entropy

MDPI

Article

Efficient Privacy-Preserving *K*-Means Clustering from Secret-Sharing-Based Secure Three-Party Computation

Weiming Wei [1], Chunming Tang [1,*] and Yucheng Chen [2]

[1] School of Mathematics and Information Science, Guangzhou University, Guangzhou 510006, China
[2] School of Mathematics, Jiaying University, Meizhou 514015, China
* Correspondence: ctang@gzhu.edu.cn

Abstract: Privacy-preserving machine learning has become an important study at present due to privacy policies. However, the efficiency gap between the plain-text algorithm and its privacy-preserving version still exists. In this paper, we focus on designing a novel secret-sharing-based *K*-means clustering algorithm. Particularly, we present an efficient privacy-preserving *K*-means clustering algorithm based on replicated secret sharing with honest-majority in the semi-honest model. More concretely, the clustering task is outsourced to three semi-honest computing servers. Theoretically, the proposed privacy-preserving scheme can be proven with full data privacy. Furthermore, the experimental results demonstrate that our proposed privacy version reaches the same accuracy as the plain-text one. Compared to the existing privacy-preserving scheme, our proposed protocol can achieve about 16.5×–25.2× faster computation and 63.8×–68.0× lower communication. Consequently, the proposed privacy-preserving scheme is suitable for secret-sharing-based secure outsourced computation.

Keywords: privacy-preserving *K*-means clustering; secure outsourced computation; replicated secret sharing; semi-honest model

Citation: Wei, W.; Tang, C.; Chen, Y. Efficient Privacy-Preserving *K*-Means Clustering from Secret-Sharing-Based Secure Three-Party Computation. *Entropy* **2022**, *24*, 1145. https://doi.org/10.3390/e24081145

Academic Editors: Andrea Prati, Luis Javier García Villalba and Vincent A. Cicirello

Received: 19 July 2022
Accepted: 15 August 2022
Published: 18 August 2022

1. Introduction

With the rapid development of machine learning, Machine Learning as a Service (MLaaS) has become a popular business. Nowadays, machine learning is also widely applied in different fields, such as finance, healthcare, image recognition, and so on. Major companies such as Microsoft, Google, Amazon, etc. are beginning to provide cloud-based MLaaS. In general, these services allow the machine learning algorithms to be updated and improved via input data from their users. In order to gain high-precision model, companies tend to come together and train a common model using their datasets.

However, with the improvement of awareness of privacy, the problems caused by data privacy leakage have become increasingly prominent. On the one hand, the user, who uses the MLaaS service, hopes that the service is conducted without revealing any information of their queries and prediction result. On the other hand, companies want to train a common model without sharing their dataset. Therefore, it is important to find a secure way for privacy-preserving machine learning (PPML) to proceed.

Privacy-preserving machine learning can be tracked back to privacy-preserving data mining, which was firstly introduced by Lindell and Pinkas [1]. Since then, more and more researchers have put focus on privacy-preserving machine learning.

The dataset can be divided into two main types: labeled data and unlabeled data. Generally speaking, the former usually uses supervised learning algorithms when training the model, while the latter unsupervised learning algorithm [2]. In recent years, most solutions of PPML only consider the supervised learning algorithm, and there is less consideration of the unsupervised learning algorithm.

As an unsupervised machine learning technique, similar input records are grouped into clusters while records belonging to different clusters should be maximally different [3].

In this work, we focus on clustering, which plays an extremely important role in data processing and analysis. The goal of clustering is to divide given unlabeled data into several disjoint subsets, such that each subset has similar properties to each other.

The *K*-means algorithm is one of the most well-known clustering algorithms. *A privacy-preserving K-means clustering, which has full data privacy, allows the parties to cluster their combined datasets without revealing any other information except for the final centroid* [3]. In other words, the information of intermediate centroids, cluster assignments, and cluster sizes should be protected in the protocol. Although there are several works for privacy-preserving *K*-means clustering at present, only few of them consider both full data privacy and efficiency. This raises the question:

Could we find a way to achieve both full data privacy for security and efficiency for practicability?

As we will show below, the answer is yes with our replicated secret-sharing-based *K*-means clustering protocol.

1.1. Related Work

In general, different algorithms have different privacy-preserving tools. For example, the chaotic system is a popular tool for image encryption [4]. Privacy-preserving *K*-means clustering falls into one of two categories: (i) homomorphic encryption-based and (ii) secure multiparty computation-based.

Homomorphic encryption (HE) was first proposed by Rivest et al. [5] in 1978. Homomorphic encryption is an encryption scheme where there exists a homomorphism relationship between operations on the plaintext and operations on the ciphertext, such that one can operation on the ciphertext can proceed without leaking any information of plaintext and it obtains the same effect as operation on plaintext after decrypting the result. HE is also widely applied in secure outsourced computation. HE can be divided into fully homomorphic encryption (FHE) and partially homomorphic encryption. FHE supports arbitrary computation on ciphertexts without any limitation. The first feasible FHE scheme was proposed by Gentry [6] in 2009, but it is inefficient. Instead of using inefficient FHE, many researchers adopt efficient partially homomorphic encryption, which only supports homomorphic addition or homomorphic multiplication. For example, RSA [5], Paillier [7], and ElGamal [8] are common partially homomorphic encryptions.

Generally speaking, *an HE-based scheme provides full data privacy, as long as the underlying HE cryptosystem is secure*. The first HE-based *K*-means clustering scheme was given by Vaidya and Clifton [9] in 2003, but it does not satisfy full data privacy. In 2007, Bunn and Ostrovsky [10] presented a two-party privacy-preserving *K*-means scheme based on additive homomorphic encryption that guarantees full data privacy in the semi-honest model. In 2015, Rao et al. [11] proposed a parallelable outsourced distributed clustering protocol based on Paillier homomorphic encryption in the federated cloud environment, but their work is inefficient due to its bit-array-based comparison; hence, Kim and Chang [12] improved it with a new secure comparison protocol. Jäschke and Armknecht [13] proposed *K*-means clustering based on FHE over the torus (TFHE) [14], which provides full privacy guarantees. Cai and Tang [15] proposed their *K*-means algorithm based on Liu's homomorphic encryption [16], which has been proven insecure [17]. Unfortunately, all of these schemes do not scale for large datasets due to the heavily homomorphic operations.

Secure two-party computation (2PC) was first presented by Yao [18] in 1982 and extended to multiparty computation (MPC) by Goldreich et al. [19] in 1987. According to the parties number, we construct 2PC protocol with Yao's garbled circuit (GC) [20], while MPC is done with secret sharing (SS). To further enhance efficiency, *t*-out-of-*n* threshold secret sharing has been applied in recent years. Moreover, the algebraic structure is another important optimized direction. For example, Shamir's secret sharing (SSS) is a famous threshold secret sharing method [21], but it generally works for finite field, such as prime field $\mathbb{Z}_p$, which is inefficient compared with several protocols operating over the-power-of-two ring $\mathbb{Z}_{2^\ell}$ in PPML. This is because the latter takes full advantage of the underlying CPU architecture.

In this work, we only focus on SS-based *K*-means clustering. Doganay et al. [22] proposed distributed privacy preserving *K*-means clustering with additive secret sharing (ASS), but this work reveals the final cluster assignments to parties. Patel et al. [23,24] proposed their *K*-means algorithm under different security model. Upmanyu et al. [25] and Baby and Chandra [26] respectively presented a distributed threshold secret sharing scheme based on the Chinese remainder theorem (CRT-SS). However, none of them provide full privacy guarantees as shown in [3]. In 2020, Mohassel et al. [27] presented 2PC *K*-means clustering protocol with 2-out-of-2 additive secret sharing, and although it provides full data privacy, it is inefficient in terms of computation and communication overhead, because their work heavily relies on garbled circuit and oblivious transfer (OT). Therefore, this scheme is not practical for large-scale clustering tasks.

As shown above, most of the existing privacy-preserving *K*-means clustering protocols take no account of full data privacy. In addition, the gap of efficiency still exists compare with plaintext training. Algorithm inefficiency also limits its practicality, especially for large-scale training tasks. In this work, we want to construct privacy-preserving *K*-means clustering, which has full data privacy for security and high efficiency for practicality.

1.2. Our Contributions

In this work, we only focus on SS-based *K*-means clustering schemes. We provide the comparison with existing SS-based *K*-means clustering in Table 1. We propose an efficient three-party computation protocol for privacy-preserving *K*-means clustering. Concretely, our contributions are described as follows:

- Our protocol provides full privacy guarantees, which allows different computing parties to cluster the combined datasets without revealing any other information except the final centroids.
- Our protocol is based on replicated secret sharing (RSS), which is a 2-out-of-3 threshold secret sharing proposed by Araki et al. [28] and is suitable for constructing efficient protocol over $\mathbb{Z}_{2^\ell}$. Our protocol is secure against a single corrupt server under a semi-honest model. We analyze the security with universal composition framework [29].
- The experimental results demonstrate that our protocol reaches the same accuracy as the plaintext *K*-means clustering algorithm. With the fast network, our privacy-preserving scheme can deal with datasets of million points in an acceptable time.

Table 1. The comparison of different SS-based *K*-means schemes. ASS: Additive secret sharing, CRT-SS: Chinese remainder theorem secret sharing, SSS: Shamir's secret sharing, RSS: Replicated secret sharing. ZKP: Zero knowledge proof, GC: Garbled circuit. OT: Oblivious transfer. N/A: Undefined. L1: intermediate centroids, L2: intermediate cluster sizes, L3: other intermediate values (e.g., intermediate cluster assignments or distance comparison results). ✓: no leakage, ✗: leakage. FDP: Full data privacy.

Scheme	Security	Technology	Domain	L1	L2	L3	FDP
Doganay et al. [22]	Semi-honest	ASS	N/A	✓	✓	✗	✗
Upmanyu et al. [25]	Semi-honest	CRT-SS	$\mathbb{Z}_p$	✓	✗	✗	✗
Patel et al. [23]	Semi-honest	SSS	$\mathbb{Z}_p$	✗	✗	✓	✗
Patel et al. [24]	Malicious	SSS+ZKP	$\mathbb{Z}_p$	✗	✗	✓	✗
Baby and Chandra [26]	N/A	CRT-SS	$\mathbb{Z}_p$	✗	✗	✗	✗
Mohassel et al. [27]	Semi-honest	ASS+GC+OT	$\mathbb{Z}_{2^\ell}$	✓	✓	✓	✓
This work	Semi-honest	RSS	$\mathbb{Z}_{2^\ell}$	✓	✓	✓	✓

1.3. Roadmap

The remaining sections are organized as follows. In Section 2, we give the definition of basic notation, threat model, and security assumption of secure computation, and plaintext algorithm related to *K*-means clustering. In Section 3, we give the cryptographic building blocks. In Section 4, we propose our efficient three-party protocol construction. We give

detailed security analysis of our protocol in Section 5. Then, we report the experimental results of our construction in Section 6. Finally, we conclude this paper in Section 7.

2. Preliminaries

2.1. Basic Notation

We denote the party i by $\mathcal{P}_i$ for each $i \in \{1,2,3\}$. For simplicity, we define $\mathcal{P}_0 = \mathcal{P}_3$, and $\mathcal{P}_4 = \mathcal{P}_1$ in the context. $x \in_R \mathbb{F}$ is chosen uniformly at random from finite set $\mathbb{F}$. We write a bold letter $\boldsymbol{v}$ to denote a d-dimension vector. The j-th component of vector $\boldsymbol{v}$ is v_j. If x is a ℓ-bit number, then $x[i]$ is its i-th bit. Let κ be the security parameter. We use $[n]$ to denote set $\{1, \cdots, n\}$. Furthermore, we assume all float-point data are encoded as ℓ-bit fixed-point number with f-bit precision, where $f < \ell$.

2.2. Threat Model and Security Assumption

Our protocol follows a static and semi-honest model [30] under the honest-majority setting, i.e., the adversary $\mathcal{A}$ only corrupts a single and fixed party during protocol executing. In this setting, the corrupted party follows protocol honestly and wants to learn the input of other parties from received messages. Therefore, the semi-honest model is also called the passive model. Furthermore, we assume the parties communicate with other parties through a secure channel and the network is synchronized.

We prove security using a universally composable framework [29] in the ideal–real paradigm [30]. Let $\mathcal{F}$ be the ideal functionality executed by a trusted third party (TTP) in the ideal world, and Π be the real protocol executed by all parties in the real world. In the ideal world, there is a simulator Sim that plays as adversary $\mathcal{A}$. Let C be the set of corrupted parties and x_i be $\mathcal{P}_i$'s input. We define the ideal interaction and the real interaction as follows:

- $\text{Ideal}_{\mathcal{F},\mathsf{Sim}}(\kappa, C; x_1, \cdots, x_n)$: Compute $(y_1, \cdots, y_n) \leftarrow \mathcal{F}(x_1, \cdots, x_n)$;
  Output $\mathsf{Sim}(C, \{(x_i, y_i), i \in C\}), (y_1, \cdots, y_n)$, where y_i is $\mathcal{P}_i$'s output.
- $\text{Real}_{\Pi,\mathcal{A}}(\kappa, C; x_1, \cdots, x_n)$: Run the protocol Π;
  Output $\{\text{View}_i, i \in C\}, (y_1, \cdots, y_n)$, where View_i is the final view of $\mathcal{P}_i$.

We say the protocol Π securely computes the functionality $\mathcal{F}$ in the semi-honest model, if the view of simulator in the ideal world is indistinguishable from the view of adversary in the real world. We refer the reader to [30] for more details.

2.3. The K-Means Clustering Algorithm

Given dataset $D = \{\mathbf{P}_1, \cdots, \mathbf{P}_n\}$ with n data points, each point $\mathbf{P}_i$ is a d-dimension vector $(P_{i1}, \cdots, P_{id})$. We form $n \times d$ matrix $\mathbf{P}$. A standard K-means clustering algorithm includes the following steps [10,27]:

1. Cluster centroids initialization: randomly choose K different points as initialized centroids $\boldsymbol{\phi}_1, \cdots, \boldsymbol{\phi}_K$ for K groups, where $\boldsymbol{\phi}_k$ is d-dimension vector $(\phi_{k1}, \cdots, \phi_{kd})$, $k \in [K]$.
2. Repeat the following until the stopping criterion (Lloyd's steps):
   (a) For $i \in [n], k \in [K]$, compute the Euclidean distance between point $\mathbf{P}_i$ and centroids $\boldsymbol{\phi}_k$ by
   $$X_{ik} = \sqrt{\sum_{j=1}^{d} \left(P_{ij} - \phi_{kj}\right)^2}. \quad (1)$$
   (b) Assign each data point $\mathbf{P}_i$ to the closest cluster m_i for $i \in [n]$. This can be done by computing $k_i \leftarrow \arg\min\{X_{i1}, \cdots, X_{iK}\}$ firstly, and then generate a K-dimension one-hot vector $\boldsymbol{c}_i$ where '1' indicates the k_i-th component of vector $(X_{i1}, \cdots, X_{iK})$. We form $K \times n$ matrix $\mathbf{C}$ such that the i-th column of $\mathbf{C}$ is the one-hot vector $\boldsymbol{c}_i$. Let $\boldsymbol{m}_k$ be the k-th row of $\mathbf{C}$.

(c) Recalculate the average of the points in each cluster. For each cluster $k \in [K]$, compute new cluster center with

$$\boldsymbol{\varphi}_k = \frac{\boldsymbol{m}_k \cdot \mathbf{P}}{D_k}, \tag{2}$$

where $D_k = \sum_{i=1}^{n} m_{ki}$ is the point number of k-th cluster.

(d) Check the stopping criterion and update the new cluster center with the average. For each $k \in [K]$, compute the Euclidean distance between $\boldsymbol{\varphi}_k$ and $\boldsymbol{\phi}_k$ at first, and then the squared error can be computed by

$$e = \sum_{k=1}^{K} e_k = \sum_{k=1}^{K} \sqrt{\sum_{j=1}^{d} (\varphi_{kj} - \phi_{kj})^2}. \tag{3}$$

Given a small error ε, if $e \geq \varepsilon$, then update $\boldsymbol{\phi}_k$ with $\boldsymbol{\varphi}_k$. Otherwise, stop the criterion and output $\boldsymbol{\varphi}_k$.

3. Building Blocks

This section gives the building blocks for our privacy-preserving K-means clustering protocol.

3.1. Correlated Randomness

In order to generate randomness among parties without any interaction, similar to [28,31,32], correlated randomness is introduced to this work.

Let $F : \mathbb{Z}_2^\kappa \times \mathbb{Z}_2^\kappa \to \mathbb{Z}_{2^\ell}$ be a secure pseudo-random function (PRF). count is a counter maintained by the parties and updated after every PRF invocation. All parties run a one-time setup to establish PRF keys. The one-time setup can be done by letting each $\mathcal{P}_i$ choose a random key k_i, and then sending it to $\mathcal{P}_{i-1}$ for $i \in [3]$. Namely, each $\mathcal{P}_i$ has the random PRF keys k_i and k_{i-1} after the setup. In this way, the parties can generate the following correlated randomness locally:

- *3-out-of-3 randomness*: $\mathcal{P}_i$ holds $\alpha_i = F_{k_i}(\mathsf{count}) - F_{k_{i-1}}(\mathsf{count})$.
- *2-out-of-3 randomness*: $\mathcal{P}_i$ holds $(\alpha_i, \alpha_{i-1}) = (F_{k_i}(\mathsf{count}), F_{k_{i-1}}(\mathsf{count}))$.

Note that 3-out-of-3 randomness has the property that $\alpha_1 + \alpha_2 + \alpha_3 \equiv 0 \bmod 2^\ell$, which is known as *zero sharing*.

3.2. Replicated Secret Sharing

Replicated secret sharing (RSS) was first proposed by Ito et al. [33]. In CCS'16, Araki et al. [28] presented 2-out-of-3 replicated secret sharing scheme, which has high throughput and low latency. As a famous 3PC framework, ABY3 is also based on this variant over 2-power ring $\mathbb{Z}_{2^\ell}$ [31]. Our protocol also builds on ABY3. Let m be a general modulus, and we describe this replicated secret sharing as follows.

- $[\![x]\!] \leftarrow \mathsf{share}(x)$: To share a secret $x \in \mathbb{Z}_m$, the dealer samples three random values $x_1, x_2, x_3 \in_R \mathbb{Z}_m$ under the constraint that $x \equiv x_1 + x_2 + x_3 \bmod m$. For $i \in [3]$, $\mathcal{P}_i$ gets (x_i, x_{i+1}). We write $[\![x]\!] := (x_1, x_2, x_3)$.
- $x \leftarrow \mathsf{reconstruct}([\![x]\!], \mathcal{P})$: To reveal $[\![x]\!]$ to all parties, $\mathcal{P}_i$ sends x_i to $\mathcal{P}_{i+1}$, then each party reconstructs x locally by computing $x_1, x_2, x_3 \in_R \mathbb{Z}_m$. To reveal $[\![x]\!]$ only to $\mathcal{P}_i$, $\mathcal{P}_{i-1}$ sends x_{i-1} to $\mathcal{P}_{i-1}$ which reconstructs x locally.

In this work, m can be a different modulus. When $m = 2^\ell$, we call that *arithmetic share* and denote it as $[\![x]\!]$ or $[\![x]\!]^{\mathsf{A}}$. When $m = 2$, we call that *boolean share* and denote it as $[\![x]\!]^{\mathsf{B}}$.

The linear operation between two shares can be computed locally because of the linearity property. This means that given public constant a, b, c and two shares $[\![x]\!] = (x_1, x_2, x_3)$, $[\![y]\!] = (y_1, y_2, y_3)$, $[\![ax \pm by \pm c]\!]$ can be locally computed as $(ax_1 \pm by_1 \pm c, ax_2 \pm by_2, ax_3 \pm by_3)$. In order to compute the shares of multiplication $[\![z]\!] = [\![xy]\!]$, the

parties generate zero sharing locally at first, and then $\mathcal{P}_i$ locally computes 3-out-of-3 share $z_i = x_i y_i + x_{i+1} y_i + x_i y_{i+1} + \alpha_i$ for each $i \in [3]$. Finally, *resharing* is performed by the parties for 2-out-of-3 sharing semantics; this can be done by $\mathcal{P}_i$ that sends z_i to $\mathcal{P}_{i+1}$. It is easy to see that each party only sends 1 ring element per multiplication. Compare that with ASS in the 3PC setting, wherein RSS reduces 50% communication overhead. In this context, we denote RSS multiplication protocol as Π_{Mul}.

If the dealer wants to share a random value $r \in_R \mathbb{Z}_{2^\ell}$ for j-th time to all parties, the PRF key from zero sharing can be used. In particular, $\mathcal{P}_i$ lets $r_i = F_{k_i}(j)$ and $r_{i-1} = F_{k_{i-1}}(j)$. If $\mathcal{P}_1$ wants to share his private input x to all parties, the parties first generate another zero sharing $(\beta_1, \beta_2, \beta_3)$, then define the share of r as $[\![r]\!] := (\beta_1 + x, \beta_2, \beta_3)$, and $\mathcal{P}_i$ sends x to $\mathcal{P}_{i-1}$ in the end.

Note that the appearance of decimals in the computation is unavoidable in the computation, while secret sharing only works on the integer field. To represent a real number $\bar{x} \in \mathbb{R}$, we use a fixed-point representation with f-bit precision [34]. We scale $\bar{x}$ by a factor of 2^f and represent the rounded integer $x = \lfloor 2^f \cdot \bar{x} \rceil$ as a ℓ-bit signed integer over $\mathbb{Z}_{2^\ell}$. However, the multiplication result $[\![z']\!]$ between two shares $[\![x]\!]$ and $[\![y]\!]$ will have $2f$-bit precision.

To reduce precision from $2f$ to f, Mohassel and Rindal [31] introduce two probability truncation techniques to truncate the f-bit of the result. In this work, we use the second probability truncation technique. First, the parties generate a random truncation pair $([\![s']\!], [\![s]\!])$ in the preprocessing phase, where $s' \in \{0,1\}^\ell$ with $2f$-bit precision, $s \in \{0,1\}^f$ with f-bit precision, such that $s = s'/2^f$. In the online phase, the parties jointly compute $[\![z' - s']\!]$, then compute and open $(z' - s')$, which is followed by computing $[\![z]\!] = [\![s]\!] + (z' - s')/2^f$ locally. The truncation induces error is only 2^{-f}.

3.3. Oblivious Selection Protocol

As an important part of our privacy-preserving K-means clustering protocol, we define the oblivious selection functionality $\mathcal{F}_{\mathsf{OS}}$, whose functionality takes the arithmetic shares $[\![x]\!]$, $[\![y]\!]$ and boolean share $[\![b]\!]^{\mathsf{B}}$ as input, and returns $[\![x]\!]$ if $b = 0$, and $[\![y]\!]$ otherwise. Note that $\mathcal{F}_{\mathsf{OS}}$ can be explained as $f(x, y, b) = (1 - b)x + by = x + (y - x)b$. It seems that the parties only need to compute multiplication between $(y - x)$ and b once to implement $\mathcal{F}_{\mathsf{OS}}$. However, this is non-trivial since $[\![y - x]\!]^{\mathsf{A}}$ is arithmetic share and $[\![b]\!]^{\mathsf{B}}$ is boolean share, while the RSS multiplication only works for same shares.

A natural idea is to convert $[\![b]\!]^{\mathsf{B}}$ to $[\![b]\!]^{\mathsf{A}}$ by using bit injection protocol Π_{B2A} from ABY3 [31], where three-party OT is required. Instead of using OT, we implement conversion with the Beaver trick [35]. Suppose that the parties access the precomputed conversion, and obtain precomputed random bit share $[\![c]\!]^{\mathsf{B}}$ and $[\![c]\!]^{\mathsf{A}}$ in the preprocessing phase. In the online phase, the parties compute and reconstruct the bit $e = b \oplus c$, followed by setting $[\![d]\!]^{\mathsf{A}} = [\![1 - c]\!]^{\mathsf{A}}$ if $e = 1$, and $[\![d]\!]^{\mathsf{A}} = [\![c]\!]^{\mathsf{A}}$ otherwise. Finally, the parties compute $[\![z]\!]^{\mathsf{A}} = [\![(y - x) \cdot d]\!]^{\mathsf{A}} + [\![x]\!]^{\mathsf{A}}$, where $[\![(y - x) \cdot d]\!]^{\mathsf{A}}$ can be computed by using RSS multiplication Π_{Mul} between $[\![y - x]\!]^{\mathsf{A}}$ and $[\![x]\!]^{\mathsf{A}}$.

Since that random bit $[\![c]\!]^{\mathsf{B}}$ and $[\![c]\!]^{\mathsf{A}}$ are used as the one-time pad, the corrupted party can not learn any information about b. Even though the parties reveal the masked value e in the clear, our oblivious selection protocol is still secure. Looking ahead, the oblivious selection protocol is used to find the index of minimum component from vector and generate one-hot vector.

3.4. Secure Euclidean Squared Distance Protocol

From Section 2.3, it is important to compute the Euclidean distance between two points using Equation (1). However, the square root is unfriendly to construct the SS-based protocol since it is a nonlinear operation and has an expensive communication overhead. Note that $f(x) = \sqrt{x}$ is a monotonically increasing function, which can be replaced with $f(x) = x$. This would, however, not affect the results on clustering since the only thing we

need here is the relationship of size between two values. In this way, we replace Equation (1) with the following Euclidean squared distance equation:

$$X_{ik} = \sum_{j=1}^{d} \left(P_{ij} - \phi_{kj} \right)^2. \tag{4}$$

Similarly, Equation (3) can be replaced with the following Equation (5):

$$e = \sum_{k=1}^{K} e_k = \sum_{k=1}^{K} \sum_{j=1}^{d} (\varphi_{kj} - \phi_{kj})^2. \tag{5}$$

Now, we focus on, given the share of vector $\boldsymbol{x} = (x_1, \cdots, x_d)$ and $\boldsymbol{y} = (y_1, \cdots, y_d)$, how to compute the share of Euclidean squared distance. We define this functionality as $\mathcal{F}_{\mathsf{ESD}}$. Observe that this can be done by first computing $\boldsymbol{z} = \boldsymbol{x} - \boldsymbol{y} = (x_1 - y_1, \cdots, x_d - y_d)$, and then computing the inner product between $\boldsymbol{z}$ and $\boldsymbol{z}$. A naive way is for the parties to invoke RSS multiplication d times for d-dimension, consume d truncation pair to truncate the result, locally sum the result, and reshare. The communication is $O(d)$ ring elements, which is dependent on d, the size of vector.

In this work, we use the delay re-share technique [31] to reduce communication complexity for inner product, which is only communication $O(1)$ ring element and independent of d.

Let $(\llbracket s' \rrbracket, \llbracket s \rrbracket)$ be the shares of truncation pair among the parties, where $s' \in \{0,1\}^{\ell}$ with $2f$-bit precision, $s \in \{0,1\}^{f}$ with f-bit precision, such that $s = s'/2^f$. The delay re-share technique can be explained as the following Equation (6):

$$\llbracket \boldsymbol{x} \rrbracket \cdot \llbracket \boldsymbol{y} \rrbracket = \mathsf{reconstruct}((\sum_{j=1}^{d} \llbracket x_j \rrbracket \cdot \llbracket y_j \rrbracket) + \llbracket s' \rrbracket)/2^f - \llbracket s \rrbracket. \tag{6}$$

In a word, the parties first compute a 3-out-of-3 additive sharing of each $\llbracket x_i \rrbracket \llbracket y_i \rrbracket$ locally, then sum together, mask, truncate, and reshare the final result for 2-out-of-3 replicated sharing semantics. It is easy to see that this would only require communicate 1 ring element per party, which is independent of d. Furthermore, the truncation-induced error is only 2^{-f} with respect to the overall inner product.

The protocol for secure Euclidean squared distance is described in Figure 1.

3.5. Secure Comparison Protocol

In order to obtain the relationship between two given values, we have to consider how to implement comparison when the values are shared. We define secure comparison functionality $\mathcal{F}_{\mathsf{LT}}$, which takes $\llbracket x \rrbracket$ and $\llbracket y \rrbracket$ as input, return boolean share $\llbracket b \rrbracket^{\mathsf{B}}$, where bit $b = 1$ if $x < y$, and $b = 0$ otherwise.

Let $a = (x - y)$, then z can be computed by extracting the most significant bit (MSB) of a, i.e., $b = \mathsf{MSB}(a)$. Instead of using optimized parallel-prefix-adder-based bit extraction protocol from ABY3 [31], Wagh et al. [32] present a more efficient alternative method.

Recall that $\llbracket a \rrbracket := (a_1, a_2, a_3)$, and $a \equiv a_1 + a_2 + a_3 \bmod 2^{\ell}$, one has

$$b = \mathsf{MSB}(a) = \mathsf{MSB}(a_1) \oplus \mathsf{MSB}(a_2) \oplus \mathsf{MSB}(a_3) \oplus c, \tag{7}$$

where $c \in \{0,1\}$ is a carry bit from $(\ell - 2)$-th index and can be computed by Equation (8).

$$c = \begin{cases} 1, & \text{if } 2^{\ell-1} \leq a_1 + a_2 + a_3 < 2^{\ell}, \\ 0, & \text{Otherwise.} \end{cases} \tag{8}$$

From Equation (7), we observe that $\mathcal{P}_i$ can compute $\mathsf{MSB}(a_i)$ locally; thus, the main challenge here is how to compute c in a secure way. Note that Equation (8) is also equivalent to $c = ((2a_1 + 2a_2 + 2a_3) \geq_? 2^{\ell})$, which can be computed by wrap function. Wagh et al. [32]

give us a solution for wrap function, denoted as Π_{WA}. We refer the reader to their work for correctness and security.

The secure comparison protocol is described in Figure 2. Furthermore, if one of the secrets is known to all parties, e.g., y, the share of y can be defined by letting $[\![y]\!] := (y + \alpha_1, \alpha_2, \alpha_3)$ without any interaction, where $(\alpha_1, \alpha_2, \alpha_3)$ is zero sharing generated by PRF keys among the parties. Thus, the secure comparison protocol is also work. We denote this case as $[\![b]\!]^{\mathsf{B}} \leftarrow \mathcal{F}_{\mathsf{LT}}([\![x]\!], y)$.

$$[\![z]\!] \leftarrow \Pi_{\mathsf{SED}}([\![\boldsymbol{x}]\!], [\![\boldsymbol{y}]\!])$$

Input: The parties holds two vector shares $[\![\boldsymbol{x}]\!] = ([\![x_1]\!], \cdots, [\![x_d]\!])$ and $[\![\boldsymbol{y}]\!] = ([\![y_1]\!], \cdots, [\![y_d]\!])$ over $\mathbb{Z}_{2^\ell}$.
Output: The parties get share $[\![z]\!]$, where $z = (\boldsymbol{x} - \boldsymbol{y}) \cdot (\boldsymbol{x} - \boldsymbol{y}) = \sum_{j=1}^{d} (x_j - y_j)^2$.
Preprocessing:

1. The parties generate random truncation pair $([\![s']\!], [\![s]\!])$ by using (Mohassel and Rindal, Π_{trunc2}), where $s' \in \{0,1\}^\ell$ with $2f$-bit precision, $s \in \{0,1\}^f$ with f-bit precision, such that $s = s'/2^f$.

Online:

1. The parties locally compute the vector shares $[\![\boldsymbol{z}]\!] = ([\![z_1]\!], \cdots, [\![z_d]\!])$, where $[\![z_j]\!] = [\![x_i]\!] - [\![y_j]\!]$, $j \in [d]$.
2. The parties locally compute the 3-out-of-3 shares $[\![z']\!] = \sum_{j=1}^{d} [\![z_j]\!] \cdot [\![z_j]\!]$.
3. The parties locally mask the share $[\![z' + s']\!] = [\![z']\!] + [\![s']\!]$.
4. The parties reconstruct $(z' + s')$ and then locally compute $(z' + s')/2^f$.
5. The parties reshare and output the 2-out-of-3 shares $[\![z]\!] = (z' + s')/2^f - [\![s]\!]$.

Figure 1. Secure Euclidean squared distance protocol [31].

$$[\![b]\!]^{\mathsf{B}} \leftarrow \Pi_{\mathsf{LT}}([\![x]\!], [\![y]\!])$$

Input: The parties hold the arithmetic shares of x and y over $\mathbb{Z}_{2^\ell}$.
Output: The parties get the boolean share of bit $b = (x <_? y)$.
Online:

1. The parties locally compute the shares of a, where $a \leftarrow x - y$.
2. The parties compute the boolean share of bit c by invoking (Wagh et al., Π_{WA}), where $c \leftarrow ((2a_1 + 2a_2 + 2a_3) \geq_? 2^\ell)$.
3. Each $\mathcal{P}_i$ locally computes the boolean share of bit b, where $b \leftarrow \mathsf{MSB}(a_i) \oplus c$.
4. The parties output $[\![b]\!]^{\mathsf{B}}$.

Figure 2. Secure comparison protocol [32].

3.6. Secure Assignment Protocol

Recall that once the parties obtain the shares of the Euclidean squared distance between a point and all centorids, the following step is to assign this point to the closest cluster. This step can be abstracted as the question of, given K-dimension secret shared vector $[\![\boldsymbol{v}]\!] = ([\![v_1]\!], \cdots, [\![v_K]\!])$, how to compute the secret shared one-hot vector $\boldsymbol{e}$, where '1' appears in the k-th component, and $k = \arg\min\{v_1, \cdots, v_K\}$. We denote secure assignment functionality as $\mathcal{F}_{\mathsf{Assign}}$. The idea is straightforward. We implement secure assignment functionality with the following protocol Π_{Assign} (see Figure 3).

$$[\![\boldsymbol{e}]\!] \leftarrow \Pi_{\mathsf{Assign}}([\![\boldsymbol{v}]\!])$$

Input: The parties hold shares of vector $\boldsymbol{v} = (v_1, \cdots, v_K)$ over $\mathbb{Z}_{2^\ell}$.
Output: The parties get arithmetic shares of one-hot vector $\boldsymbol{e}_k = (e_1, \cdots, e_K)$, where $e_j = 0 \; \forall j \neq k$ and $e_k = 1$, and $k = \arg\min\{v_1, \cdots, v_K\}$.
Online:

1. Set $[\![v_{\min}]\!] \leftarrow [\![v_1]\!]$ and $[\![\boldsymbol{e}]\!] \leftarrow [\![\boldsymbol{e}_1]\!]$.
2. For $j = \{2, 3, \cdots, K\}$, compute as follows:
   (a) Compute $[\![b]\!]^{\mathsf{B}} \leftarrow \Pi_{\mathsf{LT}}([\![v_j]\!], [\![v_{\min}]\!])$;
   (b) Compute $[\![v_{\min}]\!] \leftarrow \Pi_{\mathsf{OS}}([\![v_{\min}]\!], [\![v_j]\!], [\![b]\!]^{\mathsf{B}})$;
   (c) Compute $[\![\boldsymbol{e}]\!] \leftarrow \Pi_{\mathsf{OS}}([\![\boldsymbol{e}]\!], [\![\boldsymbol{e}_j]\!], [\![b]\!]^{\mathsf{B}})$;
3. Output $[\![\boldsymbol{e}]\!]$.

Figure 3. Secure Assignment protocol.

3.7. Secure Division Protocol

As shown in Section 2.3, the parties need to recalculate the average vector of the points in each cluster in a secure way. Note that the average can be split by computing addition and division, where addition can be computed locally, hence the key point is to compute division. If the divisor is known to all parties, then division can be computed locally. However, observe that the divisor denotes the number of each cluster point and should be protected since full data privacy is required. Thus, the computation of division becomes difficult in our scenarios. We define secure division functionality $\mathcal{F}_{\mathsf{Div}}$ as follows: Given secret shared value $[\![a]\!]$ and $[\![b]\!]$ with $b \in \mathbb{Z}^+$, the parties compute the share $[\![c]\!]$, such that $c = a/b$.

Instead of invoking division garbled circuit protocol, we implement division with numerical method. The numerical method is one of the most commonly used techniques for constructing SS-based secure protocol due to its efficiency. In this work, we implement division using Goldschmidt's algorithm [36], which approximates the desired operation as a series of multiplication.

Let w_0 be an initial approximation of $1/b$, and $\epsilon_0 := 1 - b \cdot w_0$ be the relative error for the approximation w_0 such that $|\epsilon_0| < 1$. The Goldschmidt algorithm iteratively computes the following Equation (9):

$$c = \frac{a}{b} = a \cdot \frac{1}{b} \approx a \cdot w_0(1 + \epsilon_0)(1 + \epsilon_0^2) \cdots (1 + \epsilon_0^{2^{t-1}}), \tag{9}$$

where t is the number of iterations. When $t \to +\infty$, one has c converges to a/b [36]. We set $t = 2$, which is sufficient for a close approximation with our choice of fixed-point precision.

Catrina and Saxena [34] give us a good initial approximation of w_0 in the interval $[0.5, 1)$, that is $w_0 = 2.9142 - 2b$. However, the major challenge of our work is that $b \in \mathbb{Z}^+$ does not belong to the interval $[0.5, 1)$. In this work, we use the technique proposed by Wagh et al. [32]. The key insight here is that b is interpreted as a value with $(\alpha + 1)$-bit fixed-point precision but not f-bit precision, where $\alpha \in \mathbb{Z}$ such that $2^\alpha \leq b < 2^{\alpha+1}$. Thus, one should first extract α. The secure division protocol is described in Figure 4. From step 1 and step 2, we extract and reveal α to all parties, which only leaks the range of b and nothing else.

$$[\![c]\!] \leftarrow \Pi_{\mathsf{Div}}([\![a]\!], [\![b]\!])$$

Input: The parties hold arithmetic shares of a, b over $\mathbb{Z}_{2^\ell}$.
Output: The parties get the arithmetic share of $c = a/b$.
Online:

1. Initialize $\alpha \leftarrow 0$.
2. For $i = \{\ell - 1, \cdots, 1, 0\}$, compute as follows:
   (a) Compute $[\![d]\!]^{\mathsf{B}} \leftarrow \Pi_{\mathsf{LT}}([\![b]\!], [\![2^{2^i + \alpha}]\!])$, and reveal d to all parties.
   (b) Set $\alpha \leftarrow \alpha + 2^i$ if $d = 1$.

   The parties get α in the clear, where $2^\alpha \leq b < 2^{\alpha+1}$.
3. Set fixed-point precision as $f' = \alpha + 1$.
4. Compute the shares of $w_0 \leftarrow 2.9142 - 2b$.
5. Compute the shares of $\epsilon_0 \leftarrow 1 - b \cdot w_0$ and $\epsilon_1 \leftarrow \epsilon_0^2$.
6. Compute the shares of $c \leftarrow aw_0(1 + \epsilon_0)(1 + \epsilon_1)$.
7. Output $[\![c]\!]$ over $\mathbb{Z}_{2^\ell}$ with f-bit precision.

Figure 4. Secure division protocol [32].

4. Privacy-Preserving *K*-Means Clustering

We now give a formal description of our privacy-preserving K-means clustering protocol, following the basic building blocks outlined above.

4.1. Secret Distribution

Recall that all data are held by the data owners in the secure outsourced scenarios, thus secret distribution phase is completed by the data owner. This implies that all data are *horizontal partitioned*. As an optimized, instead of generating 2-out-of-3 replicated shares directly, the data owner generates 3-out-of-3 additive shares and then sends to three computing parties/servers, who reshare the shares and obtain valid 2-out-of-3 replicated shares. In this way, we reduce communication costs of the data owner by a half.

4.2. Cluster Initialization

As shown in Section 2.3, we need to initialize K centroids before the Lloyd's steps. In this work, we assume that the data owner chooses K random points as the initialized centroids and secret share to three computing parties for simple. In this way, cluster initialization can be combined with secret distribution.

4.3. Lloyd's Steps

4.3.1. Approximation of Euclidean Distance

Recall that we replace Euclidean distance with Euclidean squared distance, which is not affect the final result as we shown in Section 3.4. For $i \in [n]$ and $k \in [K]$, the parties first invoke $\mathcal{F}_{\mathsf{SED}}$ to compute the shares of Euclidean squared distance X_{ik} between data point $\mathbf{P}_i$ and centroid $\boldsymbol{\phi}_k$, and then form $n \times K$ matrix $\mathbf{X}$.

4.3.2. Assigning Data Points to the Closest Cluster

For $i \in [n]$, we denote $\mathbf{X}_i$ as the i-th row vector of $\mathbf{X}$. In order to assign data point $\mathbf{P}_i$ to the closet cluster, the parties invoke our $\mathcal{F}_{\mathsf{Assign}}$ protocol as described in Section 3.6. The one-hot vector output $[\![\boldsymbol{c}_i]\!] \leftarrow \mathcal{F}_{\mathsf{Assign}}([\![\mathbf{X}_i]\!])$ indicates which cluster center this data point is assigned to. We form $K \times n$ cluster matrix $[\![\mathbf{C}]\!]$, such that the i-th column of $\mathbf{C}$ is $\boldsymbol{c}_i$.

4.3.3. Recalculating Cluster Centers

Given cluster matrix $\mathbf{C}$, the parties need to recalculate each centroid. Observe that for each $k \in [K]$, the k-th cluster center has $D_k = \sum_{i=1}^{n} \mathbf{C}_{ki}$ points exactly. Instead of computing $\boldsymbol{\varphi}_k = \frac{\mathbf{C}_k \cdot \mathbf{P}}{D_k}$ separately, one can first compute $\mathbf{M} = \mathbf{C} \cdot \mathbf{P}$, and then compute $\boldsymbol{\varphi}_k = \frac{\mathbf{M}_k}{D_k}$, where

$\mathbf{M}_k$ is the k-th row of $\mathbf{M}$. Given secret shared matrix $[\![\mathbf{C}]\!]$, the new centroid matrix $\boldsymbol{\varphi}$ can be computed by vectorized multiplication technique [31] and secure division protocol (as described in Section 3.7). Furthermore, the parties also compute the shares of Euclidean squared distance e_k between $\boldsymbol{\varphi}_k$ and $\boldsymbol{\phi}_k$ for checking the stopping criterion.

4.3.4. Checking the Stopping Criterion and Updating Centroids

In order to check the stopping criterion, the parties first locally compute $e = \sum_{k=1}^{K} e_k$, and then compare with a given small error ε, stop the criterion if $e < \varepsilon$, otherwise update $\boldsymbol{\varphi}$ and continue next round. This can be done by invoking secure comparison protocol Π_{LT}. The only reveal message is $b = (e <_? \varepsilon)$, which does not affect full data privacy.

4.4. Main Construction

The secure K-means protocol is described in Figure 5. According to the definition of full data privacy, the information about the intermediate centroids, cluster assignments, and cluster sizes should be protected. From Figure 5, we can see that the only information we leak is the range of D_k and nothing else. Therefore, our construction provides full data privacy.

Parameters:

- Number of clusters K; number of data points n; dimension d.
- Ideal $\mathcal{F}_{\mathsf{OS}}, \mathcal{F}_{\mathsf{SED}}, \mathcal{F}_{\mathsf{LT}}, \mathcal{F}_{\mathsf{Assign}}, \mathcal{F}_{\mathsf{Div}}$ primitives.

Secret Distribution:

1. The data owner generates the 3-out-of-3 additive shares of data points matrix $\mathbf{P} = (\mathbf{P}_1, \cdots, \mathbf{P}_n)$ with dimension $n \times d$.
2. The data owner sends the shares to three semi-honest computing parties/servers $\mathcal{P}_1, \mathcal{P}_2, \mathcal{P}_3$.
3. Each party reshares the shares and obtains valid 2-out-of-3 replicated shares.

Initialization:

1. The data owner chooses K random points as the initialized centroids, forms $K \times d$ centroid matrix $\boldsymbol{\phi} = (\boldsymbol{\phi}_1, \cdots, \boldsymbol{\phi}_K)$.
2. The data owner shares $\boldsymbol{\phi}$ to three computing parties similar to secret distribution.

Lloyd's Steps:

For $t \in [T]$, the parties repeat the following steps until the stopping criterion.

1. For $i \in [n], k \in [K]$, the parties compute the shares of the Euclidean squared distance by invoking $[\![X_{ik}]\!] \leftarrow \mathcal{F}_{\mathsf{SED}}([\![\mathbf{P}_i]\!], [\![\boldsymbol{\phi}_k]\!])$. The parties form $n \times K$ matrix $\mathbf{X}$.
2. For $i \in [n]$, the parties assign point $\mathbf{P}_i$ to the closest cluster by invoking $[\![\boldsymbol{c}_i]\!] \leftarrow \mathcal{F}_{\mathsf{Assign}}([\![\mathbf{X}_i]\!])$. The parties forms $K \times n$ matrix $[\![\mathbf{C}]\!]$ such that the i-th column of $\mathbf{C}$ is $\boldsymbol{c}_i$.
3. The parties compute the share of matrix multiplication $[\![\mathbf{M}]\!] \leftarrow [\![\mathbf{C}]\!] \cdot [\![\mathbf{P}]\!]$ by using RSS vectorized multiplication technique. For $k \in [K]$, the parties jointly recalculate the sharing $[\![\boldsymbol{\varphi}_k]\!]$ of the new centroid as follows:
   (a) Compute the shares of the denominator $[\![D_k]\!] \leftarrow \sum_{i=1}^{n} [\![\mathbf{C}_{ki}]\!]$.
   (b) Compute the shares of the average of the points in each cluster by invoking $[\![\boldsymbol{\varphi}_k]\!] \leftarrow \mathcal{F}_{\mathsf{Div}}([\![\mathbf{M}_k]\!], [\![D_k]\!])$.
   (c) Compute the shares of Euclidean squared distance between $\boldsymbol{\varphi}_k$ and $\boldsymbol{\phi}_k$ by invoking $[\![e_k]\!] \leftarrow \mathcal{F}_{\mathsf{SED}}([\![\boldsymbol{\varphi}_k]\!], [\![\boldsymbol{\phi}_k]\!])$.
4. The parties update the cluster centroids as follows:
   (a) The parties locally compute the sharing of error $[\![e]\!] \leftarrow \sum_{i=1}^{K} [\![e_k]\!]$.
   (b) The parties invoke $[\![b]\!]^{\mathsf{B}} \leftarrow \mathcal{F}_{\mathsf{LT}}([\![e]\!], \varepsilon)$ and reveal b.
   (c) If $b = 1$, then stop criterion and reveal $\boldsymbol{\varphi}$. Otherwise, replace $[\![\boldsymbol{\phi}]\!]$ with $[\![\boldsymbol{\varphi}]\!]$.

Figure 5. Our Privacy-preserving K-means Protocol.

5. Security Analyses

Our protocol follows the universally composable framework [29] and provides security against a single corrupted party under the semi-honest model. The universally composable framework guarantees the security of arbitrary composition of different protocols. Therefore, we only need to prove the security of individual protocols.

Theorem 1. $\Pi_{\mathsf{OS}}, \Pi_{\mathsf{SED}}, \Pi_{\mathsf{LT}}, \Pi_{\mathsf{Assign}}, \Pi_{\mathsf{Div}}$ *securely realizes* $\mathcal{F}_{\mathsf{OS}}, \mathcal{F}_{\mathsf{SED}}, \mathcal{F}_{\mathsf{LT}}, \mathcal{F}_{\mathsf{Assign}}, \mathcal{F}_{\mathsf{Div}}$, *respectively, in the presence of one semi-honest corrupt party under the hybrid model.*

Proof of Theorem 1. We list the security of those protocols as follows:

Security for Π_{OS}: This can be reduced to the security of one-time pad and RSS multiplication Π_{Mul}.

Security for Π_{SED}: This protocol is based on the vectorized multiplication protocol [31], which has been proven secure under the semi-honest model.

Security for Π_{LT}: This can be reduced to the security of wrap function protocol, which has been proven secure in [32].

Security for Π_{Assign}: This can be reduced to the security of Π_{LT} and Π_{OS}.

Security for Π_{Div}: This protocol has been proven secure in [32]. □

Theorem 2. *The protocol in Figure 5 securely computes the K-means clustering under the semi-honest model, given the ideal* $\mathcal{F}_{\mathsf{SED}}, \mathcal{F}_{\mathsf{Assign}}, \mathcal{F}_{\mathsf{Mul}}, \mathcal{F}_{\mathsf{LT}}$, *and* $\mathcal{F}_{\mathsf{Div}}$ *functionalities, respectively.*

Proof of Theorem 2. We exhibit a simulator Sim for simulating a corrupt party $\mathcal{P}_1$. The simulator for $\mathcal{P}_2$ and $\mathcal{P}_3$ should be the same as $\mathcal{P}_1$.

Sim simulates the view of corrupt $\mathcal{P}_1$, which consists of his input/output and received messages, and proceeds as follows:

1. Calls $\mathcal{F}_{\mathsf{SED}}$ simulator $\mathsf{Sim}_{\mathcal{F}_{\mathsf{SED}}}(\llbracket \mathbf{P}_i \rrbracket, \llbracket \boldsymbol{\phi}_k \rrbracket)$ to simulate step 1, then appends its output to the view;
2. Calls $\mathcal{F}_{\mathsf{Assign}}$ simulator $\mathsf{Sim}_{\mathcal{F}_{\mathsf{Assign}}}(\llbracket \mathbf{X}_i \rrbracket$ to simulate step 2, then appends its output to the view;
3. Calls $\mathcal{F}_{\mathsf{Mul}}$ simulator $\mathsf{Sim}_{\mathcal{F}_{\mathsf{Mul}}}(\llbracket \mathbf{C} \rrbracket, \llbracket \mathbf{P} \rrbracket)$ to simulate step 3, then appends its output to the view;
4. Calls $\mathcal{F}_{\mathsf{Div}}$ simulator $\mathsf{Sim}_{\mathcal{F}_{\mathsf{Div}}}(\llbracket \mathbf{M}_k \rrbracket, \llbracket D_k \rrbracket)$ to simulate step 3(b), then appends its output to the view.
5. Calls $\mathcal{F}_{\mathsf{LT}}$ simulator $\mathsf{Sim}_{\mathcal{F}_{\mathsf{LT}}}(\llbracket e \rrbracket, \varepsilon)$ to simulate step 4(b), then appends its output to the view.

We argue the indistinguishability of the produced transcript from the execution of real world. At first, we formally show the simulation by proceeding the sequence of hybrid transcripts T_0, T_1, T_2, T_3, T_4, T_5, where T_0 is the real view of $\mathcal{A}$, and T_5 is the output of Sim.

Hybrid 1. Let T_1 be the same as T_0, except the $\mathcal{F}_{\mathsf{SED}}$ execution is replaced with running the simulator $\mathsf{Sim}_{\mathcal{F}_{\mathsf{SED}}}(\llbracket \mathbf{P}_i \rrbracket, \llbracket \boldsymbol{\phi}_k \rrbracket)$. Because Π_{SED} has been proven secure, thus $\mathsf{Sim}_{\mathcal{F}_{\mathsf{SED}}}$ is guaranteed to produce output indistinguishable from the execution of real world. Therefore, T_1 and T_0 are indistinguishable.

Hybrid 2. Let T_2 be the same as T_1, except the $\mathcal{F}_{\mathsf{Assign}}$ execution is replaced with running the simulator $\mathsf{Sim}_{\mathcal{F}_{\mathsf{Assign}}}(\llbracket \mathbf{X}_i \rrbracket)$. This functionality outputs the share of the Euclidean squared distance between $\mathbf{P}_i$ and $\boldsymbol{\phi}_k$, which does not reveal any information about the result. Moreover, the output of $\mathsf{Sim}_{\mathcal{F}_{\mathsf{Assign}}}$ is indistinguishable from the execution of real world, thus T_2 and T_1 are indistinguishable.

Hybrid 3. Let T_3 be the same as T_2, except the $\mathcal{F}_{\mathsf{Mul}}$ execution is replaced with running the simulator $\mathsf{Sim}_{\mathcal{F}_{\mathsf{Mul}}}(\llbracket \mathbf{C} \rrbracket, \llbracket \mathbf{P} \rrbracket)$. Because Π_{Mul} has been proven secure, $\mathsf{Sim}_{\mathcal{F}_{\mathsf{Mul}}}$ is guaranteed to produce output indistinguishable from the execution of the real world. Therefore, T_3 and T_2 are indistinguishable.

Hybrid 4. Let T_4 be the same as T_3, except the $\mathcal{F}_{\mathsf{Div}}$ execution is replaced with running the simulator $\mathsf{Sim}_{\mathcal{F}_{\mathsf{Div}}}(\llbracket \mathbf{M}_k \rrbracket, \llbracket D_k \rrbracket)$. This is because Π_{Div} has been proven secure, and its

output is the share, which is indistinguishable from the pseudo-randomness. In other words, the output of $\mathsf{Sim}_{\mathcal{F}_{\mathsf{Div}}}$ is indistinguishable from the execution of real world. Thus, T_4 and T_3 are indistinguishable.

Hybrid 5. Let T_5 be the same as T_4, except the $\mathcal{F}_{\mathsf{LT}}$ execution is replaced with running the simulator $\mathsf{Sim}_{\mathcal{F}_{\mathsf{LT}}}(\llbracket e \rrbracket, \varepsilon)$. The output is the share, which does not reveal any information about the data points. In addition, the output of $\mathsf{Sim}_{\mathcal{F}_{\mathsf{LT}}}$ is indistinguishable from the execution of real world. Thus T_5 and T_4 are indistinguishable.

In summary, T_0 and T_5 are indistinguishable, which is end of the proof. □

6. Experiments

6.1. Experimental Setup

We implement our privacy-preserving clustering protocol and report the experimental results in this section. All experiments are executed on Ubuntu 22.04 LTS with Intel(R) Xeon(R) Gold 5222 CPU @3.41GHz and 256 GB RAM. All parties run in the same network and the connection is simulated using the Linux `tc` command. The LAN setting has 0.2 ms round-trip latency and 5 Gbps network bandwidth, while the WAN setting has 20 ms round-trip latency and 400 Mbps network bandwidth. We implement our protocol with the `C++` open source framework FALCON (https://github.com/snwagh/falcon-public (accessed on 18 June 2022)) [32].

In all experiments, we assume the orginal data has been normalized in the same level. For the public parameters, we set bit-length $\ell = 64$, fixed-point precision $f = 13$, and the number of iteration $t = 2$. Table 2 summarizes the real datasets used in our experiments. Furthermore, we also compare with Mohassel et al. [27] in the self-generated dataset.

Table 2. Descriptions of the datasets we used in experiments, where n is the number of data points, K is the number of clusters, and d is the dimension. We also report the accuracy of different datasets, if the ground truth model of dataset exists.

Dataset	n	K	d	Accuracy
Iris	150	3	4	92.67%
arff	1000	4	2	98.20%
Self-generated	$\{10,000, 100,000\}$	$\{2,5\}$	$\{5,10,15,20\}$	—

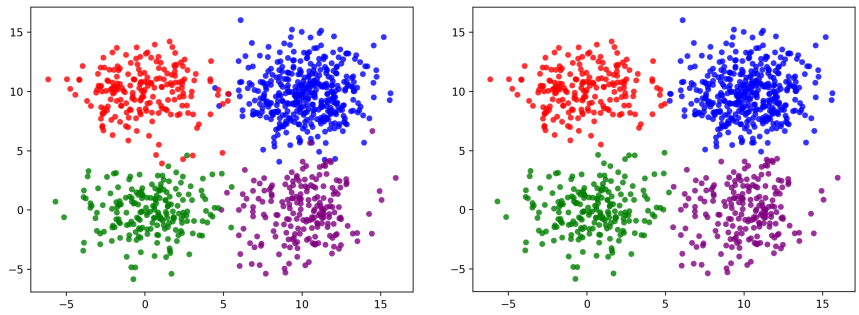
6.2. Accuracy

In order to evaluate the accuracy of clustering classification for the real dataset, we usually compare it to the ground truth model. We downloaded the ground truth model of Iris and arff from Github repository (https://github.com/deric/clustering-benchmark (accessed on 18 June 2022)). Note that the standard K-means clustering is sensitive to the initialized centroids; thus, we ran the algorithm many times with different initialized centroids and take the best result as the global optimal solution.

We used 2D dataset arff and 4D dataset Iris for evaluating accuracy and report the experimental results in Table 2. For a visual comparison, the experimental result of dataset arff is shown in Figure 6. Compared to the ground truth model of dataset arff, we reached 98.20% accuracy in our privacy-preserving model. For dataset Iris, we reached 92.67% accuracy in our privacy-preserving model. Both accuracy results are the same as the plaintext algorithm; hence, our privacy-preserving protocol is feasible.

Recall that our secure division protocol adopts the numerical method and the result of the secure multiplication protocol needs to be truncated; thus, the privacy-preserving result $\boldsymbol{\varphi}$ is only approximate to the plain result but not the exact result. In fact, our experiment shows that the relative error is about 10^{-2}, which means this part has a negligible impact on model accuracy compare with the plaintext algorithm. In other words, our privacy-preserving protocol is feasible, even if $\boldsymbol{\varphi}$ is only approximate to the truth value. For the large-scale dataset, we argue that the relative error can be improved by taking bigger public parameters ℓ, f, and t. However, it will require more runtime and communication cost.

There is a trade-off between runtime and accuracy of the division. We do not consider the accuracy of experiment in the following section.



(**a**) Ground Truth Model (**b**) Plaintext and Privacy-preserving Model

Figure 6. Comparison of accuracy for ground truth, plaintext, and privacy-preserving model for 2D dataset arff. Our privacy-preserving model reaches the same accuracy as the plaintext model. The accuracy is 98.20% compared to the ground truth model.

6.3. Runtime and Communication

In this section, we focus on the total communication cost and runtime of our privacy-preserving protocol. We ran each experiment five times and computed the average of wall clock runtime as the reported runtime. We report the experimental results in Table 3. Note that iteration T depends on the initialized clustering centroids; hence, we set it to a fixed value in this experiment (say, $T = 10$).

Table 3. The comparison of wall clock runtime and communication cost with different dimensions in the self-generated dataset both with LAN and WAN setting, where n is the number of data points, K is the number of clusters, d is the dimension, and the iteration $T = 10$.

Parameters			Runtime		Comm. (MB)
n	K	d	LAN (s)	WAN (min)	
10,000	2	5	63.8160	134.3737	37.1516
		10	63.8020	134.3832	37.2976
		20	63.6132	134.3962	37.5896
	5	5	160.9406	336.0150	134.2790
		10	161.2164	336.1254	134.6440
		20	161.3586	336.2652	135.3740
100,000	2	5	474.7150	1336.1333	370.1520
		10	473.9687	1336.1968	370.2980
		20	475.1037	1336.2415	370.5900

As shown in Table 3, the runtime and communication overhead are independent from the dimension d; this is because we enjoy the benefit from the vectorized multiplication and delay re-share technique [31].

Even though $n = 100,000$, our scheme lasted less than 8 min and communicated less than 400 MB in total under the LAN setting. Although the overall communication cost is low, we observe that the runtime of our scheme is not good at the WAN setting. The reason for this is because WAN has low-bandwidth and high-latency, while the secret-sharing-based schemes usually have much communication rounds that is bad for this setting. Thus, we argue that our privacy-preserving scheme is practical when the network is fast, high-bandwidth, and low-latency. We estimate that our privacy-preserving protocol can be used to deal with datasets of million points in an acceptable time (for example, within 2 h for clustering one million points to 2 groups in the LAN setting).

6.4. Comparison with Mohassel et al. [27]

Recall that Mohassel et al. [27] also provides full data privacy guarantees (see Table 1); thus, we also compare to their work with the self-generated dataset in our experiment environment. We downloaded the code of Mohassel et al. [27] from their Github repository (https://github.com/osu-crypto/secure-kmean-clustering (accessed on 18 June 2022)). In order to save time, all experiments are only considered under the localhost setting, which has 0.027ms round-trip latency and 41.1Gbps network bandwidth. We ran each protocol five times and report the average of wall clock runtime and communication costs. Instead of running the code of Mohassel et al. [27] in every iteration, we measured its runtime for one round iteration and multiplied by the number of iterations T to save time.

Table 4 presents the computation cost and communication cost of our protocol compared with [27]. Our experimental results demonstrate that the computation costs of our protocol is about 16.5×–25.2× faster than [27], and the communication cost is about 63.8×–68.0× less than [27]. This is because their construction relies heavily on garbled circuit and oblivious transfer, while our scheme is only based on replicated secret sharing.

Table 4. Comparison with Mohassel et al. [27] in large-scale self-generated datasets under the localhost setting, where n is the number of data points, K is the number of clusters, and dimension $d = 2$.

Parameters			Runtime (min)			Communication (MB)		
n	K	T	[27]	This Work	Improved Factor	[27]	This Work	Improved Factor
10^4	2	10	1.77	**0.09**	19.5×	2377	**37**	64.1×
		20	3.36	**0.19**	17.9×	4733	**74**	63.8×
	5	10	4.69	**0.28**	16.5×	9121	**134**	68.0×
		20	9.46	**0.56**	16.9×	18220	**268**	68.0×
10^5	2	10	15.33	**0.74**	20.8×	23731	**370**	64.1×
		20	29.74	**1.15**	20.4×	47262	**740**	63.9×
	5	10	46.51	**1.85**	25.2×	91128	**1339**	68.0×
		20	91.61	**3.65**	25.1×	181867	**2678**	67.9×

7. Conclusions and Future Work

In this work, we presented an efficient RSS-based privacy-preserving K-means clustering scheme over $\mathbb{Z}_{2^\ell}$ under the semi-honest model. Our scheme provides full data privacy. The experiment report shows that our protocol is highly efficient and practical, as well as suitable for large-scale clustering tasks when the network is fast. Therefore, we argue that our scheme is suitable for secret-sharing-based secure outsourced computation.

The next direction of future work can extend our scheme to the malicious adversarial setting, which will be a non-trivial problem. This is because malicious adversary may not follow protocol specifying and deviate arbitrarily in any phase. For example, the adversary makes the corrupted party sends incorrect messages, such that it can break the correctness of protocol. Therefore, we should ensure that the sending message of the parties are correct. A promising direction for this case is to introduce the SPDZ protocol [37], where the correctness of the sending message can be protected by using message authentication codes (MACs).

Author Contributions: Conceptualization, W.W. and C.T.; methodology, W.W. and C.T.; software, W.W.; supervision, C.T.; writing—original draft preparation, W.W. and Y.C.; writing—review and editing, W.W. and Y.C. All authors have read and agreed to the published version of the manuscript.

Funding: This research was supported by National Natural Science Foundation of China under Grant No. 61772147.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

Acknowledgments: The authors would like to thank all anonymous reviewers for their comments.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Lindell, Y.; Pinkas, B. Privacy Preserving Data Mining. *J. Cryptol.* **2002**, *15*, 177–206. [CrossRef]
2. Tan, P.N.; Steinbach, M.; Karpatne, A.; Kumar, V. *Introduction to Data Mining*, 2nd ed.; Pearson: New York, NY, USA, 2018.
3. Hegde, A.; Möllering, H.; Schneider, T.; Yalame, H. SoK: Efficient Privacy-Preserving Clustering. *Proc. Priv. Enhancing Technol.* **2021**, *2021*, 225–248. [CrossRef]
4. Chen, Y.; Tang, C.; Ye, R. Cryptanalysis and Improvement of Medical Image Encryption Using High-Speed Scrambling and Pixel Adaptive Diffusion. *Signal Process.* **2020**, *167*, 107286. [CrossRef]
5. Rivest, R.L.; Adleman, L.; Dertouzos, M.L. On data banks and privacy homomorphisms. *Found. Secur. Comput.* **1978**, *4*, 169–180.
6. Gentry, C. A Fully Homomorphic Encryption Scheme. Ph.D. Thesis, Stanford University, Stanford, CA, USA, 2009.
7. Paillier, P. Public-Key Cryptosystems Based on Composite Degree Residuosity Classes. In Proceedings of the Advances in Cryptology—EUROCRYPT '99, Santa Barbara, CA, USA, 15–19 August 1999; pp. 223–238.
8. Elgamal, T. A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms. *IEEE Trans. Inf. Theory* **1985**, *31*, 469–472. [CrossRef]
9. Vaidya, J.; Clifton, C. Privacy-Preserving k-Means Clustering over Vertically Partitioned Data. In Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Washington, DC, USA, 24–27 August 2003; pp. 206–215.
10. Bunn, P.; Ostrovsky, R. Secure Two-Party k-Means Clustering. In Proceedings of the 14th ACM Conference on Computer and Communications Security—CCS '07, Alexandria, VA, USA, 28–31 October 2007; p. 486.
11. Rao, F.Y.; Samanthula, B.K.; Bertino, E.; Yi, X.; Liu, D. Privacy-Preserving and Outsourced Multi-User K-Means Clustering. In Proceedings of the 2015 IEEE Conference on Collaboration and Internet Computing (CIC), Hangzhou, China, 27–30 October 2015; pp. 80–89.
12. Kim, H.J.; Chang, J.W. A Privacy-Preserving k-Means Clustering Algorithm Using Secure Comparison Protocol and Density-Based Center Point Selection. In Proceedings of the 2018 IEEE 11th International Conference on Cloud Computing (CLOUD), San Francisco, CA, USA, 2–7 July 2018; pp. 928–931.
13. Jäschke, A.; Armknecht, F. Unsupervised Machine Learning on Encrypted Data. In Proceedings of the International Conference on Selected Areas in Cryptography, Calgary, AB, Canada, 15–17 August 2018; pp. 453–478.
14. Chillotti, I.; Gama, N.; Georgieva, M.; Izabachène, M. Faster Fully Homomorphic Encryption: Bootstrapping in Less Than 0.1 Seconds. In Proceedings of the International Conference on the Theory and Application of Cryptology and Information Security, Hanoi, Vietnam, 4–8 December 2016; pp. 3–33.
15. Cai, Y.; Tang, C. Privacy of Outsourced Two-party K-means Clustering. *Concurr. Comput. Pract. Exp.* **2021**, *33*, e5473. [CrossRef]
16. Liu, D. Practical Fully Homomorphic Encryption without Noise Reduction. *IACR Cryptol. ePrint Arch.* **2015**, *2015*, 468.
17. Wang, Y. Notes on Two Fully Homomorphic Encryption Schemes without Bootstrapping. Cryptology ePrint Archive, Report 2015/519. 2015. Available online: https://eprint.iacr.org/2015/519 (accessed on 18 June 2022).
18. Yao, A. Protocols for Secure Computations. In Proceedings of the FOCS, Chicago, IL, USA, 3–5 November 1982.
19. Goldreich, O.; Micali, S.; Wigderson, A. How to Play Any Mental Game. In Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing, New York, NY, USA, 25–27 May 1987; pp. 218–229.
20. Yao, A.C.C. How to Generate and Exchange Secrets. In Proceedings of the 27th Annual Symposium on Foundations of Computer Science, Toronto, ON, Canada, 27–29 October 1986; pp. 162–167.
21. Shamir, A. How to Share a Secret. *Commun. ACM* **1979**, *22*, 612–613. [CrossRef]
22. Doganay, M.C.; Pedersen, T.B.; Saygin, Y.; Savaş, E.; Levi, A. Distributed Privacy Preserving K-Means Clustering with Additive Secret Sharing. In Proceedings of the 2008 International Workshop on Privacy and Anonymity in Information Society, Nantes, France, 29 March 2008; pp. 3–11.
23. Patel, S.; Garasia, S.; Jinwala, D. An Efficient Approach for Privacy Preserving Distributed K-Means Clustering Based on Shamir's Secret Sharing Scheme. In Proceedings of the IFIP International Conference on Trust Management, Surat, India, 21–25 May 2012; pp. 129–141.
24. Patel, S.; Patel, V.; Jinwala, D. Privacy Preserving Distributed K-Means Clustering in Malicious Model Using Zero Knowledge Proof. In Proceedings of the International Conference on Distributed Computing and Internet Technology, Bhubaneswar, India, 5–8 February 2013; pp. 420–431.
25. Upmanyu, M.; Namboodiri, A.M.; Srinathan, K.; Jawahar, C.V. Efficient Privacy Preserving K-Means Clustering. In Proceedings of the Pacific-Asia Workshop on Intelligence and Security Informatics, Hyderabad, India, 21 June 2010; pp. 154–166.

26. Baby, V.; Chandra, N.S. Distributed Threshold K-Means Clustering for Privacy Preserving Data Mining. In Proceedings of the 2016 International Conference on Advances in Computing, Communications and Informatics (ICACCI), Jaipur, India, 21–24 September 2016; pp. 2286–2289.
27. Mohassel, P.; Rosulek, M.; Trieu, N. Practical Privacy-Preserving k-Means Clustering. *Proc. Priv. Enhancing Technol.* **2020**, *2020*, 414–433. [CrossRef]
28. Araki, T.; Furukawa, J.; Lindell, Y.; Nof, A.; Ohara, K. High-Throughput Semi-Honest Secure Three-Party Computation with an Honest Majority. In Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, 24–28 October 2016; pp. 805–817.
29. Canetti, R. Universally Composable Security: A New Paradigm for Cryptographic Protocols. In Proceedings of the 42nd IEEE Symposium on Foundations of Computer Science, Washington, DC, USA, 14–17 October 2001; pp. 136–145.
30. Goldreich, O. *The Foundations of Cryptography—Volume 2: Basic Applications*; Cambridge University Press: New York, NY, USA, 2004.
31. Mohassel, P.; Rindal, P. ABY3: A Mixed Protocol Framework for Machine Learning. In Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, Toronto, ON, Canada, 15–19 October 2018; pp. 35–52.
32. Wagh, S.; Tople, S.; Benhamouda, F.; Kushilevitz, E.; Mittal, P.; Rabin, T. Falcon: Honest-Majority Maliciously Secure Framework for Private Deep Learning. *Proc. Priv. Enhancing Technol.* **2021**, *2021*, 188–208. [CrossRef]
33. Ito, M.; Saito, A.; Nishizeki, T. Secret Sharing Scheme Realizing General Access Structure. *Electron. Commun. Jpn.* **1989**, *72*, 56–64. [CrossRef]
34. Catrina, O.; Saxena, A. Secure Computation with Fixed-Point Numbers. In *Financial Cryptography and Data Security*; Hutchison, D., Kanade, T., Kittler, J., Kleinberg, J.M., Mattern, F., Mitchell, J.C., Naor, M., Nierstrasz, O., Pandu Rangan, C., Steffen, B., et al., Eds.; Springer: Berlin/Heidelberg, Germany, 2010; Volume 6052, pp. 35–50.
35. Beaver, D. Precomputing Oblivious Transfer. In Proceedings of the Advances in Cryptology—CRYPTO' 95, Santa Barbara, CA, USA, 27–31 August 1995; pp. 97–109.
36. Goldschmidt, R.E. Applications of Division by Convergence. Ph.D. Thesis, Massachusetts Institute of Technology, Singapore, 1964.
37. Damgård, I.; Pastro, V.; Smart, N.; Zakarias, S. Multiparty Computation from Somewhat Homomorphic Encryption. In Proceedings of the Advances in Cryptology—CRYPTO 2012, Santa Barbara, CA, USA, 19–23 August 2012; pp. 643–662.