign of x, which is easy to achieve. Then the question can be discussed in two cases:

Case I: If $|x| \geq \beta^\eta$, then $t_x > 0$. We should keep the first η significant digits of x as m_x.

Case II: If $|x| < \beta^\eta$, then $t_x \leq 0$. We should expand x to η digits.

Based on the above idea, the overall steps of **SITF** protocol are shown as follows:

Step-1: First, CP and CSP jointly determine whether x is equal to 0, this can be achieved by the following calculations:

$[Z] = \mathbf{SEQ}([x], [0])$; $[1-Z] = [1]\cdot[Z]^{N-1}$.

Step-2: Then, CP and CSP jointly determine whether x is greater than 0.

$[S_1] = [\lfloor\frac{N}{2}\rfloor]$; $[s] = \mathbf{SLT}([S_1], [x])$;
$[1-s] = [1]\cdot[s]^{N-1}$; $[-x] = [x]^{N-1}$;
$[|x|] = \mathbf{RSM}([s], [x])\cdot\mathbf{RSM}([1-s], [-x])$.

Step-3: CP and CSP jointly determine whether $|x|$ is equal or greater than β^η.

$[B_0] = \mathbf{SLT}([|x|], [\beta^\eta])$; $[1-B_0] = [1]\cdot[B_0]^{N-1}$.

Step-4: Then, CP and CSP jointly solve the first case that $|x| \geq \beta^\eta$.

$[u_1] = [0]$; $[T_1] = [0]$; $[m_1] = [0]$; $[A_1] = [|x|]$.
for($i = 0$ to $\eta_{max} - 1$){
  $[M_1] = \mathbf{SMOD}(\beta^{\eta_{max}-i-1}, [|x|])$;
  $[E_1] = [\eta_{max} - i - 1]$;
  $[u_1^*] = \mathbf{SEQ}([M_1], [|x|])$; $[t_1] = \mathbf{SXOR}([u_1^*], [u_1])$;
  $[T_1] = [T_1]\cdot\mathbf{RSM}([t_1], [E_1])$;
  $[u_1] = [u_1^*]$; $[v_1] = [A_1]\cdot[M_1]^{N-1}$;
  $[A_1] = [A_1]\cdot[V_1]^{N-1}$; $[v_1]_i = [v_1]$;
}
$[T_2] = [T_1]\cdot[\eta]^{N-1}$; $[T_3] = [1]\cdot[T_2]$;
$[B_2] = \mathbf{SEXP}(\beta, [T_3])$; $[I_2] = \mathbf{SINV}([B_2])$;
for($i = 0$ to $\eta_{max} - 1$){
  $[E_1] = [\eta_{max} - i - 1]$; $[r_1] = \mathbf{SLT}([T_2], [E_1])$;
  $[v_1] = [v_1]_i$; $[v_1^*] = \mathbf{RSM}([v_1], [I_2])$;
  $[V_1] = \mathbf{RSM}([r_1], [v_1^*])$; $[m_1] = [m_1]\cdot[V_1]$;
}

Step-5: After that, CP and CSP jointly solve the second case that $0 < |x| < \beta^\eta$.

$[u_2] = [0]$; $[T_4] = [0]$; $[m_2] = [0]$; $[A_2] = [|x|]$.
for($i = 0$ to $\eta - 1$){
  $[E_2] = [\eta - i - 1]$; $[M_2] = \mathbf{SMOD}(\beta^{\eta-i-1}, [|x|])$;
  $[u_2^*] = \mathbf{SEQ}([M_2], [|x|])$; $[t_2] = \mathbf{SXOR}([u_2^*], [u_2])$;
  $[T_5] = [T_5]\cdot\mathbf{RSM}([t_2], [E_2])$; $[u_2] = [u_2^*]$;
  $[v_2] = [A_2]\cdot[M_2]^{N-1}$; $[A_2] = [A_2]\cdot[v_2]^{N-1}$;
  $[v_2]_i = [v_2]$;
}. Let $[T_6] = [\eta]\cdot[T_5]^{N-1}$ and $[B_3] = \mathbf{SEXP}(\beta, [T_6])$.
for($i = 0$ to $\eta - 1$){
  $[E_2] = [\eta - i - 1]$; $[r_2] = \mathbf{SLT}([T_5], [E_2])$;
  $[1-r_2] = [1]\cdot[r_2]^{N-1}$; $[v_2] = [v_2]_i$;
  $[v_2^*] = \mathbf{RSM}([v_2], [B_3])$;
  $[V_2] = \mathbf{RSM}([1-r_2], [v_2^*])\cdot\mathbf{RSM}([v_2], [0])$;
  $[m_2] = [m_2]\cdot[V_2]$;
}. Let $[T_7] = [T_6]^{N-1}$.

Step-6: Then CP and CSP jointly calculate:

$[m] = \mathbf{RSM}([1-B_0], [m_1])\cdot\mathbf{RSM}([B_0], [m_2])$;
$[t] = \mathbf{RSM}([1-B_0], [t_1])\cdot\mathbf{RSM}([B_0], [t_2])$;

Step-7: Finally, CP and CSP jointly calculate the final result:

$\langle 0\rangle = \{[s_0], [m_0], [t_0]\}$;
$[s_f^*] = \mathbf{RSM}([1-Z], [s_0])\cdot\mathbf{RSM}([Z], [1-s])$;
$[t_f^*] = \mathbf{RSM}([1-Z], [t_0])\cdot\mathbf{RSM}([Z], [t])$;
$[m_f^*] = \mathbf{RSM}([1-Z], [m_0])\cdot\mathbf{RSM}([Z], [m])$;

Let $\langle f\rangle = \{[s_f^*], [m_f^*], [t_f^*]\}$. Then, CP and CSP can jointly compute $\langle x\rangle = \mathbf{SITF}([x])$.

2) SECURE FLOATING POINT NUMBER TO INTEGER (SFTI)

Given one encrypted number $\langle x\rangle$, the goal of **SFTI** protocol is to calculate the result $[f]$, s.t. $f = \lfloor x\rfloor$. For a floating point number x, we can easily obtain the sign of x by s_x. Then the question can be discussed in three cases.

Case I: if $t_x \geq 0$, then $x = (-1)^{s_x} \cdot m_x \cdot \beta^{t_x}$.

Case II if $t_x \leq -\eta$, since $\beta^{\eta-1} \leq m_x < \beta^{\eta}$, we have $0 \leq m_x \cdot \beta^{t_x} < 1$ which means $|x| < 1$. Therefore, $x = 0$.

Case III: if $-\eta < t_x < 0$, we just keep the first $\eta + t_x$ significant digits as $|x|$.

Based on the above analysis, the overall steps of **SFTI** protocol are shown as follows:

Step-1: First, CP and CSP jointly determine whether m_x is equal to 0, this can be achieved by the following calculations: $[Z_m] = \mathbf{SEQ}([m_x], [0])$; $[1 - Z_m] = [1] \cdot [Z_m]^{N-1}$;

Step-2: Then, CP and CSP jointly determine whether s_x is equal to 0, this can be achieved by the following calculations: $[Z_s] = \mathbf{SEQ}([m_s], [0])$; $[1 - Z_s] = [1] \cdot [Z_s]^{N-1}$;

Step-3: CP and CSP jointly divide t_x into three conditions. First, we determine whether t_x is equal or greater than 0.

Let $[T_1] = \mathbf{SLT}([t_x], [0])$; $[1 - T_1] = [1] \cdot [T_1]^{N-1}$; Then, we determine whether t_x is less than or equal to $-\eta$. Let $[T_2] = \mathbf{SLT}([-\eta], [t_x])$; $[1 - T_2] = [1] \cdot [T_2]^{N-1}$.

Step-4: Now, CP and CSP jointly solve the first condition that $t_x \geq 0$.

$[B_1] = \mathbf{SEXP}(\beta, [t_x])$; $[x_1] = \mathbf{RSM}([m_x], [B_1])$.

Step-5: Next, CP and CSP jointly solve the second condition that $t_x \leq -\eta$. We have $[x_2] = [0]$.

Step-6: Finally, CP and CSP jointly solve the third condition that $-\eta < t_x < 0$.

$[x_3] = [0]$; $[-t_x] = [t_x]^{N-1}$; $[B_2] = \mathbf{SEXP}(\beta, [-t_x])$;
$[I_1] = \mathbf{SINV}([B_2])$; $[M] = [m_x]$.
for $(i = 0$ to $\eta - 2)\{$
$[u] = \mathbf{SMOD}(\beta^{\eta-1-i}, [m_x])$; $[v] = [M] \cdot [u]^{N-1}$;
$[r] = \mathbf{RSM}([v], [I_1])$; $[M] = [M] \cdot [v]^{N-1}$;
$[E_1] = [\eta - i]$; $[q] = \mathbf{SLT}([-t_x], [E_1])$;
$[1 - q] = [1] \cdot [q]^{N-1}$; $[r^*] = \mathbf{RSM}([q], [r])$;
$[x_3] = [x_3] \cdot [r^*]$;
$\}$

Step-7: CP and CSP jointly combine the above cases.

$[X_1] = \mathbf{RSM}([1 - T_2], [x_2]) \cdot \mathbf{RSM}([T_2], [x_3])$;
$[X_2] = \mathbf{RSM}([1 - T_1], [x_1]) \cdot \mathbf{RSM}([T_1], [X_1])$;
$[X] = \mathbf{RSM}([1 - Z_m], [0]) \cdot \mathbf{RSM}([Z_m], [X_2])$;
$[-X] = [X]^{N-1}$.

Step-8: CP and CSP jointly calculate the final result.

$[f] = \mathbf{RSM}([1 - Z_s], [X]) \cdot \mathbf{RSM}([Z_s], [-X])$.

Based on the above steps, CP and CSP can compute $[\lfloor x \rfloor] = \mathbf{SFTI}(\langle x \rangle)$.

V. SECURE AND EVALUABLE CLUSTERING

In this section, we first introduce Square Euclidean Distance, which will be used as a measure of the distance between samples. Next, we give the implementation of the homomorphic algorithm of K-means. To evaluate the performance of the clustering results on ciphertext, we design and implement the homomorphic silhouette coefficient.

A. SECURE SQUARE EUCLIDEAN DISTANCE (SSED)

Given two arrays of the same length, $x = \{x_0, \cdots, x_{n-1}\}$ and $y = \{y_0, \cdots, y_{n-1}\}$, then the square euclidean distance $\mathbf{SED}(x, y) = \sum_{i=0}^{n-1}(x_i - y_i)^2$. Therefore, given two integer ciphertext arrays of the same length, $x = \{[x_0], \cdots, [x_{n-1}]\}$ and $y = \{[y_0], \cdots, [y_{n-1}]\}$, the square euclidean distance on integer ciphertext is defined in Algorithm 1.

Algorithm 1 Secure Square Euclidean Distance(SSED)

Input: $x = \{[x_0], \cdots, [x_{n-1}]\}$, $y = \{[y_0], \cdots, [y_{n-1}]\}$
Output: the integer ciphertext $[D]$ of square euclidean distance between x and y

Initialize $[D] = [0]$;
for $i = 0$ to $n - 1$ **do**
  $[d_i] = [x_i] \cdot [y_i]^{N-1}$; $[s_i] = \mathbf{RSM}([d_i], [d_i])$;
  $[D] = [D] \cdot [s_i]$;
end for
return $[D]$;

B. SECURE K-MEANS CLUSTERING

K-means is a commonly-used cluster algorithm. Given a set of samples(i.e. the observations) $X = \{x_0, \cdots, x_{m-1}\}$ where each feature x_i has n features, i.e. $x_i = \{x_i^0, \cdots, x_i^{n-1}\}$. The algorithm partitions the samples into K disjoint subsets $S = \{S_1, S_2, \ldots, S_K\}$ so as to minimize the within-cluster sum of squares. Here, we will introduce K-means algorithm module by module. Then, we will introduce the implementation of secure K-means clustering based on our toolkit MPOCT.

1) SECURE SAMPLE PARTITION

For each sample $x_i (0 \leq i < m)$ in the dataset, the K-means algorithm calculates the distance $D_{i,k}$ from each sample x_i to each cluster center C_k, and assigns the sample x_i to the cluster to which the nearest cluster center belongs. Suppose C_i is the index of the cluster to which sample x_i belongs. The implementation of the above partition of samples on ciphertext is shown in Algorithm 2.

Algorithm 2 Secure Sample Partition(SSP)

Input: i) encrypted samples $X = \{x_0, \cdots, x_{m-1}\}$;
  ii) encrypted centroids of cluster $C = \{C_0, \cdots, C_{K-1}\}$;
Output: i) encrypted cluster index of samples $c = \{[c_0], \cdots, [c_{K-1}]\}$;

for $j = 0$ to $m - 1$ **do**
  for $k = 0$ to $K - 1$ **do**
    Calculate $[d_{j,k}] = \mathbf{SSED}(x_j, C_k)$;
  end for
  Let $d_j = \{[d_{j,0}], \cdots, [d_{j,K-1}]\}$;
  $[c_j], [\min\{d_{j,0}, d_{j,1}, \cdots, d_{j,K-1}\}] = \mathbf{SIAN}(d_j)$;
end for
Let $c = \{[c_0], [c_1], \cdots, [c_{m-1}]\}$;
return c;

2) SECURE CLUSTER UPDATING

For each partitioned cluster set S_k, K-means algorithm will recalculate its cluster center $C_k = (C_k^0, C_k^1, \cdots, C_k^{n-1})$ where the i-th center of the j-th feature are computed

as follows:

$$C_k^j = \frac{1}{|S_k|} \sum_{x_i \in S_k} x_i^j, \quad 0 \le j < n \tag{3}$$

The implementation of the above cluster updating approach on ciphertext is shown in Algorithm 3.

Algorithm 3 Secure Cluster Updating(SCU)

Input: i) encrypted samples $X = \{x_0, \cdots, x_{m-1}\}$;
ii) encrypted cluster index of samples $c = \{[c_0], \cdots, [c_{K-1}]\}$;
Output: i) encrypted cluster centroids $C = \{C_0, \cdots, C_{K-1}\}$;

for $k = 0$ to $K-1$ **do**
  Let $[n_k] = [0]$;
  for $j = 0$ to $m-1$ **do**
    $[E_1] = \mathbf{SEQ}([c_j], [k])$; $[1 - E_1] = [1] \cdot [E_1]^{N-1}$;
    Let $[C_k^l] = [0] (0 \le l < n)$;
    for $l = 0$ to $n-1$ **do**
      $[t_j^l] = \mathbf{RSM}([1 - E_1], [x_j^l])$; $[C_k^l] = [C_k^l] \cdot [t_j^l]$;
    end for
    $[n_k] = [n_k] \cdot [1 - E_1]$;
  end for
  for $l = 0$ to $n-1$ **do**
    $[C_k^l] = \mathbf{SDIV}([C_k^l], [n_k])$;
  end for
  Let $C_k = \{[C_k^0], [C_k^1], \cdots, [C_k^{n-1}]\}$
end for
Let $C = \{C_0, C_1, \cdots, C_{K-1}\}$;
return C;

3) SECURE K-MEANS

Based on the above two modules, for a given number of iterations E, the implementation of K-means clustering algorithm on ciphertext are shown in Algorithm 4.

Algorithm 4 Secure K-means(SKM)

Input: i)Encrypted samples $X = \{x_0, \cdots, x_{m-1}\}$;
ii) Cluster number K;
iii) Iteration number E;
Output: i) Encrypted cluster index of samples $c = \{[c_0], \cdots, [c_{K-1}]\}$;
ii) Encrypted cluster centroids $C = \{C_0, \cdots, C_{K-1}\}$;

Random choice $k (k \le m)$ samples $x_{r_1}, \cdots, x_{r_k}$ as cluster centroids $C = \{C_0, \cdots, C_{K-1}\}$.
for $i = 0$ to $E-1$ **do**
  Partition the samples to K clusters: $c = \mathbf{SSP}(X, C)$;
  Updating cluster centroids: $C = \mathbf{SCU}(X, c)$;
end for
return C, c;

C. EVALUABLE CLUSTERING ON CIPHERTEXT

Since the previous schemes did not provide an evaluation index on the ciphertext of the clustering result, DOs can only obtain the clustering results returned by the server. However, it is impossible to evaluate the clustering results due to DOs' limited local resources. Here, we choose silhouette coefficient [28] as the evaluation index of clustering. Here we describe the silhouette coefficient module by module and give the corresponding implementation on ciphertext based on our tookit MPOCT.

1) SECURE DISTANCE CALCULATION BETWEEN SAMPLES

First, the silhouette coefficient needs to calculate the distance between two samples $d_{i,j}$. This is easy to implement on ciphertext, which is shown in algorithm 5.

Algorithm 5 Secure Distance Calculation Between Samples(SDCS)

Input: i) sample number m;
ii) cluster number K;
iii) encrypted sample set $X = \{x_0, \cdots, x_{m-1}\}$;
Output: i) the encrypted distance between samples $d = \{[d_{i,j}]\}$;

Let $d_{i,j}$is the distance between x_i and x_j, then $[d_{i,i}] = [0], (0 \le i \le m-1)$.
for $i = 0$ to $m-1$ **do**
  for $j = i+1$ to $m-1$ **do**
    Calculate $[d_{i,j}] = \mathbf{SSED}(x_i, x_j)$;
    Let $[d_{j,i}] = [d_{i,j}]$;
  end for
end for
Let $d = \{[d_{i,j}]\}$;
return d;

2) SECURE DISTANCE CALCULATION BETWEEN SAMPLE AND CLUSTER

Next, the silhouette coefficient needs to calculate the distance between each sample x_i and each cluster S_k, which is defined as follows:

$$D_{i,k} = \sum_{x_j \in S_k} d_{i,j} \tag{4}$$

Let N_k is the number of samples contained in each cluster, $E_{i,k} \in \{0, 1\}$ is the relationship between sample x_i and cluster c_k, T is the product of all N_k, i.e. $T = \prod_{i=0}^{K-1} N_k$. The implementation of the above process on the ciphertext is shown in Algorithm 6.

3) SECURE COHESION CALCULATION

Then, we come to the calculation of cohesion. For sample x_i, its cohesion A_i is defined as follows:

$$A_i = \frac{\sum_{x_j \in S_i} \mathbf{SSED}(x_i, x_j)}{|S_{c_i}| - 1} \tag{5}$$

where S_{c_i} is the cluster to which x_i belongs and $|S_{c_i}|$ is number of samples in S_{c_i}. The implementation process of the above formula on the ciphertext is shown in Algorithm 7.

Algorithm 6 Secure Distance Calculation Between Sample and Cluster(SDCSC)

Input: i) sample number m;
ii) cluster number K;
iii) the encrypted distance between samples $d = \{[d_{i,j}]\}$;
iv) encrypted cluster index of samples $c = \{[c_0], \cdots, [c_{K-1}]\}$;

Output: i) encrypted distance between samples to clusters $D = \{[D_{i,k}]\}$;
ii) the encrypted number of samples contained in each cluster$N = \{[N_0], [N_1], \cdots, [N_{K-1}]\}$;
iii) the encrypted relationship matrix between samples and clusters $E = \{[E_i, k]\}$;
iv) the encrypted product T;

for $k = 0$ to $K - 1$ **do**
  for $i = 0$ to $m - 1$ **do**
    $[e_0] = \mathbf{SEQ}([k], [c_i])$;
    Let $[E_{i,k}] = [1] \cdot [e_0]^{N-1} = [1 - e_0]$;
    Update N_k: $[N_k] = [N_k] \cdot [E_{i,k}]$
    for $j = 0$ to $m - 1$ **do**
      $[U] = \mathbf{RSM}([E_{i,k}], [d_{i,j}])$;
      Update $D_{j,k}$: $[D_{j,k}] = [D_{j,k}] \cdot [U]$;
    end for
  end for
end for
Initial $[T] = [1]$, which is used to calculate the product of all N_k;
for $k = 0$ to $K - 1$ **do**
  $[T] = \mathbf{RSM}([T], [N_k])$;
end for
Let $D = \{[D_{i,k}]\}$;
Let $E = \{[E_{i,k}]\}$;
Let $N = \{[N_0], [N_1], \cdots, [N_{K-1}]\}$
return D, N, E, T;

4) SECURE SEPARATION CALCULATION

Then we come to the calculation of separation b_i. For sample x_i, separation b_i is defined as follows:

$$B_i = \min_{0 \le k < K, k \ne c_i} \{ \frac{\sum_{x_j \in S_k} d(x_i, x_j)}{|S_k|} \} \tag{6}$$

The implementation process of the above formula on the ciphertext is shown in Algorithm 8.

5) SECURE SILHOUETTE COEFFICIENT

Silhouette coefficient combines cohesion and separation. Based on the above modules, we can calculate the silhouette coefficient of x_i and the overall silhouette coefficient sc, i.e.

$$sc_i = \frac{B_i - A_i}{\max(A_i, B_i)} \tag{7}$$

$$sc = \frac{1}{m} \sum_{i=1}^{m} sc_i \tag{8}$$

Algorithm 7 Secure Cohesion Calculation(SCOC)

Input: i) sample number m;
ii) cluster number K;
iii) encrypted distance between samples to clusters $D = \{[D_{i,k}]\}$;
iv) the encrypted relationship matrix between samples and clusters $E = \{[E_i, k]\}$;
v) the encrypted number of samples contained in each cluster$N = \{[N_0], [N_1], \cdots, [N_{K-1}]\}$;

Output: i) encrypted cohension of all samples $A = \{[A_0], [A_1], \cdots [A_{m-1}]\}$;

$[P_i] = [0], [Q_i] = [0], 0 \le i < m$;
for $k = 0$ to $K - 1$ **do**
  $[e_1] = \mathbf{SEQ}([1], [N_k])$;
  $[e_2] = [1] \cdot [e_1]^{N-1} = [1 - e_1]$;
  $[n_k] = [N_k] \cdot [-1] = [N_k - 1]$;
  for $i = 0$ to $m - 1$ **do**
    $[u_0] = \mathbf{RSM}([E_{i,k}], [n_k])$;
    $[u_0] = \mathbf{RSM}([u_0], [e_1]) \cdot \mathbf{RSM}([1], [e_2])$;
    $[Q_i] = [Q_i] \cdot [u_0] = [Q_i + u_0]$;
    $[v_{i,k}] = \mathbf{RSM}([D_{i,k}], [E_{i,k}])$;
    $[v_{i,k}] = \mathbf{RSM}([v_{i,k}], [e_1])$;
    $[P_i] = [P_i] \cdot [v_{i,k}]) = [P_i + v_{i,k}]$
  end for
end for
for $i = 0$ to $m - 1$ **do**
  $\langle P_i \rangle = \mathbf{SITF}([P_i])$; $\langle Q_i \rangle = \mathbf{SITF}([Q_i])$;
  $\langle A_i \rangle = \mathbf{SFPD}(\langle P_i \rangle, \langle Q_i \rangle)$;
end for
Let $A = \{\langle A_0 \rangle, \langle A_1 \rangle, \cdots, \langle A_{m-1} \rangle\}$;
return A;

Thus we can implement the silhouette coefficient on the ciphertext based on our toolkit MPOCT, which is shown in Algorithm 9.

VI. SECURITY ANALYSIS

In this section, we first analyze the security of our outsourcing computation protocols. Then, we demonstrate the security of our toolkit MPOCT.

A. THE SECURITY OF PROTOCOLS

Before we explain the security of our protocols, we first give the definition of the semantic security of a public key encryption system that supports partial decryption. As the semantic security of PCPD has been proven in [5], we will use it to demonstrate that our outsourcing computation protocols are secure.

Definition 1 (Semantic Security): Let $\mathcal{E} =$ (**KeyGen**, **Enc**, **Dec**) *be a public-key encryption scheme supporting partial decryption. We say that* $\mathcal{E}$ *is semantically secure if for any polynomial-time adversary* $\mathcal{A}$ *it has a negligible advantage (in the security parameter) in the following experiment (between the challenger and the adversary):*

Algorithm 8 Secure Separation Calculation(SSEC)

Input: i) sample number m;
ii) cluster number K;
iii) encrypted distance between samples to clusters $D = \{[D_{i,k}]\}$;
iv) the encrypted relationship matrix between samples and clusters $E = \{[E_i, k]\}$;
v) the encrypted number of samples contained in each cluster$N = \{[N_0], [N_1], \cdots, [N_{K-1}]\}$;
vi) the encrypted product T;

Output: i) encrypted separation of all samples $B = \{[B_0], [B_1], \cdots [B_{m-1}]\}$;

Let $[G] = [10^{64}]$;
for $k = 0$ to $K-1$ **do**
  $[V_k] = \mathbf{SINV}([N_k])$; $[R_k] = \mathbf{RSM}([T], [V_k])$;
  for $i = 0$ to $m-1$ **do**
    $[e_3] = [1] \cdot [E_{i,k}]^{N-1}$; $[H] = \mathbf{RSM}([D_{i,k}], [R_k])$;
    $[z_0] = \mathbf{RSM}([E_{i,k}], [G])$;
    $[z_1] = \mathbf{RSM}([e_3], [H])$;
    $[Z_{i,k}] = [z_0] \cdot [z_1]$;
  end for
end for
for $i = 0$ to $m-1$ **do**
  Let $[Z_i] = \{[Z_{i,0}], [Z_{i,1}], \cdots, [Z_{i,K-1}]\}$;
  $[I_i], [m_i] = \mathbf{SIAN}([Z_i])$;
  $\langle m_i \rangle = \mathbf{SITF}([m_i])$; $\langle T \rangle = \mathbf{SITF}([T])$;
  $\langle B_i \rangle = \mathbf{SFPD}(\langle m_i \rangle, \langle T \rangle)$;
end for
Let $B = \{\langle B_0 \rangle, \langle B_1 \rangle, \cdots, \langle B_{m-1} \rangle\}$;
return B;

Algorithm 9 Secure Silhouette Coefficient(SSC)

Input: i) sample number m;
ii) cluster number K;
iii) encrypted sample set $X = \{x_0, \cdots, x_{m-1}\}$;
iv) encrypted cluster index of samples $c = \{[c_0], \cdots, [c_{K-1}]\}$;

Output: i) encrypted silhouette coefficient $\langle sc \rangle$;

$d = \mathbf{SDCS}(m, K, X)$;
$D, N, E, T = \mathbf{SDCSC}(m, K, d, c)$;
$A = \mathbf{SCOC}(m, K, D, E, N)$;
$B = \mathbf{SSEC}(m, K, D, E, N, T)$;
for $i = 0$ to $m-1$ **do**
  $\langle U_i \rangle = \mathbf{SFPS}(\langle B_i \rangle, \langle A_i \rangle)$;
  Compare A_i with B_i: $[l_i] = \mathbf{SFPC}(\langle A_i \rangle, \langle B_i \rangle)$;
  $[L_i] = \mathbf{SEQ}([l_i], [1])$; $\langle L_i \rangle = \mathbf{SITF}([L_i])$;
  $\langle 1 - L_i \rangle = \mathbf{SFPS}(\langle 1 \rangle, \langle L_i \rangle)$;
  $\langle M_0 \rangle = \mathbf{SFPM}(\langle L_i \rangle, \langle A_i \rangle)$;
  $\langle M_1 \rangle = \mathbf{SFPM}(\langle 1 - L_i \rangle, \langle B_i \rangle)$;
  $\langle W_i \rangle = \mathbf{SFPM}(\langle M_0 \rangle, \langle M_1 \rangle) = \langle \max(A_i, B_i) \rangle$;
  $\langle sc_i \rangle = \mathbf{SFPD}(\langle U_i \rangle, \langle W_i \rangle)$;
end for Initialize $\langle S \rangle = \langle 0 \rangle$;
for $i = 0$ to $m-1$ **do**
  $\langle S \rangle = \mathbf{SFPA}(\langle S \rangle, \langle sc_i \rangle)$;
end for
$\langle sc \rangle = \mathbf{SFPM}(\langle sc \rangle, \langle \frac{1}{m} \rangle)$;
return $\langle sc \rangle$;

1) *The challenger runs* $\mathbf{KeyGen}(1^k)$ *to obtain a public and private key pair* (pk, sk) *and splits the private key* sk *into two parts* sk_1 *and* sk_2*. Then the challenger sends the public key* pk *as well as one part of the secret key, e.g. sk1 to the adversary* $\mathcal{A}$*.*
2) *The adversary* $\mathcal{A}$ *chooses two equal-length messages* m_0 *and* m_1*. Then the adversary* $\mathcal{A}$ *sends them to the challenger.*
3) *The challenger chooses a random bit* $b \in \{0, 1\}$ *and sends the ciphertext* $c^* = \mathbf{Enc}(m_b)$ *to* $\mathcal{A}$*.*
4) *The adversary* $\mathcal{A}$ *outputs a bit* b' *as a guess of* b*.*

The adversary's advantage in the above experiment is defined as $Adv_{\mathcal{E}}(k) := |Pr[b' = b] - \frac{1}{2}|$.

Theorem 1 ([5]): PCPD satisfies Semantic Security, i.e., $Adv_{PCPD}(k) := |Pr[b' = b] - \frac{1}{2}| < \frac{1}{2^k}$.

Here, we discuss the security model for securely implementing an ideal functionality in the presence of non-colluding semi-honest adversaries. For simplicity, we operate according to the specific scenario of our functionality, which involves three parties, challenger DO (i.e. D_0), CP (i.e. S_1 and CSP i.e. S_2). Therefore, we need to construct three simulators $\mathrm{Sim} = \mathrm{Sim}_{D_0}, \mathrm{Sim}_{S_1}, \mathrm{Sim}_{S_2}$ to against three kinds of adversaries $(\mathcal{A}_{D_0}, \mathcal{A}_{S_1}, \mathcal{A}_{S_2})$ that respectively corrupt D_0, S_1 and S_2.

Theorem 2: The **SCDP** *protocol described in Section IV is to securely calculate the ciphertext result of division on the premise that the ciphertext of the dividend and the plaintext of the divisor are known in the presence of semi-honest (non-colluding) adversaries* $\mathcal{A} = (\mathcal{A}_{D_0}, \mathcal{A}_{S_1}, \mathcal{A}_{S_2})$.

Proof: We now demonstrate how to construct three independent simulators $\mathrm{Sim}_{D_0}, \mathrm{Sim}_{S_1}, \mathrm{Sim}_{S_2}$.

Sim_{D_0} receives x and y as input and simulates $\mathcal{A} = \mathcal{A}_{D_0}$ as follows: it generates a public integer y and ciphertext $[x] = Enc(x)$ of x. Finally, it returns $[x]$ and y to $\mathcal{A}_{D_0}$ and outputs $\mathcal{A}_{D_0}$'s entire view.

The view of $\mathcal{A}_{D_0}$ consists of the encrypted data. And the views of $\mathcal{A}_{D_0}$ in the real and ideal executions are indistinguishable due to the semantic security of PCPD.

Sim_{S_1} simulates $\mathcal{A}_{S_1}$ as follows: First, it generates (fictitious) a public number y and encryption $[x]$ and $[|y|]$ by running $Enc(\cdot)$ on randomly chosen x and above y. Then it uses $[x]$ and $|y|$ as the inputs of $\mathrm{Sim}_{S_1}^{\mathrm{SMOD}}(\cdot, \cdot)$ and generates $[M_1]$. Next, it uses $[|y|]$ as the input of $\mathrm{Sim}_{S_1}^{\mathrm{SINV}}(\cdot)$ and generates $[I_1]$. Then, it calculates $[M_2] = [x] \cdot [M_1]^{N-1}$, uses $[M_2]$ and $[I_1]$ as the inputs of $\mathrm{Sim}_{S_1}^{\mathrm{RSM}}(\cdot, \cdot)$ and generates $[f]$. If $y < 0$, it calculates $[f^*] = [f]^{N-1}$. Otherwise, $[f^*] = [f]$. Finally, Sim_{S_1} sends the encryption $[M_1]$, $[I_1]$, $[M_2]$, $[f]$, $[f^*]$ to $\mathcal{A}_{S_1}$. If $\mathcal{A}_{S_1}$ replies with $\perp$, then Sim_{S_1} returns $\perp$.

Sim_{S_2} is analogous to Sim_{S_1}. □

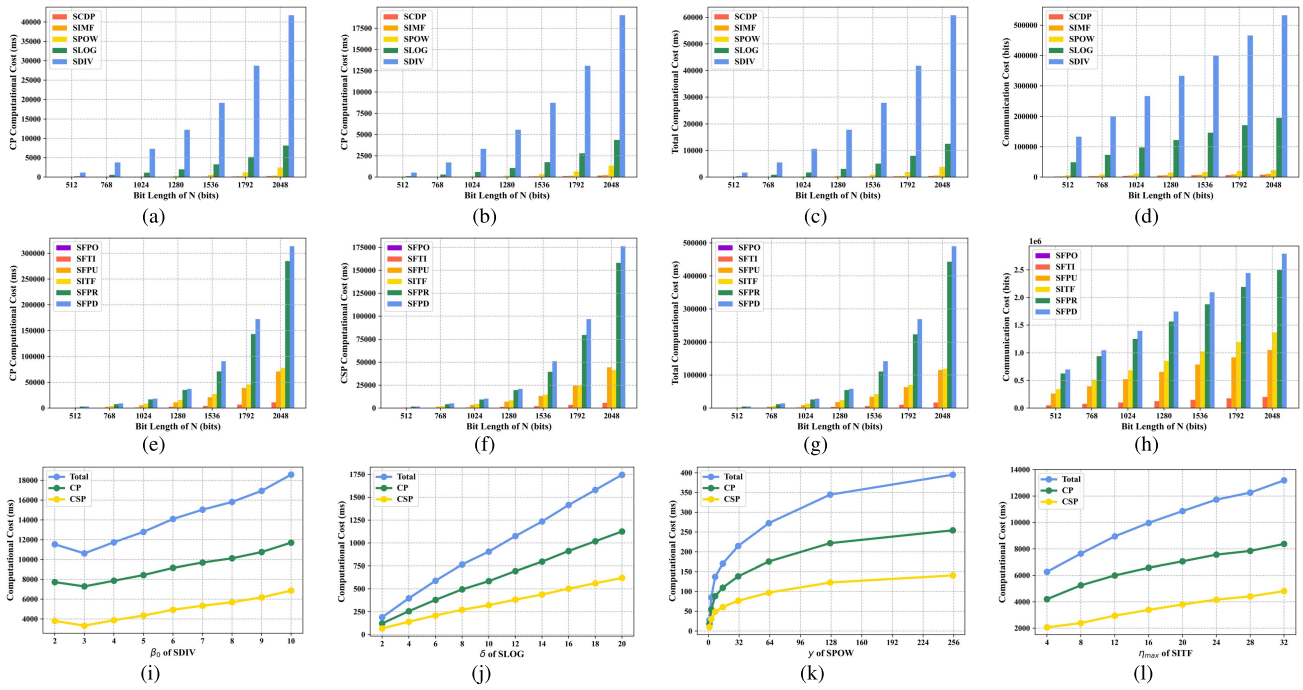


FIGURE 3. Simulation results: (a) Running time on CP (vary with bit length of N). (b) Running time on CSP (vary with bit length of N). (c) Total Running time (vary with bit length of N). (d) Communication cost (vary with bit length of N). (e) Running time on CP (vary with bit length of N, $\eta = 16$). (f) Running time on CSP (vary with bit length of N, $\eta = 16$). (g) Total Running time (vary with bit length of N, $\eta = 16$). (h) Communication cost (vary with bit length of N, $\eta = 16$). (i) Running time of SDIV (vary with base β_0, η_0 is the smallest positive integer that satisfies $\eta_0\beta_0 \geq 10^{15}$). (j) Running time of SLOG (vary with δ). (k) Running time of SPOW (vary with y). (l) Running time of SITF (vary with $\eta_{max} = 32$).

The security proofs of **SDIV**, **SIMF**, **SPOW**, **SLOG** are similar to that of the **SCDP** under the semi-honest (non-colluding) adversaries $\mathcal{A} = (\mathcal{A}_{D_0}, \mathcal{A}_{S_1}, \mathcal{A}_{S_2})$. For the encrypted floating point number calculations (include **SFPR**, **SFPD**, **SFPU**, **SFPO**, **SFTI** and **SITF**), the security relies on the basic encrypted integer calculation (the prove method is similar to that of **SCDP**), which has been proven. Due to the semantic security of PCPD, it is secure for all calculations to be performed in the ciphertext domain.

B. THE SECURITY OF MPOCT

Here, we give an analysis to show that our toolkit MPOCT can resist the system attackers defined in Section 2. The specific analysis is as follows:

- If adversary $\mathcal{A}^*$ eavesdrops on the transmission between DO and CP, then $\mathcal{A}^*$ can obtain the original encrypted data and the final result. In addition, adversary $\mathcal{A}^*$ can obtain the encrypted result transmitted between CP and CSP through eavesdropping. However, these data are encrypted during transmission. Based on the semantic security of the PCPD cryptosystem, adversary $\mathcal{A}^*$ will not be able to decrypt the ciphertext without knowing the private key of DO. Since the public key and private key in the system are distributed securely to each participant by KGC, our system model will not be affected by Man-in-the-middle attack.
- Suppose adversary $\mathcal{A}^*$ has compromised the CP (or CSP) to obtain the challenge RU's partially private key. As the private key is randomly split by executing **KeyS** algorithm of PCPD, adversary $\mathcal{A}^*$ is unable to recover the private key of the challenger DO to decrypt the ciphertext. In addition, adversary $\mathcal{A}^*$ cannot obtain useful information even if the CSP is compromised, because our protocols use the known technique of "blinding" the plaintext [29]: given the ciphertext of the message, we use the additive homomorphism of the PCPD cryptosystem to add random messages to it. Therefore, the original plaintext becomes blinded.
- If adversary $\mathcal{A}^*$ has a private key belonging to another DO (i.e. which is not the private key of challenger DO). Since the private keys of different DOs in our system are irrelevant, adversary $\mathcal{A}^*$ is still unable to decrypt the ciphertext of challenger DO.

VII. EXPERIMENTAL EVALUATION

In this section, we first evaluate the performance of protocols in MPOCT. Then we analyze the performance of SKM and SSC.

A. EXPERIMENTS OF MPOCT

First, we let N be 1024 bits to achieve 80-bit security [30]. The computation cost and communication overhead of the proposed MPOCT are evaluated using a Python program, and the experiments are performed on a single server with 2.3GHz one-core processor and 8GB RAM memory. The performances of protocols for both integers and FPNs in

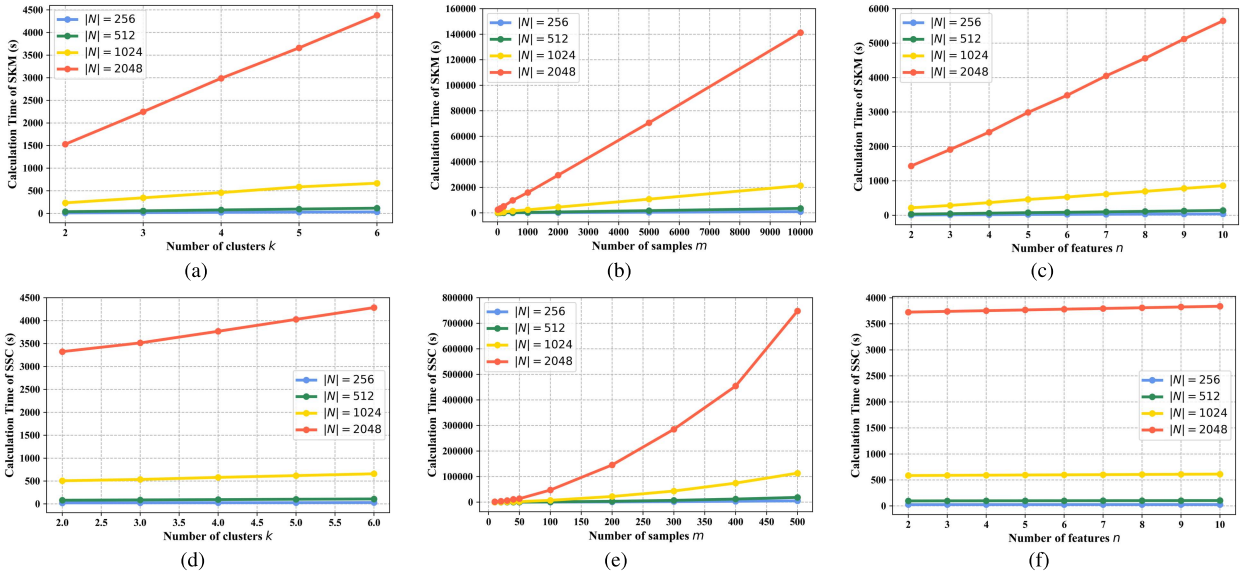


FIGURE 4. Performance of SKM & SSC. (a) Calculation time of SKM per iteration(vary with number of clusters k, $m = 50$, $n = 5$). (b) Calculation time of SKM per iteration (vary with number of samples m, $k = 4$, $n = 5$). (c) Calculation time of SKM per iteration (vary with number of features n, $k = 4$, $m = 50$). (d) Calculation time of SSC (vary with number of clusters k, $m = 20$, $n = 5$). (e) Calculation time of SSC (vary with number of samples m, $k = 4$, $n = 5$). (f) Calculation time of SSC (vary with number of features n, $k = 4$, $m = 20$).

TABLE 2. The Performance of protocols for Integer (80-bit security level).

Protocol	CP compute.	CSP compute.	Total compute.	Commu.
RSM	13.927 ms	7.698 ms	21.625 ms	1.256 KB
SLT	7.550 ms	5.202 ms	12.752 ms	1.002 KB
SXOR	32.278 ms	16.570 ms	48.848 ms	2.510 KB
SEQ	49.091 ms	26.850 ms	75.941 ms	4.514 KB
SEXP	5.461 ms	5.341 ms	10.802 ms	0.750 KB
SINV	3.863 ms	6.293 ms	10.156 ms	0.750 KB
SMOD	16.376 ms	10.244 ms	26.620 ms	1.752 KB
SCDP	34.795 ms	24.678 ms	59.473 ms	3.758 KB
SIMF	49.089 ms	32.596 ms	81.685 ms	5.014 KB
SPOW	138.281 ms	76.443 ms	214.724 ms	11.304 KB
SLOG	1.136 s	0.610 s	1.746 s	95.360 KB
SDIV	7.294 s	3.324 s	10.618 s	260.226 KB

TABLE 3. The Performance of protocols for FPNs (80-bit security level).

Protocol	CP compute.	CSP compute.	Total compute.	Commu.
SFPA	5.542 s	3.470 s	9.012 s	513.285 KB
SFPM	1.497 s	0.957 s	2.454 s	140.731 KB
SFPC	1.623 s	0.867 s	2.490 s	136.893 KB
SFPS	1.103 s	0.601 s	1.704 s	98.010 KB
FPEQ	0.163 s	0.085 s	0.248 s	14.329 KB
SFPO	3.60 ms	0 ms	3.60 ms	0 KB
SFPU	5.548 s	3.470 s	9.018 s	513.285 KB
SFPR	16.770 s	9.312 s	26.082 s	1.194 MB
SFPD	18.270 s	10.266 s	28.536 s	1.331 MB
SFTI	1.368 s	0.702 s	2.070 s	97.888 KB
SITF	8.609 s	4.579 s	13.188 s	667.670 KB

MPOCT are respectively shown in Table 2 and Table 3, while some protocols are evaluated under specific parameters: **SPOW**($y = 31$), **SLOG**($\delta = 20$), **SDIV**($\beta_0 = 3$, $\eta_0 = 32$), **SITF**($\eta_{max} = 32$). We can find that the protocols of integer ciphertext is faster in computing time and lower in communication overhead than protocols of FPNs. Then, we discuss the factors that affect the performance of these protocols. From Figure 3(a)-(h), we can easily find that both the running time and communication overhead of the protocols increase with N. This is because the running time required for basic operations (modular multiplication and exponential) increases while N increases. Meanwhile, more bits need to be transmitted. In addition, the running time of some protocols will also change according to specific parameters, which is shown in Figure 3(i)-(l). We find that SDIV has the highest computational efficiency, when β_0 is equal to 3. In addition, the calculation efficiency of SLOG changes linearly with δ and the calculation efficiency of SITF also changes linearly with η_{max}. Since SPOW is based on the idea of Binary Exponentiation, the calculation efficiency only changes logarithmically with y.

B. EXPERIMENTS OF SKM AND SSC

1) COMPUTATIONAL EFFICIENCY ANALYSIS

According to the algorithm process of SKM and SSC, we found that the main factors affecting the computational efficiency are the bit size of the public key $|N|$, the number of clusters k, the number of samples m and the number of features n. From Figure 4(a)(b)(c), we find that the calculation time of SKM is linearly related to k, m and n. From Figure 4(d)(e)(f), we find that the calculation time of SSC is non-linearly related to m, but it is linearly related to k and n. Since SSC needs to calculate the distance between paired samples, the calculation time of SSC is related to m^2. Therefore, the calculation time of SSC will increase significantly when the number of samples m increases. In addition, we have

TABLE 4. Dataset statistic.

Dataset	Samples	Attributes	Classes
Iris[1]	150	4	3
Wine[2]	178	13	3
Breast Cancer[3]	569	30	2

TABLE 5. Classification accuracy on public datasets.

Dataset	K-means	SKM	DP-K-means
Iris	0.853	0.833	0.813
Wine	0.972	0.966	0.893
Breast Cancer	0.912	0.910	0.772

also compared with the previous scheme PPODC [7] and PPCOM [9]. We fix the bit size N of public key in our Paillier cryptosystem to be 1024. When $m = 10000, n = 10, k = 4$, the running time of PPODC on Python is 13360 minutes per iteration and 1856 minutes on PPCOM, while our solution SKM only needs 577 minutes, which is more than 20 times faster than PPODC and 3 times faster than PPCOM.

2) PERFORMANCE ANALYSIS

To demonstrate the effectiveness of SKM and SSC, we conduct extensive experiments over three public datasets. The dataset description is shown in Table 4.

First, we compare the performance of three different K-means algorithms, i.e. SKM, K-means on plaintext, K-means using differential privacy(DP-K-means [22]). Parameter configuration: the number of iterations E is set as 10. The privacy-preserving budget of DP-K-means ϵ is set as 1. Since the datasets have provided label information, we conduct our experiments in two aspects: with and without labels.

Table 5 shows the classification accuracy of three methods with labels. We can find that the K-means on plaintext outperforms the others with the highest accuracy. Our scheme SKM achieves the second place, and it is competitive with the performance of K-means on plaintext. This is because SKM requires a given fixed number of iterations E and thus may not fully converge. To ensure convergence, it is only necessary to set a larger number of iterations. However, the computational cost increases linearly as the number of iterations E increases. Therefore, in order to balance the computational efficiency and the usability of the algorithm, we choose a small number of iterations E for experiments. DP-K-means performs relatively inferior due to the added noise in data.

Since ground truth labels are hard to obtain in a real application scenario, we use silhouette coefficient for evaluation. As an effective index to evaluate the performance of clustering, the silhouette coefficient ranges from -1 to 1. The closer the it is to 1, the better the clustering performance is; on the contrary, the closer the silhouette coefficient is to -1, the worse the clustering performance is. Table 6 shows the silhouette coefficient of three methods. The situation is similar to those with labels. K-means on plaintext surpasses the others with the highest silhouette coefficient. The performance

TABLE 6. Silhouette coefficient on public datasets.

Dataset	K-means	SKM	DP-K-means
Iris	0.495	0.480	0.457
Wine	0.285	0.284	0.255
Breast Cancer	0.347	0.345	0.289

TABLE 7. Performance of secure silhouette coefficient.

Dataset	SC	SSC
Iris	0.480	0.480
Wine	0.284	0.284
Breast Cancer	0.345	0.345

of our scheme SKM is competitive with that on K-means. DP-K-means, however, performs relatively inferior for the same reason above.

Last, we validate the effectiveness of secure silhouette coefficient. Based on the clustering results obtained by SKM, we evaluate the performance of silhouette coefficient on the decrypted plaintext(SC) and secure silhouette coefficient on the ciphertext(SSC). We can observe from Table 7 that the proposed SSC in this paper can ensure the same availability as that on the plaintext.

VIII. CONCLUSION AND FUTURE WORK

In this paper, a novel multifunctional and privacy-preserving outsourcing computation tookit (**MPOCT**) is proposed to support several homomorphic computing protocols including division and power on ciphertext of integers and floating point numbers. Concretely, we first extend several homomorphic operations on the ciphertext of floating-point numbers based on the previous framework **POCF**. However, due to the low efficiency and high communication overhead on the ciphertext of floating-point numbers, we further extend the secure outsourcing computation protocols on the ciphertext of integers. After that, homomorphic mutual conversion protocols between integer and floating-point ciphertext are proposed to balance the efficiency and feasibility of computation. Next, we implement a homomorphic K-means algorithm based on **MPOCT** for clustering and design the homomorphic silhouette coefficient as the evaluation index, providing an informative cluster assessment for local users with limited resources. Comprehensive experimental results and security analysis have proved the proposed **MPOCT** can achieve efficiency and utility without privacy leakage to unauthorized parties.

In the future, **MPOCT** is expected to resort to homomorphic neural network modules such as homomorphic convolution and homomorphic pooling. In addition, **MPOCT** can only be applied to the algorithms with certain termination conditions(i.e. the homomorphism of the while loop cannot be implemented) since the servers cannot be aware of the comparison results of ciphertext. An improved scheme of solving this problem is to be explored in the future.

REFERENCES

[1] C. Butpheng, K.-H. Yeh, and H. Xiong, "Security and privacy in IoT-cloud-based e-health systems—A comprehensive review," *Symmetry*, vol. 12, no. 7, p. 1191, Jul. 2020.

[2] 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.

[3] Z. A. Almusaylim and N. Z. Jhanjhi, "Comprehensive review: Privacy protection of user in location-aware services of mobile cloud computing," *Wireless Pers. Commun.*, vol. 111, pp. 541–564, Oct. 2019.

[4] Y. Yang, X. Huang, X. Liu, H. Cheng, J. Weng, X. Luo, and V. Chang, "A comprehensive survey on secure outsourced computation and its applications," *IEEE Access*, vol. 7, pp. 159426–159465, 2019.

[5] X. Liu, R. H. Deng, W. Ding, R. Lu, and B. Qin, "Privacy-preserving outsourced calculation on floating point numbers," *IEEE Trans. Inf. Forensics Security*, vol. 11, no. 11, pp. 2513–2527, Nov. 2016.

[6] X. Liu, K.-K. R. Choo, R. H. Deng, R. Lu, and J. Weng, "Efficient and privacy-preserving outsourced calculation of rational numbers," *IEEE Trans. Depend. Sec. Comput.*, vol. 15, no. 1, pp. 27–39, Feb. 2018.

[7] F.-Y. Rao, B. K. Samanthula, E. Bertino, X. Yi, and D. Liu, "Privacy-preserving and outsourced multi-user K-Means clustering," in *Proc. IEEE Conf. Collaboration Internet Comput. (CIC)*, Hangzhou, China, Oct. 2015, pp. 80–89.

[8] N. Almutairi, F. Coenen, and K. Dures, "K-means clustering using homomorphic encryption and an updatable distance matrix: Secure third party data clustering with limited data owner interaction," in *Proc. Int. Conf. Big Data Anal. Knowl. Discovery*, Lyon, France, Aug. 2017, pp. 274–285.

[9] H. Rong, H. Wang, J. Liu, J. Hao, and M. Xian, "Privacy-preserving K-means clustering under multiowner setting in distributed cloud environments," *Secur. Commun. Netw.*, vol. 2017, Jan. 2017, Art. no. 3910126.

[10] P. Sun, "Security and privacy protection in cloud computing: Discussions and challenges," *J. Netw. Comput. Appl.*, vol. 160, Jun. 2020, Art. no. 102642.

[11] J. Domingo-Ferrer, O. Farràs, J. Ribes-González, and D. Sánchez, "Privacy-preserving cloud computing on sensitive data: A survey of methods, products and challenges," *Comput. Commun.*, vols. 140–141, pp. 38–60, May 2019.

[12] P. Paillier, "Public-key cryptosystems based on composite degree residuosity classes," in *Proc. Int. Conf. Theory Appl. Cryptograph. Techn.* Berlin, Germany, Springer, 1999, pp. 223–238.

[13] T. ElGamal, "A public key cryptosystem and a signature scheme based on discrete logarithms," *IEEE Trans. Inf. Theory*, vol. IT-31, no. 4, pp. 469–472, Jul. 1985.

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

[15] Z. Brakerski, C. Gentry, and V. Vaikuntanathan, "(Leveled) fully homomorphic encryption without bootstrapping," *ACM Trans. Comput. Theory*, vol. 6, no. 3, pp. 1–36, 2014.

[16] J. Fan and F. Vercauteren, "Somewhat practical fully homomorphic encryption," *IACR Cryptol. ePrint Arch.*, vol. 2012, p. 144, Mar. 2012.

[17] J. H. Cheon *et al.*, "Homomorphic encryption for arithmetic of approximate numbers," in *Proc. Int. Conf. Theory Appl. Cryptol. Inf. Secur.* Cham, Switzerland, Springer, 2017, pp. 409–437.

[18] Y. Liu, Y. Luo, Y. Zhu, Y. Liu, and X. Li, "Secure multi-label data classification in cloud by additionally homomorphic encryption," *Inf. Sci.*, vol. 468, pp. 89–102, Nov. 2018.

[19] X. Liu, R. H. Deng, K.-K.-R. Choo, Y. Yang, and H. Pang, "Privacy-preserving outsourced calculation toolkit in the cloud," *IEEE Trans. Dependable Secure Comput.*, vol. 17, no. 5, pp. 898–911, Sep. 2020.

[20] J. Li, X. Kuang, S. Lin, X. Ma, and Y. Tang, "Privacy preservation for machine learning training and classification based on homomorphic encryption schemes," *Inf. Sci.*, vol. 526, pp. 166–179, Jul. 2020.

[21] H. Zong, H. Huang, and S. Wang, "Secure outsourced computation of matrix determinant based on fully homomorphic encryption," *IEEE Access*, vol. 9, pp. 22651–22661, 2021.

[22] C. Xia, J. Hua, W. Tong, and S. Zhong, "Distributed K-Means clustering guaranteeing local differential privacy," *Comput. Secur.*, vol. 90, Mar. 2020, Art. no. 101699.

[23] P. Mohassel, M. Rosulek, and N. Trieu, "Practical privacy-preserving K-means clustering," *Proc. Privacy Enhancing Technol.*, vol. 2020, no. 4, pp. 414–433, Oct. 2020.

[24] Y. Zou, Z. Zhao, S. Shi, L. Wang, Y. Peng, Y. Ping, and B. Wang, "Highly secure privacy-preserving outsourced k-Means clustering under multiple keys in cloud computing," *Secur. Commun. Netw.*, vol. 2020, pp. 1–11, Mar. 2020.

[25] S. Z. Selim and M. A. Ismail, "K-means-type algorithms: A generalized convergence theorem and characterization of local optimality," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. PAMI-6, no. 1, pp. 81–87, Jan. 1984.

[26] T. Wang, Y. Mei, W. Jia, X. Zheng, G. Wang, and M. Xie, "Edge-based differential privacy computing for sensor–cloud systems," *J. Parallel Distrib. Comput.*, vol. 136, pp. 75–85, Feb. 2020.

[27] Q. Do, B. Martini, and K.-K.-R. Choo, "A forensically sound adversary model for mobile devices," *PLoS ONE*, vol. 10, no. 9, Sep. 2015, Art. no. e0138449.

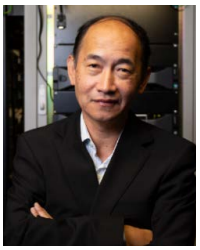
[28] P. J. Rousseeuw, "Silhouettes: A graphical aid to the interpretation and validation of cluster analysis," *J. Comput. Appl. Math.*, vol. 20, no. 1, pp. 53–65, Jan. 1987.

[29] A. Peter, E. Tews, and S. Katzenbeisser, "Efficiently outsourcing multiparty computation under multiple keys," *IEEE Trans. Inf. Forensics Security*, vol. 8, no. 12, pp. 2046–2058, Dec. 2013.

[30] D. Asiedu and A.-M. Salifu, "Security evaluation of pailiar homormophic encryption scheme," *Asian J. Res. Comput. Sci.*, vol. 2020, pp. 12–17, Nov. 2020.

JIALIN LI received the bachelor's degree in software engineering from Jilin University, in 2019. She is currently pursuing the master's degree in software engineering from East China Normal University. Her research interests include semantic segmentation, graph data processing, and privacy-preserving related topics.

PENGHAO LU received the bachelor's degree in science from the East China University of Science and Technology. He is currently pursuing the master's degree in software engineering from East China Normal University, Shanghai, China. His research interests mainly include machine learning and privacy-preserving.



XUEMIN LIN (Fellow, IEEE) is a Distinguished Professor (Scientia) and the Head of the Database and Knowledge Research Group, School of Computer Science and Engineering, UNSW. He is a Concurrent Professor with the School of Software, East China Normal University, a Distinguished Visiting Professor with Tsinghua University, and the Visiting Chair Professor with Fudan University. His research interests include scalable processing and mining of large-scale data, including graph, spatial-temporal, streaming, and text and uncertain data. He was an Associate Editor of *ACM Transactions Database Systems*, from 2008 to 2014, and IEEE TRANSACTIONS ON KNOWLEDGE AND DATA ENGINEERING, from 2013 to 2015, and an Associate Editor-in-Chief of IEEE TRANSACTIONS ON KNOWLEDGE AND DATA ENGINEERING, from 2015 to 2016. Currently, he has been the Editor-in-Chief of IEEE TRANSACTIONS ON KNOWLEDGE AND DATA ENGINEERING, since 2017.