Efficient Clustering on Encrypted Data

Mengyu Zhang[1], Long Wang[1,2], Xiaoping Zhang[3(✉)], Zhuotao Liu[1,2], Yisong Wang[3], and Han Bao[3]

[1] Institute for Network Sciences and Cyberspace, Tsinghua University, Beijing, China
mengyu-z23@mails.tsinghua.edu.cn, {longwang,zhuotaoliu}@tsinghua.edu.cn
[2] Zhongguancun Laboratory, Beijing, China
[3] Department of Computer Science and Technology, Tsinghua University, Beijing, China
zhxp@tsinghua.edu.cn, {wys19,baoh17}@mails.tsinghua.edu.cn

Abstract. Clustering is a significant unsupervised machine learning task widely used for data mining and analysis. Fully homomorphic encryption allows data owners to outsource privacy-preserving computations without interaction. In this paper, we propose a fully privacy-preserving, effective, and efficient clustering scheme based on CKKS, in which we construct two iterative formulas to solve the challenging ciphertext comparison and division problems, respectively. Although our scheme already outperforms existing work, executing it on datasets MNIST and CIFAR-10 still results in unacceptable run time and memory consumption. To further address the above issues, we propose a block privacy-preserving clustering algorithm that splits records into subvectors and clusters these subvectors. Experimental results show that the clustering accuracy of our original algorithm is almost equivalent to the classical k-means algorithm. Compared to a state-of-the-art FHE-based scheme, our original algorithm not only outperforms theirs in accuracy but is also 4 orders of magnitude faster than theirs. In experiments testing our block algorithm, we conclude that the run time and memory consumption of this algorithm are greatly reduced.

Keywords: Fully Homomorphic Encryption · clustering · privacy-preserving

1 Introduction

Clustering is a significant tool for data mining and data analysis. It reveals intrinsic patterns in many real-world problems and is widely used in many fields [4,23]. In many scenarios, different dimensions of the same records are held by two parties respectively, that is, the dataset is split vertically. For example, a bank wants to perform cluster analysis on customers, but the customer clustering based on the customer information held by the bank is not accurate enough. The

This research is partially supported by Zhongguancun Laboratory.

C. Pöpper and L. Batina (Eds.): ACNS 2024, LNCS 14583, pp. 213–236, 2024.
https://doi.org/10.1007/978-3-031-54770-6_9

government holds private information such as the income status of citizens, which can help banks obtain more accurate clustering results. Note that government data is generally stored on specific servers and cannot be transmitted to any organization or individual in any form. Additionally, these two organizations typically communicate over a WAN, with no guarantees of latency or bandwidth. Therefore, to obtain more accurate results, the bank (the data owner) needs to send the data to the government (the server), and the government (the server) performs clustering on the joint dataset and returns the result. However, sending customer information to the government in plaintexts can lead to the disclosure of private information. Thence, it is necessary to propose a privacy-preserving and efficient clustering scheme.

Fully Homomorphic Encryption (FHE) is a cryptographic primitive that protects private information contained in data by performing homomorphic operations on encrypted data. FHE is more suitable for scenarios with high latency as the FHE-based algorithms allow non-interactive setting. However, the previous FHE-based privacy-preserving clustering works [3,11,18] leak partial private information to servers during the calculation. To avoid the disclosure of private information during the calculation, Jäschke and Armknecht [13] propose a fully privacy-preserving clustering scheme based on FHE. However, their approximate ciphertext division introduces errors, and their expensive bitwise ciphertext comparison leads to impractical run times (about 6 days for 400 records).

To solve the above problems, in this paper, we propose a fully privacy-preserving, effective, and efficient clustering scheme based on FHE. We choose the approximate homomorphic encryption scheme CKKS [7] as our encryption algorithm, which performs approximate arithmetic and is able to encode floating point numbers. It is a challenge that CKKS does not directly support ciphertext comparison and division used in the clustering algorithm. To achieve ciphertext comparison, we construct an iterative formula with two fixed points of opposite signs to approximate the sign function. For ciphertext division, we construct a function whose root is the reciprocal of the divisor, and apply Newton's method to approximate the root of this function. After making minor changes to k-means clustering algorithm, we propose a fully privacy-preserving and efficient clustering algorithm that prevents the server from inferring private information.

Although, our fully privacy-preserving clustering algorithm has outperformed existing schemes, executing it on some well-known large datasets (e.g. MNIST, CIFAR-10, etc.) leads to unacceptable run time and memory consumption (about 3500 minutes and 1303GB memory on MNIST). Through theoretical analysis, the number of clusters k is an important factor affecting the run time and memory consumption, therefore, reducing k can greatly increase the practicability of our algorithm. To reduce k, we split the input vectors into disjoint subvectors and cluster these subvectors. We refer to this optimized clustering algorithm as the block privacy-preserving clustering algorithm. In addition, this optimized scheme also reduces the number of consecutive multiplications, which is limited by a predetermined parameter since CKKS is a leveled homomorphic encryption.

We first test our fully privacy-preserving clustering scheme on several popular small datasets to assess clustering accuracy and efficiency. In terms of

accuracy, the results show that our method performs as well as the original k-means clustering algorithm and significantly outperforms the state-of-the-art privacy-preserving clustering work [13]. As for efficiency, we adopt two implementation optimization techniques, one batching, and the other multithreading. The results show that these two optimizations reduce the run time by three orders of magnitude in total, making our scheme more efficient. Then, we compare the efficiency of our scheme with state-of-the-art [13]. We use our fully privacy-preserving clustering algorithm with batching, which achieves nearly identical accuracy as the original (plaintext) k-means clustering algorithm. Compared with the FHE-based scheme [13], our method is about 19573x faster. In summary, our fully privacy-preserving clustering scheme is efficient and effective compared to existing works.

Afterwards, we test our block privacy-preserving clustering algorithm on two large datasets (MNIST, CIFAR-10) on which our vanilla fully privacy-preserving clustering algorithm fails to run using our server due to the long run time and excessive memory consumption. As can be seen from the results, our block clustering solution can be performed on these datasets with acceptable run time and memory consumption. In addition, this block clustering algorithm is also suitable for users who would like to trade a little accuracy for high efficiency.

- We construct two iterative formulas to solve the ciphertext comparison and division problems respectively, and propose a fully privacy-preserving clustering scheme based on FHE, which can ensure that no private information is revealed during the calculation.
- To further reduce the run time and memory consumption of our algorithm, we propose the block privacy-preserving clustering algorithm. This algorithm divides the input vectors into disjoint subvectors and performs clustering on these subvectors.
- We conduct a series of experiments on various widely used datasets to evaluate our scheme. From the results, it can be concluded that our block clustering solution performs well in terms of both efficiency and memory consumption.

2 Related Works

In recent years, privacy-preserving clustering has been widely discussed in the literature. Some literature [27,29,30,36] leverage Partially Homomorphic Encryption (PHE) and two non-colluding clouds and design an interactive protocol between them. However, these schemes are not suitable for large data scenarios due to the high communication cost. Some previous works [5,12,26,31,32] apply differential privacy to protect the private information of individuals. However, the accuracy drops significantly due to the noise introduced in the clustering algorithm. Multiparty Computation (MPC) is proposed to implement privacy-preserving clustering in [24,35], which requires interaction between data owners.

Focusing on FHE, Liu et al. [18] propose an outsourced clustering scheme. Almutairi et al. [3] create an updatable distance matrix (UDM) and apply it to

improve the efficiency of their clustering algorithm. Gheid et al. [11] present a clustering protocol that does not use any cryptographic scheme. However, these works leak intermediate values which contain private information, such as the sizes of clusters or the distances between records and cluster centers. Wu et al. [37] propose an efficient and privacy-preserving outsourced k-means clustering scheme based on Yet Another Somewhat Homomorphic Encryption (YASHE). There are two non-colluding cloud servers in their model, one of which is used for addition and multiplication in ciphertexts, and the other is responsible for comparison in plaintexts. Therefore, the server that computes in plaintexts can infer partially private information from the data. Different from these works, our clustering scheme is fully privacy-preserving, which allows data owners to outsource clustering without revealing any private information.

Jäschke and Armknecht present a completely privacy-preserving clustering scheme based on FHE in [13]. Due to the impractical run time of their algorithm, they trade accuracy for efficiency. In other words, they create an approximate comparison scheme by truncating the few least significant bits in ciphertexts. However, the run time of the dataset with 400 records is still 25.79 days, which is far from practical for large datasets. Different from their bitwise approximate comparison scheme, our ciphertext comparison is more efficient through an iterative approach. Also, their ciphertext division is approximate and introduces errors into their clustering algorithm, whereas our ciphertext division is exact.

3 Background

3.1 Approximate Homomorphic Encryption CKKS

To achieve privacy-preserving clustering, we choose the approximate homomorphic encryption algorithm CKKS [7] as our cryptographic scheme. We use CKKS instead of other RLWE-based schemes for two main reasons, one is its ability to encode floating point numbers, and the other is its high efficiency.

Cheon et al. propose the formal definitions of CKKS in [7]. We define that N is a power of two and $R = \mathbb{Z}[X]/(X^N + 1)$ is a ring of integers, where $\mathbb{Z}[X]$ is a polynomial ring with integer coefficients. A message $\mathbf{m} \in \mathbb{C}^{N/2}$ is firstly encoded into a polynomial $m \in R$ called plaintext. Let L be a level parameter and $q_l = 2^l$ for $1 \leq l \leq L$. We denote $R_q = \mathbb{Z}_q[X]/(X^N + 1)$ as a modulo-q quotient ring of R. The modulo q operation on each element of R_q is denoted as $[\cdot]_q$. The distribution $X_\mathrm{s} = HWT(h)$ outputs a polynomial with coefficients $\{-1, 0, 1\}$, whose Hamming weight is h. The distributions X_enc and X_e are the discrete Gaussian distribution.

Set the secret key to be $sk \leftarrow (1, s)$ and the public key to be $pk \leftarrow ([-a \cdot s + e]_{q_L}, a)$, where $s \leftarrow X_\mathrm{s}$, $a \leftarrow U(R_{q_L})$, and $e \leftarrow X_\mathrm{e}$. To encrypt the plaintext m, we can compute $c = [v \cdot pk + (m + e_0, e_1)]_{q_L}$, where $v \leftarrow X_\mathrm{enc}$ and $e_0, e_1 \leftarrow X_\mathrm{e}$. The ciphertext $c = (c_0, c_1)$ is decrypted by calculating $m' = [c_0 + c_1 \cdot s]_{q_l}$.

CKKS is a leveled homomorphic encryption, which means that this homomorphic encryption scheme limits the number of consecutive multiplications by

a pre-determined parameter. Cheon et al. [6] extend CKKS to a fully homomorphic encryption scheme by proposing a bootstrapping algorithm that can refresh low-level ciphertexts. The input of the bootstrapping algorithm is a ciphertext with the lowest modulus $c \in R_{q_1}^2$ and the output is a ciphertext with a refreshed modulus $c' \in R_{q_{L'}}^2$, where $L' < L$. And the results of decrypting c and c' are almost equal.

3.2 Newton's Method

Newton's method is an approximate root-finding algorithm. This method can solve equations of the form $f(x) = 0$, where f is a real-valued function and the derivative of f is denoted by f'.

Suppose $x_0 \in [a, b]$ is an initial guess for a root of f, denoted as r. We find the line tangent to $f(x)$ at $x = x_0$ and compute the intersection of the x-axis with this tangent, denoted by $(x_1, 0)$. $x_1 = x_0 - \frac{f(x_0)}{f'(x_0)}$, where x_1 is a better approximation of r than x_0. Repeat the above steps to continuously refine the approximation, and x_{n+1} can be computed as follows [10]:

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} \tag{1}$$

If Newton's method converges on $[a, b]$, a sufficiently accurate value can be reached when the number of iterations is large enough [25]. That is $\lim_{n \to +\infty} x_n = r$.

4 System Architecture and Threat Model

4.1 System Architecture

In our system architecture, there are two actors (server and data owner). The dataset is split vertically, with the server and the data owner holding different dimensions of the same records. The joint dataset, denoted D, has n records of m dimensions. We assume that the dataset D_1 held by the data owner contains n records of m_1 dimensions, and the dataset D_2 owned by the server contains the same n records of m_2 dimensions, where $m = m_1 + m_2$. Since the private set intersection technique has been proposed in previous work, we can suppose that D_1 and D_2 are already aligned. Therefore, $D = D_1 || D_2$.

In the scenario we describe, to get better clustering results, the data owner has to send D_1 to the server, which executes the clustering algorithm on the joint dataset D. The server is untrusted, which means it tries to infer private information from the data, thus the data owner needs to outsource the clustering to the server in ciphertexts (Fig. 1).

Before exploiting the outsourced clustering service, the data owner needs to generate the secret key sk kept private by itself as well as the public key pk and the evaluation key evk both shared with the server. Then, the data owner encodes n records into plaintexts and encrypts these plaintexts into ciphertexts.

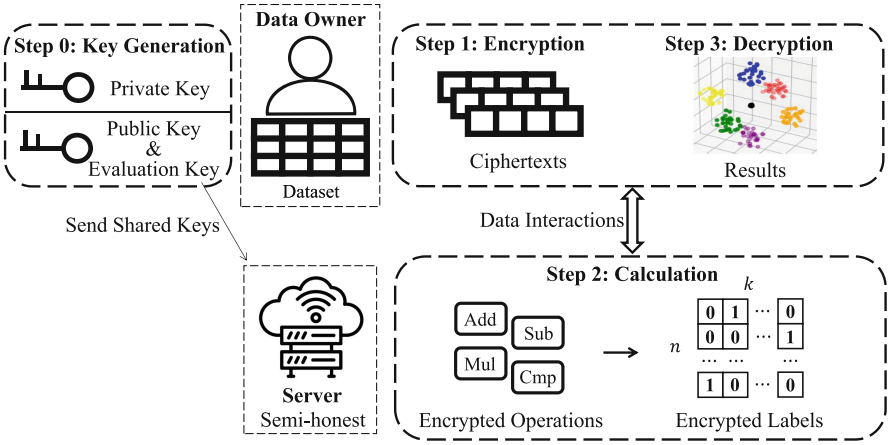

Fig. 1. Our System Architecture

After that, the data owner sends these ciphertexts and some parameters to the server. Among them, the parameters include m_1, the number of clusters denoted by k, and the number of iterations for clustering denoted by T. After receiving these ciphertexts and parameters, the server executes k-means clustering in ciphertexts and returns the ciphertext results to the data owner. Finally, the data owner decrypts the results and obtains the labels of all records through a simple calculation.

4.2 Threat Model

In this paper, we consider a semi-honest security model, which means that the server and the user strictly follow the protocol, but they may try to infer private information about the datasets D_1 and D_2. We assume that m_1, k, T are the public system parameters known by both parties: m_1 only represents the dimensions of the records owned by the data owner, and k, and T are the parameters for clustering. Furthermore, we consider the communication channel to be insecure, which means that eavesdropping attacks are possible.

In our protocol, D_1 is homomorphically encrypted locally and transmitted to the server as ciphertexts for clustering. Therefore, an adversary eavesdropping the communication channel or the server cannot deduce private information from the encrypted D_1. In addition, all records in D_2 are not sent to the user in any form, so the user cannot infer any private information from D_2.

Baiyu and Micciancio [16] point out that CKKS does not satisfy indistinguishability under chosen plaintext attacks with decryption oracles (IND-CPA$^{\mathrm{D}}$) security. However, chosen plaintext attacks with decryption oracles cannot occur in our system, since (1) only the data owner can choose the plaintexts to be encrypted; (2) no ciphertext and the corresponding plaintext are sent to the

server simultaneously; (3) the data owner never sends the decryption results to the server.

4.3 Security

In this paper, our goal is to achieve a simulation-based security definition.

Definition. *A protocol* Π *between a data owner having as input a dataset* D_1 *and a server having as input a dataset* D_2 *is a cryptographic clustering protocol if it satisfies the following guarantees:*

- ***Correctness.*** *On each input dataset* D_1 *of the data owner and each dataset* D_2 *held by the server, the output of the data owner at the end of the protocol is the correct clustering result on the joint dataset.*
- ***Security:***
  - *We require that a corrupted, semi-honest data owner does not learn anything about the input* D_2 *of the server. Formally, we require the existence of an efficient simulator* $\mathtt{Sim}_C$ *such that* $\mathtt{View}_C^{\Pi} \approx_c \mathtt{Sim}_C(D_1, \mathtt{out})$*, where* $\mathtt{View}_C^{\Pi}$ *denotes the view of the data owner in the execution of* Π *and* $\mathtt{out}$ *denotes the output of the clustering.*
  - *We also require that a corrupted, semi-honest server does not learn anything about the input* D_1 *of the data owner. Formally, we require the existence of an efficient simulator* $\mathtt{Sim}_S$ *such that* $\mathtt{View}_S^{\Pi} \approx_c \mathtt{Sim}_S(D_2)$*, where* $\mathtt{View}_S^{\Pi}$ *denotes the view of the server in the execution of* Π*.*

5 Fully Privacy-Preserving Clustering Scheme Based on FHE

In this section, we propose a fully privacy-preserving clustering scheme based on FHE algorithm CKKS. We choose Lloyd's algorithm to solve the k-means clustering problem. The reasons for choosing Lloyd's algorithm are (1) it is simple to implement; (2) it has effective local fine-tuning capability [9]. However, CKKS cannot directly support division and comparison in ciphertexts, which are used in Lloyd's algorithm. Therefore, we adopt iterative method and Newton's method to solve ciphertext comparison and division respectively. After making minor changes to Lloyd's algorithm, we implement a fully privacy-preserving clustering algorithm.

5.1 Preliminaries

The goal of k-means clustering is to partition the dataset into k sets [21]. Lloyd's algorithm [19] can solve the k-means clustering problem and is one of the most extensively used clustering algorithms for statistical data analysis.

The dataset D held by the data owner contains n records denoted $t_1, \ldots, t_n$. Each record t_i is an m-dimensional real vector, denoted as $t_i = (t_{i1}, \ldots, t_{im}) \in$

Algorithm 1: The Lloyd's algorithm

Input n, m, k, T, $D = \{t_1, \ldots, t_n\}$
Initialize centroids $c_1, \ldots, c_k$ randomly
repeat T **times**
  Calculate distance of each record to each centroid
    $\text{diff}_{ij} \leftarrow \sum_q (t_{iq} - c_{jq})^2$
    $(i = 1, \ldots, n,\ j = 1, \ldots, k,\ q = 1, \ldots, m)$
  Find the closest centroid to each record
    $\text{label}_i \leftarrow j$ with minimal diff_{ij}
    $(i = 1, \ldots, n,\ j = 1, \ldots, k)$
  Count the number of records in each cluster
    $\text{cnt}_j \leftarrow$ the number of i-s where $\text{label}_i = j$
    $(i = 1, \ldots, n,\ j = 1, \ldots, k)$
  Update centroids
    $c_j \leftarrow \frac{\sum_i t_i}{\text{cnt}_j}$ where $\text{label}_i = j$
    $(i = 1, \ldots, n,\ j = 1, \ldots, k)$
Output label and $c_1, \ldots, c_k$

$\mathbb{R}^m$. Lloyd's algorithm divides the dataset into k clusters $u_1, \ldots, u_k$, whose centroids are denoted by $c_1, \ldots, c_k$, where $c_i = (c_{i1}, \ldots, c_{im}) \in \mathbb{R}^m$. The complete Lloyd's algorithm is shown in Algorithm 1.

Since we are concerned with the vertical partition scenario, for each record t_i, the data owner holds $t_{i1}, \ldots, t_{im_1}$ and the server holds $t_{i(m_1+1)}, \ldots, t_{im}$. According to our threat model (described in Sect. 4.2), the data owner only provides encrypted $t_{i1}, \ldots, t_{im_1}$ to the server, so the operation between ciphertext and plaintext is involved in Step 3 because $t_{i(m_1+1)}, \ldots, t_{im}$ is in plaintexts. Since CKKS supports addition, subtraction, and multiplication between ciphertext and plaintext, and the results of these operations are in ciphertexts, the distance array diff is in ciphertexts.

In addition, in Step 3, since CKKS cannot directly support square root and absolute value operations, we calculate the square of the Euclidean distance between each record t_i and each cluster center c_j as diff_{ij}. Each element diff_{ij} in diff array is only used to compare with some other elements, so squaring all elements in diff does not affect the result. Step 4 to 6 must be calculated in ciphertexts, and the ciphertext comparison in Step 4 and the ciphertext division in Step 6 are challenging for CKKS.

5.2 Ciphertext Comparison

In Step 4, we compare the distance between each record and each centroid to find the closest centroid for each record. Since CKKS does not support comparison in ciphertexts directly, the label array cannot be simply calculated, and we need to transform it into some HE-friendly representation.

Considering that the result of Step 4 is used to count the number of records in each cluster and update centroids (Step 5 and 6), it is natural to construct a

one-hot encoding vector for each record. The one-hot encoding vector of record t_i is denoted as minIndex_i $(i = 1, \ldots, n)$:

$$\text{minIndex}_i = (0, \ldots, 0, 1, 0, \ldots, 0)$$

$$\text{minIndex}_{ij} = \begin{cases} 1, & j = \text{label}_i \\ & (\min\{\text{diff}_{i1}, \ldots, \text{diff}_{ik}\} = \text{diff}_{ij}) \\ 0, & j \neq \text{label}_i \\ & (\text{diff}_{ij} \text{ is not the minimum}) \end{cases} \tag{2}$$

where only the label_i-th dimension of minIndex_i is set to 1 and all other dimensions are 0. In other words, minIndex_{ij} equals one only when c_j is the closest centroid to t_i. After this transformation, Steps 5 and 6 can be immediately rewritten as follows:

$$\text{Step 5:} \quad \text{cnt}_j = \sum_{i=1}^{n} \text{minIndex}_{ij} \tag{3}$$

$$\text{Step 6:} \quad c_j = \frac{\sum_{i=1}^{n} t_i \cdot \text{minIndex}_{ij}}{\text{cnt}_j} \tag{4}$$

Now the challenge is to calculate the minIndex array in ciphertexts. And we introduce the sign function sgn(x) to facilitate comparison, where sgn(x) = 1 if $x > 0$, sgn(x) = 0 if $x = 0$, and sgn(x) = -1 if $x < 0$.

During the ciphertext calculation process, the server will not obtain any intermediate results and related information. Therefore, in order to find the cluster center closest to the record t_i and mark the corresponding dimension of minIndex_i as 1, the server needs to compare the k numbers pairwise. Obviously, if diff_{ij} is the minimum value, then for any $q \in [1, k]$ and $q \neq j$, $\frac{\text{sgn}(\text{diff}_{iq} - \text{diff}_{ij}) + 1}{2}$ always equals 1. If diff_{ij} is not the minimum, then there exists $q \in [1, k]$ and $q \neq j$ such that $\frac{\text{sgn}(\text{diff}_{iq} - \text{diff}_{ij}) + 1}{2} = 0$. Therefore, to compute minIndex_{ij}, we compare diff_{ij} with the other $k - 1$ distance differences and multiply all the results:

$$\text{minIndex}_{ij} = \prod_{q=1,\ q \neq j}^{k} \frac{\text{sgn}(\text{diff}_{iq} - \text{diff}_{ij}) + 1}{2} \tag{5}$$

According to Eq. 5, if diff_{ij} is the minimum value over $\{\text{diff}_{i1}, \ldots, \text{diff}_{ik}\}$, then all $k - 1$ factors of minIndex_{ij} equal to 1, i.e., $\text{minIndex}_{ij} = 1$; otherwise, minIndex_{ij} has at least one 0 in its multiplication factors, thus $\text{minIndex}_{ij} = 0$. Therefore, minIndex_{ij} in Eq. 5 satisfies the definition in Eq. 2.

However, $\text{sgn}(x)$ cannot be computed in CKKS directly since it is not a polynomial function. Therefore, the goal is to approximate $\text{sgn}(x)$ using HE-friendly operations. We use the iterative method to solve the above problem. We construct an iterative formula containing two fixed points, requiring that the signs of these two fixed points be different:

$$a_{i+1} = \varphi_c(a_i) = -\frac{3}{2} a_i (a_i - 1)(a_i + 1) \quad (i \in \mathbb{N}) \tag{6}$$

We can determine the convergence of Eq. 6 with $a_0 \in (-1, 1)$:

$$\lim_{n \to +\infty} \sqrt{3} a_n = \begin{cases} 1, & a_0 \in (0, 1) \\ -1, & a_0 \in (-1, 0) \\ 0, & a_0 = 0 \end{cases} \tag{7}$$

Therefore, the domain of Eq. 6 is defined as $(-1, 1)$, and the initial value a_0 is chosen as the input to the sign function denoted as x_{in}, where $x_{in} \in (-1, 1)$.

Algorithm 2: The sign function in ciphertexts

Input: x_{in}, R_c
Output: the approximate value of $\mathrm{sgn}(x_{in})$

```
a_0 = x_in
for i ← 0 to R_c − 1 do
    a_{i+1} = −(3/2) a_i (a_i − 1)(a_i + 1)
return √3 a_{R_c}
```

We can also handle the case where x_{in} are not in the range $(-1, 1)$. Suppose x'_{in} is in $(-M, M)$ for some constant M. In this case, the initial value a_0 is assigned as x'_{in}/M. Since the above ciphertext sign function only contains HE-friendly operations, CKKS can directly compute it.

5.3 Ciphertext Division

In Step 6, we need to divide the sum $\sum_{i=1}^{n} t_i \cdot \mathrm{minIndex}_{ij}$ by the count cnt_j to get the centroid c_j. We apply Newton's method (described in Sect. 3.2) to solve the ciphertext division. We define the output of the function $\mathrm{div}(a)$ as the reciprocal of a, where a is in $(0, M)$ for some constant M. Construct a function f(x), whose root is $\frac{1}{a}$, such as: $f(x) = \frac{1}{x} - a$. And we utilize Newton's method to approximate the root of this function. According to Eq. 1, we can obtain the iterative formula:

$$x_{i+1} = \varphi_d(x_i) = x_i - \frac{f(x_i)}{f'(x_i)} = x_i(2 - a \cdot x_i) \tag{8}$$

We need to determine the domain (a neighborhood of $\frac{1}{a}$) where the above iteration converges, and choose an appropriate initial value x_0 in the domain. Equation 8 converges if $x_0 = \frac{1}{a}$ or $\exists \delta > 0, \forall x \in (\frac{1}{a} - \delta, \frac{1}{a} + \delta) \setminus \{\frac{1}{a}\}$ satisfy

$$\left| \varphi_d(x) - \frac{1}{a} \right| < \left| x - \frac{1}{a} \right| \tag{9}$$

According to the above criteria, we can calculate that the domain is $(0, \frac{2}{a})$. Since $0 < a < M$, we choose $x_0 = \frac{2}{M} \in (0, \frac{2}{a})$ as the initial value. Therefore, we utilize this iteration to construct our ciphertext division algorithm (Algorithm 3).

Algorithm 3: The ciphertext division algorithm

Input: a, M, R_d
Output: the approximate value of $\frac{1}{a}$
1 $x_0 = \frac{2}{M}$
2 **for** $i \leftarrow 0$ ***to*** $R_d - 1$ **do**
3 $\quad x_{i+1} = x_i(2 - a \cdot x_i)$
4 **return** x_{R_d}

The above complete division algorithm uses only subtractions and multiplications in ciphertexts, which are directly supported by the CKKS homomorphic encryption. Hence $c_j = \sum_{i=1}^{n} t_i \cdot \text{minIndex}_{ij} \cdot \text{div}(\text{cnt}_j)$, where $\text{cnt}_j \in (0, n]$.

5.4 Converting the One-Hot Vectors to Label in Plaintexts

To achieve Step 5 and 6, we convert the label array into the one-hot encoding array minIndex. After receiving and decrypting the minIndex array, the data owner needs to extract the label from the minIndex. The data owner finds the index of the element which equals one in minIndex_{ij} for each record i as label_i:

$$\text{label}_i = j \ s.t. \ \text{minIndex}_{ij} = 1 \tag{10}$$

5.5 The Complete Algorithm for Privacy-Preserving Clustering

We summarize the above steps and describe the complete algorithm for fully privacy-preserving clustering. Our scheme contains only HE-friendly operations and is completely secure which means that no private information is revealed to the server during calculation. In Step 2, the data owner randomly generates k cluster centers $c_1, \ldots, c_k$ and encrypts them.

In Step 3, the server calculates the distance between each record t_i and each cluster center c_j, denoted as diff_{ij}. According to our problem setting, the records in D_1 are in ciphertexts, the records in D_2 are in plaintexts, and the centroids are in ciphertexts. CKKS directly supports addition, subtraction, and multiplication between ciphertext and plaintext, and the results of these operations are in ciphertexts. Therefore, all terms of sum are in ciphertexts, and diff_{ij} is also in ciphertexts. In Step 4, the server compares diff_{ij} with diff_{iq} by computing $\text{sgn}(\text{diff}_{iq} - \text{diff}_{ij})$ (Algorithm 2), where $1 \leq q \leq k, q \neq j$. Then, the server multiplies the $k-1$ results to get minIndex_{ij}, where $\text{minIndex}_{ij} = 1$ when diff_{ij} is the minimum value, and $\text{minIndex}_{ij} = 0$ otherwise.

In Step 5, the server counts the number of records belonging to each cluster. In Step 6, the server recalculates each cluster center c_j. Afterwards, the server repeats Step 3 to 6 for T iterations. In Step 7, the server sends the encrypted minIndex array to the data owner. After decrypting minIndex, the data owner extracts the label array according to Eq. 10.

Algorithm 4: The fully privacy-preserving clustering algorithm

Input n, $m = m_1 + m_2$, k, T
$D_1 = \{t_{11}, \ldots, t_{1m_1}, \ldots, t_{n1}, \ldots, t_{nm_1}\}$ (in ciphertexts)
$D_2 = \{t_{1(m_1+1)}, \ldots, t_{1m}, \ldots, t_{n(m_1+1)}, \ldots, t_{nm}\}$ (in plaintexts)
Data owner generates centroids $c_1, \ldots, c_k$ in $[0,1]^m$ randomly and sends them to the server in ciphertexts
repeat T **times**
  Calculate distance of each record to each centroid
  for (i,j) **in** $\{1,\ldots,n\} \times \{1,\ldots,k\}$ **do**
    $\mathrm{diff}_{ij} \leftarrow \sum_{q=1}^{m} (t_{iq} - c_{jq})^2$
  Calculate the minIndex array
  for (i,j) **in** $\{1,\ldots,n\} \times \{1,\ldots,k\}$ **do**
    $\mathrm{minIndex}_{ij} \leftarrow \prod_{q=1, q\neq j}^{k} \frac{\mathrm{sgn}(\mathrm{diff}_{iq} - \mathrm{diff}_{ij}) + 1}{2}$
  Count the number of records in each cluster
  for $j \leftarrow 1$ ***to*** k **do**
    $\mathrm{cnt}_j \leftarrow \sum_{i=1}^{n} \mathrm{minIndex}_{ij}$
  Update centroids
  for $j \leftarrow 1$ ***to*** k **do**
    $c_j \leftarrow \sum_{i=1}^{n} (t_i \cdot \mathrm{minIndex}_{ij} \cdot \mathrm{div}(\mathrm{cnt}_j))$
Output minIndex (in ciphertexts)
Data owner extracts label from minIndex

5.6 Security Proof

The view of the server includes the ciphertext dataset from data owner, the dimension of the input dataset, the number of clusters, and the number of iterations for clustering. When the data owner is corrupted, `Sim` receives the plaintext parameters m_1, k, T, the public key and the ciphertext $\texttt{HE.Enc}(pk, D_1)$ from the data owner. In return, it sends $\texttt{HE.Enc}(pk, \texttt{res})$ for a randomly chosen `res` from $\mathbb{R}^n$, and the data owner cannot distinguish between real-world distributions and simulated distributions.

The view of the data owner includes the input dataset of the data owner, the dimension of the input dataset, the number of clusters, and the number of iterations for clustering. When the server is corrupted, `Sim` sends the plaintext parameters m_1, k, T, a randomly chosen `inp` from $\mathbb{R}^n$ and the public key $\texttt{HE.Enc}(pk, \texttt{inp})$ to the server. The server cannot distinguish between real-world distributions and simulated distributions.

6 An Optimized Algorithm

In this section, to further reduce the run time and memory consumption, we propose an optimized clustering algorithm called block privacy-preserving clustering algorithm. We have described a fully privacy-preserving clustering scheme in Sect. 5.5, however, when we test this algorithm on the popular MNIST dataset [15], the results show that the run time and memory consumption are unacceptable (about 3500 min and 1303 GB memory).

In terms of run time, since ciphertext multiplication is a time-consuming operation, multiple multiplications dramatically affect the efficiency of our clustering algorithm. On the other hand, CKKS is a leveled homomorphic encryption and limits the multiplication level by a pre-determined parameter. Multiple consecutive multiplications are also unfriendly to CKKS. In our algorithm, computing the minIndex array (Step 4 in Algorithm 4) requires multiple multiplications in ciphertexts. This step contains two parts: evaluating the sign function using the difference of two distances and multiplying the evaluation results. And there are $\Theta(nk^2)$ ciphertext multiplications in the second part of this step, where $\Theta(k)$ consecutive multiplications are required to calculate each element in the minIndex array. That is, reducing k can reduce run time.

In terms of memory consumption, in order to calculate the minIndex array with multithreading enabled, $\Theta(nk^2)$ memory is required. That is, when k is large, calculating the minIndex requires a large amount of memory.

However, with the advent of the big data era, the scale of the dataset to be clustered is gradually increasing, and the number of clusters k of the dataset becomes larger. Therefore, it is necessary to propose a new clustering algorithm that reduces run time and memory consumption by reducing k.

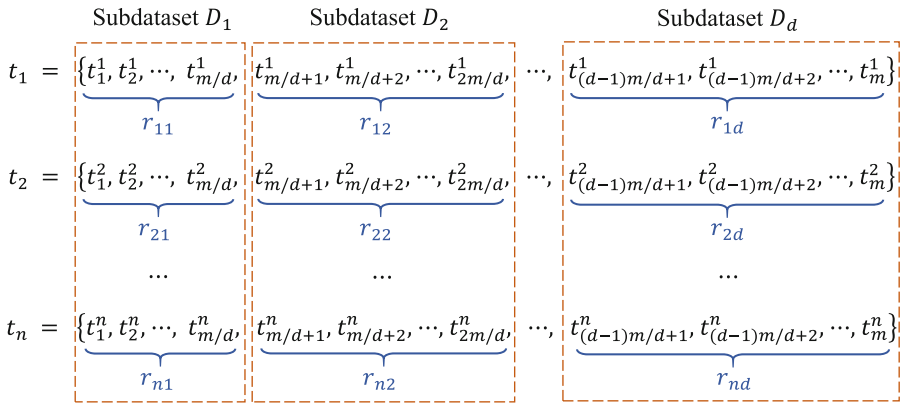

Fig. 2. Schematic Diagram of Splitting the Dataset

6.1 Block Clustering Scheme

To reduce k, we split the input vectors into disjoint subvectors and cluster these subvectors. As shown in Fig. 2, each record t_i in the dataset D is divided into d subvectors denoted $r_{i1}, \ldots, r_{id}$. Each subvector r_{iq} is of m/d dimensions, where m is a multiple of d and q is in $[1, d]$. We denote the set of subvectors $r_{1q}, \ldots, r_{nq}$ as the subdataset D_q.

We perform our privacy-preserving clustering scheme on each subdataset D_q independently, and each subvector r_{iq} is identified by a label L_{iq}. The label L_i of the record t_i is defined as the Cartesian product of the labels of its corresponding subvectors:

$$L_i = L_{i1} \times L_{i2} \times \ldots \times L_{id} \tag{11}$$

Since all subdatasets are equally important, they have the same number of subclusters. In that case, each subdataset is divided into k' subclusters, where

$$k' = \sqrt[d]{k} \tag{12}$$

We assume that k' is always a positive integer. The centroid c_j is the concatenation of the centroids of its corresponding subclusters.

We now summarize our complete scheme for block privacy-preserving clustering. Before delivering the ciphertexts and the parameters to the server, the data owner needs to preprocess the dataset D, including dividing t_i to generate r_{iq} and calculating k'. Then, the fully privacy-preserving block clustering algorithm can be described as Algorithm 5.

In Step 7, the data owner needs to extract the label of each subvector from minIndex$'$ according to the method described in Sect. 5.4. Afterward, the data owner obtains the label of each record by calculating the Cartesian product of its corresponding subvector labels according to Eq. 11.

The above algorithm only contains HE-friendly operations and does not break our threat model in Sect. 4.2. As shown in Table 1, we compare the number of multiplications in each calculation step between Algorithm 4 and Algorithm 5. The efficiency of the block clustering algorithm has been significantly improved, especially when calculating the minIndex array.

The number of consecutive multiplications also decreases in Algorithm 5. When calculating each element in minIndex array, the number of consecutive multiplications decreases from $\Theta(k)$ to $\Theta(\sqrt[d]{k})$. Additionally, memory consumption drops from $\Theta(nk^2)$ to $\Theta(nk^{\frac{2}{d}})$.

Although Algorithm 5 can greatly reduce the run time and memory consumption, it loses the clustering accuracy compared to Algorithm 4. In addition, Algorithm 5 is more suitable for datasets whose records are relatively evenly distributed in m-dimensional space.

6.2 Block Clustering Scheme with Cluster Selection

The fully privacy-preserving block clustering scheme described in Sect. 6.1 is efficient. However, Algorithm 5 only works on datasets with a certain k. When

Algorithm 5: The block privacy-preserving clustering algorithm

Input n, m, k', T, d
$r_{11}, \ldots, r_{1d}, r_{21}, \ldots, r_{2d}, \ldots, r_{n1}, \ldots, r_{nd}$ (Figure 2)
for $q \leftarrow 1$ *to* d **do**
Data owner generates centroids $c'_1, \ldots, c'_{k'}$ in $[0,1]^{m/d}$ randomly and sends them to the server in ciphertexts
repeat T **times**
Calculate distance of each record to each centroid
for (i,j) **in** $\{1,\ldots,n\} \times \{1,\ldots,k'\}$ **do**
$\text{diff}'_{ij} \leftarrow \sum_{l=1}^{m/d} (r_{iql} - c'_{jl})^2$
Calculate the minIndex$'$ array
for (i,j) **in** $\{1,\ldots,n\} \times \{1,\ldots,k'\}$ **do**
$\text{minIndex}'_{ij} \leftarrow \prod_{l=1, l\neq j}^{k'} \frac{\text{sgn}(\text{diff}'_{il} - \text{diff}'_{ij}) + 1}{2}$
Count the number of records in each cluster
for $j \leftarrow 1$ ***to*** k' **do**
$\text{cnt}'_j \leftarrow \sum_{i=1}^{n} \text{minIndex}'_{ij}$
Update centroids
for $j \leftarrow 1$ ***to*** k' **do**
$c'_j \leftarrow \sum_{i=1}^{n} (r_{iq} \cdot \text{minIndex}'_{ij} \cdot \text{div}(\text{cnt}'_j))$
Send minIndex$'$ (in ciphertexts), and the data owner extracts label$'$ from minIndex$'$

for any d, there is no positive integer k' such that $k = {k'}^d$, this algorithm fails. Therefore, to make our block clustering algorithm applicable to datasets with arbitrary k, we add a function called cluster selection to it.

We still split each input vector into d disjoint subvectors, where each subvector is of m/d dimensions. Then we cluster each subdataset (defined in Sect. 6.1) separately and compute the Cartesian product of corresponding subvectors' labels to obtain the label for each vector according to the Eq. 11. Since $\sqrt[d]{k}$ is not a positive integer, we require each subdataset to be divided into k_s subclusters, where $k_s > \sqrt[d]{k}$ and $k_s \in \mathbb{N}^*$, and the specific value of k_s is determined by the data owner. Thus, the dataset D is partitioned into ${k_s}^d$ clusters, where ${k_s}^d > k$.

Since the goal of the data owner is to divide D into k clusters, we now propose the cluster selection algorithm to remove ${k_s}^d - k$ redundant clusters. We sort all ${k_s}^d$ clusters according to the number of records in each cluster from most to least, then keep the top k clusters and record their centroids. The records that do not belong to these top k clusters are called remaining records and need to be reassigned to these clusters. We separately calculate the distance between each remaining record and each centroid, then assign each remaining record to

Table 1. The number of multiplications involved in each step of Algorithm 4 and Algorithm 5 in each step. Note that no multiplication is needed in Step 5.

	Step 3 (distances)	Step 4 (minIndex)	Step 6 (centroids)
Algorithm 4	$\Theta(nmk)$	$\Theta(nk^2)$	$\Theta(nk)$
Algorithm 5	$\Theta(nmk^{\frac{1}{d}})$	$\Theta(dnk^{\frac{2}{d}})$	$\Theta(dnk^{\frac{1}{d}})$

the nearest centroid. Afterward, the centroids of these k clusters need to be recalculated.

We recommend that the data owner implement the cluster selection algorithm. The reason is that this algorithm is time-consuming in ciphertexts, but requires a small amount of computation resource in plaintexts. In addition, the communication costs remain almost unchanged.

The data owner needs to own all dimensions of the records and centroids to guarantee correct clustering of the remaining records. Therefore, our block clustering scheme with plaintext cluster selection only works if the data owner possesses the complete dataset. At this time, the data owner outsources clustering in order to utilize the computation resource of the server. Only Algorithm 4 and Algorithm 5 support the user to simultaneously utilize the server's dataset and computation resource.

7 Experiment Results

7.1 Experiment Setup

Server Configuration. We use the Lattigo library [1] (version 4.1.0) to implement our fully privacy-preserving clustering algorithm. We choose a Ubuntu-20.04 server with an Intel(R) Xeon(R) CPU E5-2620 v4 (2.1 GHz, 32 threads) and 100GB RAM to perform all experiments. Our approach is written in Go.

Datasets. We select several datasets from different sources to evaluate our algorithm. G2 [9] is a series of synthetic datasets, each of which contains 2048 points divided into two Gaussian clusters of equal size. We choose G2-1-20, G2-2-20, G2-4-20, G2-8-20, and G2-16-20 for experiments, where 1,2,4,8,16 represent the dimensions of the dataset.

The fundamental clustering problems suite (FCPS) [33] contains nine different datasets, and we choose seven of them which are widely used in [8,17,20,28,34]. As shown in Table 2, these datasets with known labels are low-dimensional and simple, and each of them solves a certain problem in clustering.

To demonstrate the effectiveness of the scheme described in Sect. 6, we select two large datasets to test our block clustering scheme. The MNIST dataset [15] of handwritten digits contains 60000 training images and 10000 testing images, where $m = 784$. We run our algorithm on the testing images with 10 clusters. The CIFAR-10 dataset [14] consists of 60000 images divided into 10 clusters. We choose 10000 test images to perform our approach.

Table 2. The datasets chosen from FCPS [33].

Dataset	n	m	k	Main Problems
Chainlink	1000	3	2	Not linearly separable
EngyTime	4096	2	2	Gaussian mixture
Hepta	212	3	7	None
Lsun	400	2	3	Different variances and shapes
Tetra	400	3	4	Almost touching
TwoDiamonds	800	2	2	Touching clusters
WingNut	1016	2	2	Non-uniform density

Parameter Selection. We choose the following parameters for our experiments:

- Parameters of CKKS: We use a default parameter set in the Lattigo library called PN16QP1761CI. $logN$ is 16, which means that at most 65536 values can be batched simultaneously. The multiplication level is 34, indicating that the number of consecutive multiplications cannot exceed 34.
- Number of iterations: 5, 10, 20.

In all experiments, the initial k centroids are uniformly and independently randomly sampled from $[0, 1]^m$. We run each experiment with five different random seeds and average the results.

Since we focus on the vertical partition scenario, the input to Algorithm 4 consists of encrypted dataset D_1 and unencrypted dataset D_2. The operation between plaintext and ciphertext is faster than that between ciphertext, therefore, in order to make the result more convincing, we encrypt all dimensions of the records in all experiments.

7.2 Clustering Accuracy

In this section, we conduct experiments on datasets G2 and FCPS to test the clustering accuracy of the algorithm described in Sect. 5. We then compare the accuracy of our approach with the state-of-the-art privacy-preserving clustering algorithm proposed in [13].

We separately run our algorithm and vanilla Lloyd's algorithm on the datasets and count the number of records that are correctly clustered. Then we calculate the clustering accuracy by dividing the number of correctly clustered records by n. In this part, we choose $T = 5$ for G2 and $T = 10$ for FCPS. Since the order of clusters in the experiments may be different from that in the ground truths, we enumerate full permutations of $\{1, \ldots, k\}$ as the mapping of labels from the algorithm outputs to the ground truth, and find the one with most correctly clustered records.

As shown in Table 3, for all datasets, the clustering accuracy of our approach is almost identical to that of Lloyd's algorithm. On most datasets, our algorithm

Table 3. Clustering accuracy of our algorithm and Lloyd's algorithm on G2 and FCPS.

Dataset	Our Algorithm	Lloyd's Algorithm	Difference
G2-1-20	99.3%	99.4%	0.1%
G2-2-20	100%	100%	0.0%
G2-4-20	100%	100%	0.0%
G2-8-20	100%	100%	0.0%
G2-16-20	100%	100%	0.0%
Chainlink	66.5%	65.9%	−0.6%
EngyTime	94.9%	95.1%	0.2%
Hepta	85.8%	85.4%	−0.4%
Lsun	77.5%	76.8%	−0.7%
Tetra	100%	100%	0.0%
TwoDiamonds	100%	100%	0.0%
WingNut	96.3%	96.4%	0.1%

correctly clusters the same number of records as Lloyd's algorithm. Therefore, we conclude that adding privacy protection to the clustering scheme does not affect accuracy.

We compare the clustering accuracy of our approach with that of the approximate algorithm proposed in [13] and record the results in Table 4. Since the bitwise ciphertext comparison used in their algorithm would lead to unacceptable run time, they propose an approximate version that improves efficiency by truncating several least significant bits. They compute the difference in clustering accuracy between the exact clustering and their approximate algorithm to demonstrate the accuracy (exact − approximate). In addition, we calculate the difference in clustering accuracy between Lloyd's algorithm and ours, and compare our results to theirs on the four datasets, as shown in Table 4. There are two negative numbers in the second line, indicating that the accuracy of our scheme is higher than that of Lloyd's algorithm. For these four datasets, our algorithm obviously outperforms theirs in clustering accuracy.

Table 4. Accuracy difference between exact algorithm and privacy-preserving version

	Lsun	Hepta	Tetra	WingNut
Approximate Version in [13]	3.5%	4%	13%	1.25%
This Work	−0.8%	−0.5%	0%	0.1%

7.3 Run Time

In this section, we test the run time of our fully privacy-preserving clustering algorithm described in Sect. 5. Then, we compare the efficiency of our algorithm with that of the state-of-the-art FHE-based clustering scheme [13].

Due to the inefficiency of direct implementation based on CKKS, we optimize our method for efficiency by parallelizing ciphertext computation (a.k.a. batching) and multithreading. First, we leverage the SIMD capability of CKKS to parallelize operations. Denote N as the ring dimension, which defines the number of coefficients of the plaintext/ciphertext polynomials. Since we choose the conjugate-invariant variant of CKKS for the ring type, at most N real values of the plaintext can be packed into one ciphertext. In Algorithm 4, the calculation of the diff array and the minIndex array can be parallelized $\min(n, N)$ times by simply packing $\min(n, N)$ coordinates into a single ciphertext. When counting the number of records in each cluster, we sum the n values (w.l.o.g. assuming $n \leq N$) in the packed minIndex array, which can be solved by rotations and additions.

Second, we can further improve the efficiency of our algorithm by multithreading. All operations in our algorithm are independent of each cluster. When computing the distances and the minIndex array, the operations are independent of each record. Therefore, we can achieve multithreading across different batches and different clusters.

We execute the unoptimized algorithm and the algorithm with batching and multithreading on G2 and FCPS respectively, then record the results in Table 5. In this part, we choose $T = 5$ for G2 and $T = 10$ for FCPS. As shown in Table 5, batching and multithreading can significantly reduce the run time of our scheme, and the total speedup of these two optimizations reaches a maximum value of about 6426x on G2-16-20. As shown in Table 6, our optimized algorithm is 4∼5 orders of magnitude slower than vanilla Lloyd's algorithm. It is known that FHE-based algorithms are generally 9 orders of magnitude slower than corresponding plaintext algorithms [22]. We conclude that our privacy-preserving clustering scheme is efficient.

Table 5. Run time of G2 and FCPS with and without batching

Dataset	Without Optimization[4]	With Optimizations	Speedup
G2-1-20	271.0 h	222.41 s	4386 x
G2-2-20	305.5 h	221.11 s	4974 x
G2-4-20	374.6 h	250.20 s	5390 x
G2-8-20	512.6 h	311.55 s	5923 x
G2-16-20	788.8 h	441.89 s	6426 x
Chainlink	106.3 h	421.09 s	909 x
EngyTime	488.5 h	394.87 s	4454 x
Hepta	440.2 h	1213.90 s	1305 x
Lsun	129.7 h	442.65 s	1055 x
Tetra	239.6 h	620.42 s	1390 x
TwoDiamonds	95.5 h	397.73 s	864 x
WingNut	121.2 h	395.45 s	1103 x

To avoid the long run time, we run several synthetic datasets with small sizes and fit the run time in the table using least squares.

Table 6. Plaintext run time of vanilla Lloyd's algorithm on G2 and FCPS

G2-1-20	G2-2-20	G2-4-20	G2-8-20	G2-16-20	Chainlink
0.042 s	0.048 s	0.060 s	0.084 s	0.130 s	0.050 s
EngyTime	Hepta	Lsun	Tetra	TwoDiamonds	WingNut
0.177 s	0.025 s	0.023 s	0.031 s	0.036 s	0.045 s

We then compare the efficiency of our algorithm with that of [13] on the Lsun dataset. The run time reported in [13] is single-threaded. Thus, to achieve a fair comparison, we test the run time of our algorithm with a single thread. Since our method does not lose clustering accuracy, we choose the exact version of their work for comparison. As shown in Table 7, our approach spends 1606.36 s on the Lsun dataset, which is 19573x faster than [13]. Furthermore, our approach is still 1258x faster than their approximate algorithm (the fastest version in [13]), which trades accuracy for efficiency. To sum up, our fully privacy-preserving clustering scheme significantly outperforms [13] in terms of efficiency.

Table 7. Efficiency comparison with [13] on Lsun.

Work	Version	Threads	Run Time ($T = 10$)
Jäschke et al. [13]	exact	one	363.90 days
Jäschke et al. [13]	approximate	one	154.70 h
This work	exact	one	1606.36 s

7.4 Performance of Block Clustering Scheme with Cluster Selection

In this section, we evaluate the efficiency, memory consumption, and accuracy of our block clustering scheme described in Sect. 6. To demonstrate the superior performance of our block clustering algorithm, we select two large-scale, widely used datasets (MNIST, CIFAR-10) to test the run time and memory consumption. For CIFAR-10, we use the resnet20 model [2] to extract the feature vectors with $m = 64$. We choose $T = 20$ for all experiments in this section. Since $k = 10$ for MNIST and CIFAR-10, there is no d greater than 1, making $\sqrt[d]{k}$ a positive integer. Therefore, we execute our block clustering scheme with cluster selection described in Sect. 6.2 on them, where $d = 4, k_s = 2$ for MNIST, and $d = 2, k_s = 4$ for CIFAR-10.

As shown in Table 8, we first measure the run time and memory consumption of our block clustering algorithm, where $\text{time}_{\text{block}}$ and $\text{memory}_{\text{block}}$ represent the run time and memory consumption of the block clustering algorithm respectively, and $\text{time}_{\text{original}}$ and $\text{memory}_{\text{original}}$ represent these two indicators of our vanilla privacy-preserving clustering algorithm. Since our server configuration does not support our original algorithm to perform on such large datasets, $\text{time}_{\text{original}}$

and $\text{memory}_{\text{original}}$ in Table 8 are estimated. Notably, our block clustering solution can be executed on these datasets with acceptable run time. Among them, only 127 min and 47GB memory are required on CIFAR-10. That is, our block clustering scheme makes it possible to outsource privacy-preserving clustering on large datasets.

Table 8. Efficiency of block clustering scheme on MNIST and CIFAR-10

	MNIST($d = 4, k_s = 2$)	CIFAR-10($d = 2, k_s = 4$)
$\text{time}_{\text{original}}{}^2$	about 3500 min	about 350 min
$\text{time}_{\text{block}}$	850 min	127 min
$\text{memory}_{\text{original}}{}^2$	about 1303 GB	about 226 GB
$\text{memory}_{\text{block}}$	75 GB	47 GB

Due to insufficient memory, we run our original algorithm on several small datasets and fit the memory consumption and run time in the table by least squares.

We then test the clustering accuracy of our method on MNIST and CIFAR-10. We count the number of correctly clustered records and divide them by n to calculate the accuracy. The experimental results show that the accuracy loss is 10.35% on MNIST and 8.55% on CIFAR-10, which can be acceptable. In summary, our block privacy-preserving clustering algorithm is suitable for data owners who would like to trade a little accuracy for high efficiency. Furthermore, this solution enables outsourced clustering of large-scale datasets while preserving privacy.

8 Conclusions

We achieve ciphertext comparison by constructing an iterative formula with two fixed points of opposite signs to approximate the sign function. The solution of ciphertext division is to use Newton's method to approximate the reciprocal of the divisor. After solving the challenging ciphertext comparison and division, we propose a fully privacy-preserving, effective, and efficient clustering algorithm based on CKKS. However, executing our fully privacy-preserving clustering scheme on the large-scale datasets results in unacceptable run time and memory consumption. To further reduce the run time and memory consumption of our algorithm, we propose a block privacy-preserving clustering algorithm that splits records into subvectors and clusters these subvectors. Experiment results show that our original clustering algorithm has the same accuracy as Lloyd's algorithm and has a huge efficiency advantage over the baseline. In experiments testing our block clustering algorithm, it can be concluded that this algorithm performs well in terms of both efficiency and memory consumption.

References

1. Lattigo v4, August 2022. https://github.com/tuneinsight/lattigo, ePFL-LDS, Tune Insight SA
2. Pytorch cifar models, August 2022. https://github.com/chenyaofo/pytorch-cifar-models
3. Almutairi, N., Coenen, F., Dures, K.: K-means clustering using homomorphic encryption and an updatable distance matrix: secure third party data clustering with limited data owner interaction. In: Bellatreche, L., Chakravarthy, S. (eds.) DaWaK 2017. LNCS, vol. 10440, pp. 274–285. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-64283-3_20
4. Ashari, I., Banjarnahor, R., Farida, D., Aisyah, S., Dewi, A., Humaya, N.: Application of data mining with the k-means clustering method and davies bouldin index for grouping imdb movies. J. Appl. Inform. Comput. **6**(1), 07–15 (2022). https://doi.org/10.30871/jaic.v6i1.3485. https://jurnal.polibatam.ac.id/index.php/JAIC/article/view/3485
5. Balcan, M.F., Dick, T., Liang, Y., Mou, W., Zhang, H.: Differentially private clustering in high-dimensional Euclidean spaces. In: Precup, D., Teh, Y.W. (eds.) Proceedings of the 34th International Conference on Machine Learning. Proceedings of Machine Learning Research, vol. 70, pp. 322–331. PMLR (06–11 Aug 2017). https://proceedings.mlr.press/v70/balcan17a.html
6. Cheon, J.H., Han, K., Kim, A., Kim, M., Song, Y.: Bootstrapping for approximate homomorphic encryption. In: Nielsen, J.B., Rijmen, V. (eds.) EUROCRYPT 2018. LNCS, vol. 10820, pp. 360–384. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-78381-9_14
7. Cheon, J.H., Kim, A., Kim, M., Song, Y.: Homomorphic encryption for arithmetic of approximate numbers. In: Takagi, T., Peyrin, T. (eds.) ASIACRYPT 2017. LNCS, vol. 10624, pp. 409–437. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-70694-8_15
8. Estévez, P.A., Figueroa, C.J.: Online data visualization using the neural gas network. Neural Netw. **19**(6), 923–934 (2006). https://doi.org/10.1016/j.neunet.2006.05.024. advances in Self Organising Maps - WSOM'05
9. Fränti, P., Sieranoja, S.: K-means properties on six clustering benchmark datasets. Appl. Intell. **48**(12), 4743–4759 (2018). https://doi.org/10.1007/s10489-018-1238-7
10. Galántai, A.: The theory of newton's method. Journal of Computational and Applied Mathematics **124**(1), 25–44 (2000). https://doi.org/10.1016/S0377-0427(00)00435-0. https://www.sciencedirect.com/science/article/pii/S0377042700004350, numerical Analysis 2000. Vol. IV: Optimization and Nonlinear Equations
11. Gheid, Z., Challal, Y.: Efficient and privacy-preserving k-means clustering for big data mining. In: 2016 IEEE Trustcom/BigDataSE/ISPA, pp. 791–798 (2016). https://doi.org/10.1109/TrustCom.2016.0140
12. Huang, Z., Liu, J.: Optimal differentially private algorithms for k-means clustering. In: Proceedings of the 37th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, PODS 2018, pp. 395–408. Association for Computing Machinery, New York (2018). https://doi.org/10.1145/3196959.3196977, https://doi.org/10.1145/3196959.3196977
13. Jäschke, A., Armknecht, F.: Unsupervised machine learning on encrypted data. In: Cid, C., Jacobson, M.J., Jr. (eds.) Selected Areas in Cryptography - SAC 2018, pp. 453–478. Springer, Cham (2018). https://doi.org/10.1007/978-3-030-10970-7_21

14. Krizhevsky, A.: Learning multiple layers of features from tiny images. Tech. rep. (2009)
15. LeCun, Y., Cortes, C., Burges, C.: Mnist handwritten digit database. ATT Labs [Online]. (2010). http://yann.lecun.com/exdb/mnist**2**
16. Li, B., Micciancio, D.: On the security of homomorphic encryption on approximate numbers. Springer-Verlag (2021). https://doi.org/10.1007/978-3-030-77870-5_23
17. Li, F., Qian, Y., Wang, J., Dang, C., Jing, L.: Clustering ensemble based on sample's stability. Artif. Intell. **273**, 37–55 (2019). https://doi.org/10.1016/j.artint.2018.12.007
18. Liu, D., Bertino, E., Yi, X.: Privacy of outsourced k-means clustering. In: Proceedings of the 9th ACM Symposium on Information, Computer and Communications Security, pp. 123–134. ASIA CCS '14, Association for Computing Machinery, New York (2014). https://doi.org/10.1145/2590296.2590332. https://doi.org/10.1145/2590296.2590332
19. Lloyd, S.: Least squares quantization in PCM. IEEE Trans. Inf. Theory **28**(2), 129–137 (1982). https://doi.org/10.1109/TIT.1982.1056489
20. Lopez, C., Tucker, S., Salameh, T., Tucker, C.: An unsupervised machine learning method for discovering patient clusters based on genetic signatures. J. Biomed. Inform. **85**, 30–39 (2018). https://doi.org/10.1016/j.jbi.2018.07.004
21. MacQueen, J.: Some methods for classification and analysis of multivariate observations. In: Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability, vol. 1, pp. 281–297. University of California Press (1967)
22. Matsuoka, K., Banno, R., Matsumoto, N., Sato, T., Bian, S.: Virtual secure platform: A {Five-Stage} pipeline processor over {TFHE}. In: 30th USENIX security symposium (USENIX Security 21), pp. 4007–4024 (2021)
23. Minh, H.L., Sang-To, T., Abdel Wahab, M., Cuong-Le, T.: A new metaheuristic optimization based on k-means clustering algorithm and its application to structural damage identification. Knowl.-Based Syst. **251**, 109189 (2022). https://doi.org/10.1016/j.knosys.2022.109189. https://www.sciencedirect.com/science/article/pii/S0950705122005913
24. Mohassel, P., Rosulek, M., Trieu, N.: Practical privacy-preserving k-means clustering. Cryptology ePrint Archive, Paper 2019/1158 (2019), https://eprint.iacr.org/2019/1158. https://eprint.iacr.org/2019/1158
25. More, J.J., Sorensen, D.C.: Newton's method (2 1982). https://doi.org/10.2172/5326201. https://www.osti.gov/biblio/5326201
26. Ni, L., Li, C., Wang, X., Jiang, H., Yu, J.: Dp-mcdbscan: differential privacy preserving multi-core dbscan clustering for network user data. IEEE Access **6**, 21053–21063 (2018). https://doi.org/10.1109/ACCESS.2018.2824798
27. Rao, F.Y., Samanthula, B.K., Bertino, E., Yi, X., Liu, D.: Privacy-preserving and outsourced multi-user k-means clustering. In: 2015 IEEE Conference on Collaboration and Internet Computing (CIC), pp. 80–89 (2015). https://doi.org/10.1109/CIC.2015.20
28. Rodriguez, M.Z., Comin, C.H., Casanova, D., Bruno, O.M., Amancio, D.R., Costa, L.d.F., Rodrigues, F.A.: Clustering algorithms: A comparative approach. PLOS ONE **14**(1), 1–34 (01 2019). https://doi.org/10.1371/journal.pone.0210236. https://doi.org/10.1371/journal.pone.0210236
29. Rong, H., Wang, H.M., Liu, J., Xian, M.: Privacy-preserving k-nearest neighbor computation in multiple cloud environments. IEEE Access **4**, 9589–9603 (2016). https://doi.org/10.1109/ACCESS.2016.2633544

30. Samanthula, B.K., Elmehdwi, Y., Jiang, W.: k-nearest neighbor classification over semantically secure encrypted relational data. IEEE Trans. Knowl. Data Eng. **27**(5), 1261–1273 (2015). https://doi.org/10.1109/TKDE.2014.2364027
31. Stemmer, U.: Locally private k-means clustering. In: Proceedings of the Thirty-First Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2020, pp. 548–559. Society for Industrial and Applied Mathematics, USA (2020)
32. Su, D., Cao, J., Li, N., Bertino, E., Jin, H.: Differentially private k-means clustering. In: Proceedings of the Sixth ACM Conference on Data and Application Security and Privacy, CODASPY 2016, pp. 26–37. Association for Computing Machinery, New York (2016). https://doi.org/10.1145/2857705.2857708. https://doi.org/10.1145/2857705.2857708
33. Ultsch, A.: Clustering wih som: U* c. Proc. Workshop on Self-Organizing Maps (01 2005)
34. Ultsch, A.: Emergence in self organizing feature maps. In: The 6th International Workshop on Self-Organizing Maps (WSOM 2007) (2007). https://doi.org/10.2390/biecoll-wsom2007-114. https://doi.org/10.2390/biecoll-wsom2007-114
35. Wei, W., ming Tang, C., Chen, Y.: Efficient privacy-preserving k-means clustering from secret-sharing-based secure three-party computation. Entropy 24 (2022)
36. Wu, W., Liu, J., Rong, H., Wang, H., Xian, M.: Efficient k-nearest neighbor classification over semantically secure hybrid encrypted cloud database. IEEE Access **6**, 41771–41784 (2018). https://doi.org/10.1109/ACCESS.2018.2859758
37. Wu, W., Liu, J., Wang, H., Hao, J., Xian, M.: Secure and efficient outsourced k-means clustering using fully homomorphic encryption with ciphertext packing technique. IEEE Trans. Knowl. Data Eng. **33**(10), 3424–3437 (2021). https://doi.org/10.1109/TKDE.2020.2969633