Practical Privacy-Preserving K-means Clustering

Payman Mohassel* Mike Rosulek† Ni Trieu†

January 21, 2020

Abstract

Clustering is a common technique for data analysis, which aims to partition data into similar groups. When the data comes from different sources, it is highly desirable to maintain the privacy of each database. In this work, we study a popular clustering algorithm (K-means) and adapt it to the privacy-preserving context.

Specifically, to construct our privacy-preserving clustering algorithm, we first propose an efficient batched Euclidean squared distance computation protocol in the adaptive amortizing setting, when one needs to compute the distance from the same point to other points. This protocol can also serve as a key building block in many real-world applications such as Biometric Identification. Furthermore, we construct a customized garbled circuit for computing the minimum value among shared values.

We implement and evaluate our protocols to demonstrate their practicality and show that they are able to train datasets that are much larger and faster than in the previous work. The numerical results also show that the proposed protocol achieve almost the same accuracy compared to a K-means plain-text clustering algorithm.

1 Introduction

Advances in machine learning (ML) have enabled breakthroughs for solving numerous problems across various domains, for example, recommendation services, spam filtering, web search engines, fraud detection, stock market analysis and authentication technologies. Recently, cloud-based machine learning (ML) services provided by major technology companies such as Google, Microsoft, and AWS are getting popular. These services allow modular ML algorithms to be updated and improved via input from their customers. Training models for many such ML algorithms require large-scale data. In practice, the data can be collected from different sources, each of which might belong to a different entity. Internet companies regularly collect large amounts of information from users' online activities, search engines, and browsing behavior to train more accurate ML models. For example, credit card fraud-detection engines are becoming more accurate by training on large-scale data which combines transaction history, merchant data, and account holder information from financial companies and payment networks. Health data (e.g. genomic, patients) can be used to produce new diagnostic models. Since the data being classified or used for training is often sensitive and may come from different sources, it is imperative to design efficient methods to preserve privacy of data owners.

*payman.mohassel@gmail.com

†{rosulekm,trieun}@oregonstate.edu

While recent technologies enable more efficient storage and computation on big data, protecting combined data from different sources remains a big challenge. Recently, privacy-preserving machine learning via secure multiparty computation (MPC) has emerged as an active area of research that allows different entities to train various models on their joint data without revealing any information except the output. In this paper, we study privacy-preserving machine learning techniques for the clustering problem that aims to group similar data together according to some distance measure. Clustering is a popular unsupervised learning method and plays a key role in data management.

Indeed, our clustering setting has many practical applications due to the nature of MPC. For instance, two important and popular applications that motivate our clustering problem are: (1) Customer segmentation where companies can cooperate to cluster their customers into subgroups based on features of the customers. Then, they can run some prediction algorithms for each subgroup, or design dedicated marketing strategy effectively for each subgroup to maximize revenue; (2) Hospitals can cluster their patients into subgroups, then predict behavior of each subgroup, and design special medicine/treatment for each subgroup. In these examples, privacy is of utmost importance.

To this end, we design new and efficient privacy-preserving clustering protocols for an arbitrary partitioning of a dataset. Our major contributions can be summarized as follows:

- First, we introduce an efficient and secure squared Euclidean distance protocol in the adaptive amortized setting.
- Second, we build a customized garbled circuit to compute binary secret sharing of the minimum value among a list of secret shared values
- Furthermore, we present a scalable privacy-preserving clustering algorithm and design a modular approach for multi-party clustering.
- Finally, we implement and evaluate our clustering scheme to demonstrate its scalability. Our scheme is five orders of magnitude faster than the state-of-the-art work [40].

The rest of the paper is organized as follows. In Section 2, we discuss related work. In Section 3, we give preliminary information about notation, security model, cryptographic primitives, and K-means clustering. In Section 4, we describe the building blocks used in our privacy-preserving clustering scheme (which are also the key building blocks in many real-world application). In Section 5, we pull all the building blocks together and present our clustering scheme. Experimental results and the conclusion are shown in the last sections.

2 Related Work

Privacy-preserving data mining has been studied extensively for decades. To our knowledge, initial focus in this area was on secure construction of decision tree learning for distributed data sets proposed by Lindell and Pinkas [52]. This later expands to a wide range of other problems such as privacy-preserving mining of association rules [25, 67], clustering [76, 39, 41, 15, 38], linear regression [13, 69], logistic regression [35]. These papers propose solutions based on secure multiparty computation (MPC), but appear to incur high efficiency overheads and lack implementation.

In recent years, this research area has been very active due to recent advances in MPC. Gilad-Bachrach et. al. [29] propose a privacy preserving linear regression framework, which does not scale well due to extensive use of garbled circuits. Aono et. al. [17] consider a different security model for logistic regression, but in their setting, the plaintext of the aggregated data is leaked to the client who trains the model. Recent work of Mohassel and Zhang [57] customize the ABY framework [21], propose and implement an efficient framework for privacy-preserving linear regression, logistic regression, and neural networks in two-party setting. Many follow-up works and other variants of secure ML [56, 42, 45, 73, 19, 65, 18, 71, 9] have been proposed/investigated. In this section, we will focus on existing work on privacy-preserving clustering.

Clustering is a common technique for statistical data analysis used in many fields, which aims to map/partition each data point into a similar group. Earlier work on privacy-preserving clustering has been proposed by Vaidya and Clifton [76] Jagannathan and Wright [39], Jha, Kruger, and McDaniel [41] and Bunn and Ostrovsky [15], Jagannathan and Wright [38]. The work of Vaidya and Clifton [76] addresses privacy-preserving k-means clustering for vertically partitioned database (the database is distributed to different parties in a way that each party holds a subset of the attributes owned by the entity) while the work of McDaniel, and Jagannathan and Wright [41, 38] addresses horizontally partitioned database (each entity is owned by a single participant). The schemes of Jagannathan and Wright [39] and Bunn and Ostrovsky [15] work for arbitrary partitioned database. All of them except [15, 38] reveal intermediate candidate cluster centers, thereby breaching privacy. These protocols can be made more secure but require higher complexity. In [15], Bunn and Ostrovsky present a 2-party privacy-preserving k-means clustering protocol that guarantees full privacy in the semi-honest security model. The protocol hides the intermediate information by calculating the new cluster center using homomorphic encryption. Therefore, the scheme [15] is expensive due to extensive use of homomorphic encryption (HE). In [38], Jagannathan and Wright propose a simple communication-efficient clustering algorithm (called ReCluster) and describe its distributed privacy-preserving version. Their protocol works for databases that are horizontally partitioned between two parties. The high-level idea of their protocol is to have both parties separately use the ReCluster algorithm to compute $2K$ local clusters each from their own shares of the data. Next, parties use a secure merge operator to iteratively merge the $4K$ clusters into K clusters. This privacy-preserving ReCluster protocol does not leak intermediate candidate cluster centers, but reveals the merging pattern in which the adversary could potentially see which two local clusters will be merged in the next iteration.

Recently, the work of Jiawei Yuan and Yifan Tian [81] proposed a practical privacy-preserving K-means clustering scheme that can be efficiently outsourced to cloud servers. They investigated secure integration of MapReduce into their scheme, which makes their scheme extremely suitable for cloud computing environment. However, this work reveals the intermediate closet clustering centers to the server. Many recent works focus on clustering in the outsourcing setting (many parties and a trusted/untrusted mediator) [64, 53, 72, 43], or differential privacy setting [74, 83, 75, 8, 70]. There are few recent works [28, 61, 79, 40] that consider privacy preserving K-means clustering with full privacy guarantees. The solution of [28] only works for horizontally partitioned data. The distributed K-means clustering of [61] is based on Shamir's secret sharing scheme, thus their scheme requires more than two non-colluding servers. Moreover, it is not clear how to compute the distance metric in this work. The protocols [79, 40] are heavily based on homomorphic encryption and do not scale for large datasets (e.g. more than 10,000 data entries). For example, the state-of-the-art privacy preserving clustering scheme [40] requires almost 1.5 years to cluster a dataset of thousand points. Unfortunately, the paper [79] does not provide running time of their scheme, we only compare the performance of our protocol to that of [40] in Section 7.

Privacy-preserving hierarchical clustering is recently formally studied in [55]. It is well-known that the algorithm for hierarchical clustering has a complexity of $O(n^2 \log(n))$, where n is the number of data points. Today, the most commonly used clustering algorithm is K-means which is greedy and has a complexity of $O(n)$, although it has a disadvantage that we will discuss in Section

7.4.3. Thus, in this work, we focus on privacy-preserving solution for the K-means algorithm.

3 Preliminaries

We now introduce the notations and cryptographic primitives used throughout the paper.

3.1 Notation

In this work, the computational and statistical security parameters are denoted by κ, λ, respectively. We use $[\cdot]$ notation to refer to a set. For example, $[m]$ denotes the set $\{1, \ldots, m\}$. Vectors are denoted by bold letters such as $\mathbf{P}$. The i-th element of vector $\mathbf{P}$ is $\mathbf{P}[i]$. Define $[\![\mathbf{P}]\!]$ and $[\![\mathbf{P}]\!]_{\oplus}$ as the arithmetic and the binary secret sharing of a secret value $\mathbf{P}$, respectively. We denote secret sharing $\mathbf{P}^A$, $\mathbf{P}^A_{\oplus}$ and $\mathbf{P}^B$, $\mathbf{P}^B_{\oplus}$ where Alice holds $\mathbf{P}^A$, $\mathbf{P}^A_{\oplus}$ and Bob holds $\mathbf{P}^B$, $\mathbf{P}^B_{\oplus}$ such that $(\mathbf{P}^A + \mathbf{P}^B) = \mathbf{P} \mod 2^\ell$ or $\mathbf{P}^A_{\oplus} \oplus \mathbf{P}^B_{\oplus} = \mathbf{P}$. Here, the operations $+$ and $\oplus$ are addition and XOR on ℓ-bit variables, respectively.

3.2 Security Model and Computational Setting

We consider a set of parties who want to train clustering models on their joint data. At the end of the protocol, they learn the final cluster centers, and nothing else. Suppose there were a trusted third party to whom parties could send their data. The trusted third party would then train the clustering model and send the final results to parties. Unfortunately, there is no such trusted party in real-life scenarios. Secure multi-party computation enables distrustful parties to jointly evaluate any function on their private inputs without revealing anything except the result itself. There are two classical adversarial models. In the semi-honest (or honest-but-curious) model, the adversary is assumed to follow the protocol, but attempts to obtain extra information from the execution transcript. In the malicious model, the adversary may follow any arbitrary strategy. In this work, we consider the semi-honest model. Extensions to the adversarial model will be the subject of our future work.

We consider two computational settings:

1. Amortized setting where parties need to evaluate the same function many times on different inputs. For example, a crucial component of K-means clustering algorithm is Euclidean distance computation, which needs to be run repeatedly many times. Indeed, this setting has been formalized and utilized in many previous work such as garbled circuit [32, 78, 50].

2. Adaptive setting is similar to the amortized setting where the same function is evaluated many times on different inputs. However, the inputs of the current function evaluation depends on the output from the previous evaluation. We note that our adaptive setting is quite different from the one formalized in [16]. Our adaptive model is considered for performance while the model of [16] is a cryptographic game where adversarial behavior is based on the information gathered so far.

3.3 Secret Sharing

According to our privacy requirements, parties should only receive the result (e.g. cluster centers) at the end of the protocol, but all of the values computed in the intermediate steps of the algorithm

PARAMETERS: A bit length m, and two parties: sender $\mathcal{S}$ and receiver $\mathcal{R}$

FUNCTIONALITY:

- Wait for pair-input $(\boldsymbol{x}_0, \boldsymbol{x}_1) \in \{0,1\}^m$ from sender $\mathcal{S}$
- Wait for bit-input $b \in \{0,1\}$ the receiver $\mathcal{R}$
- Give output $\boldsymbol{x}_b$ to the receiver $\mathcal{R}$.

Figure 1: Oblivious Transfer functionality OT_m.

PARAMETERS: Number of clusters, number of data points n and n', dimension d, and two parties: sender and receiver

FUNCTIONALITY:

- Wait for data points $\{\mathbf{P}_1, \mathbf{P}_2, \ldots, \mathbf{P}_{n'}\} \subseteq \mathbb{F}_p^d$ from sender, and data points $\{\mathbf{P}_{n'+1}, \mathbf{P}_2, \ldots, \mathbf{P}_n\} \subseteq \mathbb{F}_p^d$ from receiver.
- Give $f(\mathbf{P}_1, \mathbf{P}_2, .., \mathbf{P}_n)$ to both parties, where f implements K-means algorithm [54].

Figure 2: The K-means Clustering Ideal Functionality.

should be unknown to either party. In our protocol, each computed intermediate value (e.g. a candidate cluster centroid) is shared as two uniformly distributed random values, where each party holds one of these two values such that their sum is the actual intermediate value. Throughout this paper, we use two different sharing schemes: Additive sharing, and Boolean sharing. We briefly review these sharing schemes but refer the reader to the paper [21] for more details.

To additively share $[\![x]\!]$ an ℓ-bit value x, the first party chooses $x^A \leftarrow \{0,1\}^\ell$ uniformly at random and sends $x^B = x - x^A \mod 2^\ell$ to the second party. In this paper, we mostly use the additive sharing, and denote it by $[\![.]\!]$ for short. For ease of composition we omit the modular operation in the protocol descriptions. To reconstruct an additively shared value $[\![x]\!]$, one party sends $[\![x]\!]$ to the party, who reconstructs the secret $x = x^A + x^B \mod 2^\ell$ locally. Arithmetic operations can now be directly applied to these shares. Given two shared arithmetic values $[\![x]\!]$ and $[\![y]\!]$, it is easy to non-interactively add the shares by having parties compute $[\![x+y]\!] = [\![x]\!] + [\![y]\!] \mod 2^\ell$.

Boolean sharing can be seen as additive sharing in the field $\mathbb{Z}_2$. The addition operation is replaced by the XOR operation and multiplication is replaced by the AND operation.

3.4 Oblivious Transfer

Oblivious Transfer (OT) is a cryptographic primitive for various efficient secure computation protocols. In OT, a sender with two input strings (x_0, x_1) interacts with a receiver who has an input choice bit b. An OT protocol allows the receiver to learn x_b without learning anything about x_{1-b}, while the sender learns nothing about b. The ideal OT functionality is described in Figure 1. We denote this OT functionality with an input string of length m by OT_m.

The first OT protocol was proposed by Rabin [63], which essentially relied on expensive public key operations. OT extension [10] refers to the idea that parties can evaluate only a small number of expensive OTs that are used as a base for performing many OTs using only cheap symmetric-key operations. Modern one-out-of-two OT extension protocols follow the structure of the IKNP protocol [37] (hereafter IKNP). Kolesnikov and Kumaresan [48] and Asharov *et al.* [7] proposed several IKNP optimizations and provided optimized implementation of the OT extension.

One useful variant of OT is Correlated OT (COT), in which the sender's OT inputs x_0, x_1 are chosen randomly subject to $x_0 \oplus x_1 = \Delta$, where Δ is chosen by the sender. In this OT variant, it is possible to let the protocol itself choose x_0 randomly. By doing so, the bandwidth requirement from sender to receiver is reduced by a half, thus the amortized communication cost for an COT is $\kappa + \ell$, where ℓ is bit-length of Δ. In our implementation, we require only this weaker OT variant.

3.5 Garbled Circuit

Garbled Circuit (GC) is currently the most common generic technique for practical two-party secure computation (2PC). GC was first introduced by Yao[80] and Goldreich *et al.* [31]. In the two-party setting, garbled circuit protocol consists of a garbler and evaluator: the garbler encodes a boolean function (e.g. min, division) into a garbled circuit using two random keys per each wire of the circuit; the evaluator first obtains corresponding keys of the input wires, and evaluates the circuit to learn the corresponding output wire key. The evaluator finally takes a decoding table, which maps the final output wire keys to the real values, and decodes the final output.

Garbled Circuit technique has seen dramatic improvements in recent years. The most notable optimized techniques are point-and-permute [12], Free-XOR [51], the half-gate [82], and fixed-key AES garbling optimizations [11]. Modern garbled circuit (GC) protocols evaluate two million AND gates per second on a 1Gbps LAN. Briefly, the ideal functionality GC is to take the parties inputs x and y respectively, and computes f on them. We denote this garbled circuit by $z \leftarrow \mathcal{GC}(x, y, f)$. In our protocols, we use "less than" and "division" garbled circuit.

3.6 Clustering Algorithm

Clustering is an important unsupervised learning technique for statistical data analysis, which have been extensively used in many fields. Given a set of data points, a clustering algorithm maps each data point into a similar group. The data points should have similar features or/and properties in the same group, but should have highly dissimilar features or/and properties between different groups. Cluster analysis is very important to gain valuable insights from the data points. There are several clustering algorithms that have their own pros and cons. Today, the most commonly used algorithm is K-means, which is greedy and computationally efficient. The K-means algorithm consists of two following steps:

(1) Initialize cluster centroids: This step can be implemented using different methods. A very common one is to pick random values for the centroids.

(2) Repeat until convergence (Lloyd's Iteration):

    (a) calculate the distance between each data point and all centroids, assign each data point to the cluster that has the closest centroid.

    (b) update the values of the centroids by computing the average of the values of the point attributes that are part of the cluster.

A Privacy-Preserving K-means clustering is an application of secure computation that allows parties, each holding a set of private data points, to cluster their their combined data sets without revealing anything except for the cluster centers. We describe the ideal functionality for K-means clustering algorithm in Figure 2.

4 Our Building Blocks

In this section, we present the enhancements we made to improve secure two-party multiplication and Euclidean distance in the adaptive amortizing setting, which are the core building blocks in many practical applications. We also introduce a customized garbled circuit to compute the minimum of shared values.

4.1 Secure Arithmetic Multiplication

Assume that Alice and Bob hold secret ℓ-bit numbers x and y respectively, and they want to obtain the arithmetic shared value of the product xy without revealing additional information beyond the output. Secure arithmetic multiplication has been well studied for decades, and can be generated based on either Homomorphic Encryption [27] or Oblivious Transfer [30]. Demmler *et al.* [21] benchmarked the generation of both OT-based and HE-based arithmetic multiplications, and show that with the advantage of recent advances in OT extension, the OT-based protocol is always faster than the HE-based one. Therefore, in this paper we focus on the OT-based protocol which works as follows: Alice and Bob invoke ℓ instances of OT where Alice acts as an OT receiver and Bob acts as an OT sender. In the i^{th} OT instance, Bob inputs a pair $(m_{i,0}, m_{i,1})$ where $m_{i,0} \leftarrow \mathbb{Z}_{2^\ell}$ and $m_{i,1} = (2^i y + m_{i,0}) \mod 2^\ell$; while Alice inputs $x[i]$ as choice bit, where $x[i]$ is the i^{th} bit of a binary expression $x = \sum_{i=0}^{\ell-1} 2^i x[i]$. The i^{th} OT enables Alice to obtain $m_{i,x[i]} = (2^i x[i] y + m_{i,0}) \mod 2^\ell$. Finally, Alice can compute the arithmetic shared value z^A by summing up $\sum_{i=0}^{\ell-1} m_{i,x[i]} \mod 2^\ell$. Similarly, Bob computes the arithmetic shared value z^B by summing up $(-\sum_{i=0}^{\ell-1} m_{i,0}) \mod 2^\ell$. It is easy to see that $z^B = xy - z^A$.

4.1.1 Revising Communication-Efficient Secure multiplication Based on 1-out-of-N OT

With recent improvement to 1-out-of-N OT, [22] proposed to replace 1-out-of-2 OT with 1-out-of-N OT for secure multiplication. In this section, we explicitly revise their 1-out-of-N OT based secure multiplication, and discuss an improved communication factor with different choices of N in Appendix A.1.

At a high-level idea[22], instead of using binary representation of her secret input x, Alice used an N-base representation, and rewrited $x = \sum_{i=0}^{\lceil \ell/\log(N) \rceil - 1} N^i x[i]$; next step is that Alice and Bob invoke $\lceil \ell/\log(N) \rceil$ instances of 1-out-of-N OT to obtain arithmetic shared value of each $N^i x[i] y$, where Alice has x and Bob has y. Concretely, in the i^{th} OT where $i \in \lceil \ell/\log(N) \rceil$, Bob acts as an OT sender with input sequence $(m_{i,0}, \ldots, m_{i,N-1})$ where $m_{i,0} \leftarrow \mathbb{Z}_{2^\ell}$ and $m_{i,j} = (N^i jy - m_{i,0}) \mod 2^\ell$; and Alice acts as an OT receiver with choice value $x[i] \in [N]$. As output from the 1-out-of-N OT, Alice obtains $m_{i,x[i]} = (N^i x[i] y - m_{i,0}) \mod 2^\ell$. Similar to the original OT-based secure multiplication, Alice computes the arithmetic shared value z^A by setting $\sum_{i=0}^{\lceil \ell/\log(N) \rceil - 1} m_{i,x[i]}$

mod 2^ℓ and Bob set $z^B = \sum_{i=0}^{\lceil \ell/\log(N) \rceil -1} m_{i,0} \mod 2^\ell$. The correctness of the protocol follows directly from the fact that $z^A + z^B = \sum_{i=0}^{\lceil \ell/\log(N) \rceil -1} (N^i x[i] y) \mod 2^\ell = xy$.

4.1.2 Secure multiplication in the adaptive amortized setting

We consider a case where Alice holds a ℓ-bit variable x and Bob *adaptively* has ℓ-bit variables $y_t, \forall t \in [T]$. They wish to compute a secure multiplication many times, where both parties obtain the arithmetic shared value of the product $xy_t, \forall t \in [T]$. Instead of repeating the above protocol T times, we propose a simple but efficient solution to compute the multiplication in the adaptive amortized setting. By selecting Alice as the OT receiver, we observe that her choice bits $x[i]$ are fixed, where $x[i]$ comes from the expression $x = \sum_{i=1}^{\lceil \ell/\log(N) \rceil} N^i x[i]$. Thus, we can reuse OT instances (i.e. reduce $T\times$ number of OT instances used to compute T multiplications) in this setting.

We present a simple batched OT protocol, inspired from [37]. Assuming that Bob holds T adaptive sequences $(m_{t,1}, \ldots, m_{t,N}), \forall t \in [T]$, while Alice has a choice value $c \in [N]$. Alice wishes to receive $m_{t,c}, \forall t \in [T]$, and nothing else. A simple solution is as follows: Alice, who acts as OT receiver with input choice c, interacts with the OT sender Bob to perform a 1-out-of-N OT on random strings. As output from the OT, Alice obtains k_c while Bob receives $(k_1, k_2, \ldots, k_N)$. Whenever a new t^{th} sequence is known by Bob, he uses these $(k_1, \ldots, k_N)$ as the encryption keys to encrypt this sequence $(m_{t,0}, \ldots, m_{t,N})$ respectively (i.e. $e_{t,i} = Enc(k_i, m_{t,i}), \forall t \in [T]$ and sends the encrypted results to Alice, who later decrypts the ciphertext $e_{t,c}$ using the decrypted key k_c and outputs $m_{t,c}$.

The combination of our observation on fixed OT choices and batched OT protocol reduces the bandwidth requirement by approximately half. For simplicity, assume that $N = 2$, performing T multiplications requires ℓT number of 1-out-of-2 OT instances, which requires $\ell T(\kappa + \ell)$ sent bits. With our batched OT technique, the bandwidth requirement is $\ell(\kappa + \ell T)$, an $\frac{T(\kappa+\ell)}{(\kappa+\ell T)}\times$ improvement. For example, for doing $T = 30$ iterations, this solution shows a factor of 2.16× and 1.59× improvement with $\ell = 32$ bits and $\ell = 64$ bits, respectively.

4.2 Secure Euclidean Squared Distance (SESD)

Euclidean distance is the ”ordinary” straight-line distance between two points, which involves computing the square root of the sum of the squares of the differences between two points in each dimension. In many algorithms (e.g. clustering, texture image retrieval, face-recognition, fingerprint-matching), we only need to compute and compare the distances among the points. Therefore, to improve the computation efficiency, the Euclidean distance can be replaced by the Euclidean squared distance (ESD)[1], which does not affect the output of the algorithms. We denote the ESD between two points x and y by $z \leftarrow \mathcal{F}_{\mathsf{EDist}}(x, y)$.

Consider two points $\mathbf{P}$ and $\boldsymbol{\phi}$, each has d dimensions. Assume that both parties have arithmetic secret shared value $[\![\mathbf{P}]\!]$ and $[\![\boldsymbol{\phi}]\!]$. They want to compute the secure Euclidean squared distance by which both parties obtain the arithmetic shared value of the output $\mathcal{F}_{\mathsf{EDist}}(\mathbf{P}, \boldsymbol{\phi})$. The Euclidean squared distance between points $\mathbf{P}$ and $\boldsymbol{\phi}$ is given as follows:

[1] ESD is not a metric, as it does not satisfy the triangle inequality.

PARAMETERS: T iterations, and two parties: the sender $\mathcal{S}$ and the receiver $\mathcal{R}$

FUNCTIONALITY:

- Wait for arithmetic secret sharings $[\![\mathbf{P}_1]\!], \ldots, [\![\mathbf{P}_n]\!]$ of n points $\mathbf{P}_i, i \in [n]$, from both parties.
- For each iteration $t \in T$:
  - Wait for arithmetic secret sharings $[\![\boldsymbol{\phi}_{t1}]\!], \ldots, [\![\boldsymbol{\phi}_{tK}]\!]$ of K points $\boldsymbol{\phi}_{tk}, k \in [K]$, from both parties.
  - For each $k \in [K]$, give arithmetic secret sharings of the output $\mathcal{F}_{\mathsf{EDist}}(\mathbf{P}_i, \boldsymbol{\phi}_{tk})$ to both parties, where $\mathcal{F}_{\mathsf{EDist}}(x, y)$ denotes Euclidean Squared Distance between two points x and y.

Figure 3: Secure Euclidean Squared Distance (SESD) functionality in the Adaptive Amortized Setting.

PARAMETERS: Two parties: sender $\mathcal{S}$ and receiver $\mathcal{R}$

FUNCTIONALITY:

- Wait for arithmetic secret sharings $[\![X_1]\!], \ldots, [\![X_K]\!]$ of K numbers from both parties.
- Give *binary* secret sharings $[\![\mathbf{C}]\!]_{\oplus}$ of the vector $\mathbf{C} = (0, \ldots, 1, \ldots, 0)$ to both parties, where the '1' appears in the k^{th} coordinate to indicate that the smallest number is X_k.

Figure 4: Secure Minimum of k Numbers, $\mathcal{F}^{K}_{\mathsf{min}}$

$$
\begin{aligned}
\mathcal{F}_{\mathsf{EDist}}([\![\mathbf{P}]\!], [\![\boldsymbol{\phi}]\!]) &= \mathcal{F}_{\mathsf{EDist}}(\mathbf{P}^A, \mathbf{P}^B, \boldsymbol{\phi}^A, \boldsymbol{\phi}^B) \\
&= \sum_{\rho=1}^{d} (\mathbf{P}^A[\rho] + \mathbf{P}^B[\rho] - \boldsymbol{\phi}^A[\rho] - \boldsymbol{\phi}^B[\rho])^2 \\
&= \sum_{\rho=1}^{d} (\mathbf{P}^A[\rho] - \boldsymbol{\phi}^A[\rho])^2 + \sum_{\rho=1}^{d} (\mathbf{P}^B[\rho] - \boldsymbol{\phi}^B[\rho])^2 \\
&\quad + 2\sum_{\rho=1}^{d} (\mathbf{P}^A[\rho] - \boldsymbol{\phi}^A[\rho])(\mathbf{P}^B[\rho] - \boldsymbol{\phi}^B[\rho]))
\end{aligned}
\tag{1}
$$

Observe that the terms $(\mathbf{P}^A[\rho] - \boldsymbol{\phi}^A[\rho])^2$ and $(\mathbf{P}^B[\rho] - \boldsymbol{\phi}^B[\rho])^2$ can be computed locally by Alice and Bob, respectively. Since the mixed term $(\mathbf{P}^A[\rho] - \boldsymbol{\phi}^A[\rho])(\mathbf{P}^B[\rho] - \boldsymbol{\phi}^B[\rho])$ leaks information if known in the clear by a party, it requires to compute this mixed term securely. Clearly, this mixed term can be computed by a secure multiplication on input $\mathbf{P}^A[\rho] - \boldsymbol{\phi}^A[\rho]$ held by Alice and input $\mathbf{P}^B[\rho] - \boldsymbol{\phi}^B[\rho]$ held by Bob.

In data mining applications (e.g. K-nearest Neighbor [44, 18]), parties need to jointly compute the Euclidean distance between each fixed point $\mathbf{P}_i$ and many points $\boldsymbol{\phi}_k$ which are (either adaptively or non-adaptively) known by parties. For example, Step (2a) of the K-means clustering algorithm (ref. 3.6) is to compute the distance between each data point and all centroids which are updated in Step (2b). Therefore, the centroids are non-adaptively known by parties in the same iteration but adaptively known between the iterations. We define this problem (SESD) as follows: Given secret shared value of n points $\mathbf{P}_i, i \in [n]$, each has d dimensions, assume that parties must do T iterations, in the t^{th} iteration they compute secure Euclidean squared distance between each point $\mathbf{P}_i$ and all K points $\boldsymbol{\phi}_{tk}, k \in [K]$. We describe the ideal functionality for SESD in Figure 3.

A direct solution [21, 15, 38, 40, 44] uses a secure multiplication to compute the mixed term $(\mathbf{P}_i^A[\rho] - \boldsymbol{\phi}_{tk}^A[\rho])(\mathbf{P}_i^B[\rho] - \boldsymbol{\phi}_{tk}^B[\rho]), \rho \in [d]$, for each Euclidean squared distance $\mathcal{F}_{\mathsf{EDist}}(\mathbf{P}_i, \boldsymbol{\phi}_{tk}), i \in [n], k \in [K]$. Let τ be a number of OT instances used to perform a secure multiplication. This solution requires $\tau dnKT$ instances of OTs to securely compute the SESD functionality described in Figure 3.

We observe that the points $\mathbf{P}_i$ are fixed during all T iterations. We propose an optimized solution to compute the mixed term in the amortized setting. We rewrite the mixed term as follow:

$$\begin{aligned} &(\mathbf{P}_i^A[\rho] - \boldsymbol{\phi}_{tk}^A[\rho])(\mathbf{P}_i^B[\rho] - \boldsymbol{\phi}_{tk}^B[\rho]) \\ &= \mathbf{P}_i^A[\rho]\big(\mathbf{P}_i^B[\rho] - \boldsymbol{\phi}_{tk}^B[\rho]\big) - \mathbf{P}_i^B[\rho]\boldsymbol{\phi}_{tk}^A[\rho] + \boldsymbol{\phi}_{tk}^A[\rho]\boldsymbol{\phi}_{tk}^B[\rho] \end{aligned} \tag{2}$$

The first and second terms can be computed using the batched secure multiplication in the adaptive amortized setting (as described in Section 4.1.2), where $\mathbf{P}_i^A[\rho]$ and $\mathbf{P}_i^B[\rho]$ are fixed. We also observe that in each t^{th} iteration, parties perform K secure multiplications $\mathbf{P}_i^A[\rho]\big(\mathbf{P}_i^B[\rho] + \boldsymbol{\phi}_{tk}^B[\rho]\big)$, $\forall k \in [K]$ with the same value $\mathbf{P}_i^A$. Similar to technique of [57], Bob who acts as OT sender concatenates the OT strings (e.g. $m_{1,0}||\ldots||m_{K,0}$) before encrypting and sending them to Alice. The same trick is applied to compute the second term $\mathbf{P}_i^B[\rho]\boldsymbol{\phi}_{tk}^A[\rho]$. In conclusion, computing the first and second terms of Eq. (2) requires only $2\tau dn$ instances of OTs for all T iterations. We use a secure multiplication to compute the third term $\boldsymbol{\phi}_{tk}^A[\rho]\boldsymbol{\phi}_{tk}^B[\rho]$ of Eq. (2), which takes $O(\tau dKT)$ OT invocations for all T iterations. In Appendix A.2, we describe our SESD construction in Figure 8 and its security proof.

Very recently, [18] proposed an efficient SESD protocol based on additive homomorphic encryption, which is used for k-Nearest neighbor search problem. However, it is not quite clear how to extend their protocol to compute many SESD in our *adaptive* amortized setting. Moreover, it is not clear how to update the the centroid without using our "Minimum of k numbers" subprotocol (see the next section), which outputs the secret share of the centroid index, not the value.

Cost. Our solution for the SESD functionality (Figure 3) requires $(2n+KT)\tau d$ number of OT instances, which is $\frac{nKT}{2n+KT}\times$ improvement compared to the previous works. For example, evaluating K-means algorithm on 2D synthetic dataset S1 [26] which contains $n = 5,000$ tuples and $K = 15$ Gaussian clusters, our solution shows a factor of 215× improvement for doing $T = 30$ iterations.

4.3 Minimum of k Numbers

Recall that a fundamental building block of many algorithms (e.g. K-means clustering [15], face-recognition [68], fingerprint-matching [14, 36], K-nearest Neighbor [44, 18]) is to compute the Euclidean squared distance between two points in the database and then determine the minimum value among these distances. Concretely, Step (2a) of the K-means clustering algorithm (ref. Section 3.6) needs to find a closest centroid to each data point. It is needed to hide the closest centroid. Unlike other secure ML problems (e.g. K-nearest Neighbor) that can output the secret share of the centroid/center, secure K-means clustering requires to output the secret share of the cluster's index indicating the closest one. We consider the problem that takes the arithmetic secret sharings

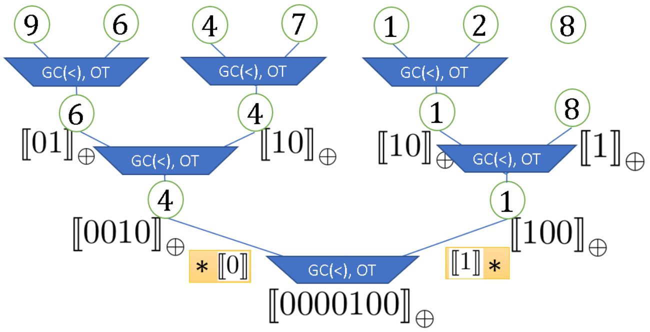

9
6
4
7
1
2
8
GC(<), OT
GC(<), OT
GC(<), OT
6
4
1
8
$\llbracket 01 \rrbracket_\oplus$
GC(<), OT
$\llbracket 10 \rrbracket_\oplus$
$\llbracket 10 \rrbracket_\oplus$
GC(<), OT
$\llbracket 1 \rrbracket_\oplus$
4
1
$\llbracket 0010 \rrbracket_\oplus$
$\llbracket 100 \rrbracket_\oplus$
$* \llbracket 0 \rrbracket$
GC(<), OT
$\llbracket 1 \rrbracket *$
$\llbracket 0000100 \rrbracket_\oplus$

Figure 5: Illustration of the main idea behind our $\mathcal{F}^K_{\mathsf{min}}$ protocol.

$\llbracket X_1 \rrbracket, \ldots, \llbracket X_K \rrbracket$ of K numbers, and returns *binary* secret sharings of the vector $\mathbf{C} = (0, \ldots, 1, \ldots, 0)$ (called index vector), where the '1' appears in the k^{th} coordinate to indicate that the smallest number is X_k. We denote this problem by $\llbracket \mathbf{C} \rrbracket_\oplus \leftarrow \mathcal{F}^K_{\mathsf{min}}(\llbracket X_1 \rrbracket, \ldots, \llbracket X_K \rrbracket)$.

In most previous work [15, 40], $\mathcal{F}^K_{\mathsf{min}}$ is implemented using generic secure computation (e.g. FHE, GC). Using FHE is still computationally expensive while the GC-based $\mathcal{F}^K_{\mathsf{min}}$ requires $K-1$ "less than" and $K-1$ "multiplexer" circuits to find the minimum value among K input numbers, and K "equality" circuits to determine the k^{th} coordinate indicating the smallest numbers. We build a customized garbled circuit to implement $\mathcal{F}^K_{\mathsf{min}}$, which requires only $K-1$ "less than" garbled circuits and $4(K-1)$ instances of OT extension. Note that the cost of "multiplexer" garbled circuit is $O(\kappa \cdot \ell)$ due to the need of garbling ℓ-bit strings [77], while the cost of OT instances is $O(\kappa + \ell)$.

Figure 5 illustrates the main idea behind our $\mathcal{F}^K_{\mathsf{min}}$ protocol. Our protocol can be described in a recursive way as follows. Assume we have secret shared index vector $\llbracket \mathbf{C}_0 \rrbracket_\oplus$ as the output of $\mathcal{F}^K_{\mathsf{min}}(\llbracket X_1 \rrbracket, \ldots, \llbracket X_{[K/2]} \rrbracket)$, we also store the shared value of minimum value X_k of $X_1, \ldots, X_{[K/2]}$. Similarly, we have $\llbracket \mathbf{C}_1 \rrbracket_\oplus, \llbracket X_{k'} \rrbracket \leftarrow \mathcal{F}^K_{\mathsf{min}}(\llbracket X_{[K/2]+1} \rrbracket, \ldots, \llbracket X_K \rrbracket)$, where $X_{k'}$ is minimum value among $X_{[K/2]+1}, \ldots, X_K$. We observe that index vector $\mathbf{C}$ is equal to the concatenation of $b\mathbf{C}_0$ and $\bar{b}\mathbf{C}_1$, where $b = 1$ indicates that the minimum value is X_k, and vice versa. Thus, the parties first evaluate a "less than" garbled circuit on the inputs X_k and $X_{k'}$. We modify the "less than" garbled circuit to output 2-bit binary shares $(\llbracket b \rrbracket_\oplus \llbracket \bar{b} \rrbracket_\oplus)$. The next step is to efficiently compute the binary secret sharing of $b\mathbf{C}_0$.

We rewrite $b\mathbf{C}_0 = (b^A \oplus b^B)(\mathbf{C}^A_0 \oplus \mathbf{C}^B_0)$, and invoke 2 OT instances to output its binary shared values. Concretely, Alice acts as OT sender with a pair input $(m_0 \oplus b^A\mathbf{C}^A_0, m_0 \oplus (b^A \oplus 1)\mathbf{C}^A_0)$ where m_0 is chosen randomly, while Bob acts as OT receiver with a choice bit b^B. As output from OT, Bob obtains $m_{b^B} = m_0 \oplus (b^A \oplus b^B)\mathbf{C}^A_0$. Similarly, Alice acts OT receiver with a choice bit b^A and obtains $m'_{b^A} = m'_0 \oplus (b^A \oplus b^B)\mathbf{C}^B_0$ while Bob acts as OT sender and knows m'_0. Alice sets $z^A = m_0 \oplus m'_{b^A}$, Bob sets $z^B = m'_0 \oplus m_{b^B}$. It is easy to see that z^A and z^B are binary secret sharing of $(b^A \oplus b^B)(\mathbf{C}^A_0 \oplus \mathbf{C}^B_0)$.

Recall that we need to store the minimum value of X_k and $X_{k'}$ for further computation. This minimum value is equal to $bX_k + \bar{b}X_{k'}$. To compute shared value of $bX_k = (b^A \oplus b^B)(X^A_k + X^B_k)$, we again need 2 OT instances, each has a choice bit b^B or b^A. However, since the same OT choice bits are used in this minimum computation and in computing the index vector $\mathbf{C}$ above, thus parties can reuse the OT by concatenating the OT sender's messages. As a result, determining minimum is almost free in terms of computational cost.

Compared to generic GC, this solution adds $\lceil \log(K) \rceil$ rounds, but K is usually small (e.g. $K = 3$ or $K = 15$). Bunn and Ostrovsky [15] proposed a protocol to find a bit output indicating smallest of two numbers by running the secure scalar products many times. With various optimizations to GC over the years, a GC-based minimum protocol is faster than that of [15]. Our protocol is similar to that of Jäschke and Armknecht [40]. However, the protocol [40] requires $K-1$ "multiplexer" circuits to obtain the minimum value of two numbers, which is mostly free in our protocol. Moreover, [40] uses FHE to compute the shares of index vector $\mathbf{C}$ while our protocol costs only four OT instances.

5 K-Mean Clustering Framework

In this section, we present our secure K-means clustering protocol and show how to put all building blocks (described in Section 4) together. Recall that the K-means clustering algorithm consists of two steps: Cluster centroids initialization, and Lloyd's iteration.

5.1 Cluster Initialization

This step can be done using different strategies. A very common one is to pick random values for the of all K groups. This approach can be easily implemented in the privacy-preserving setting by letting one party choose random centroid values, and secret share these values to other party. Another method is to use the values of K different data points as being the centroids, which is also simply implemented in this setting. We now propose another approach specified for privacy-preserving K-means clustering as follows. Each party locally runs the plain-text K-means clustering algorithm to group his/her data point into $[k/2]$ groups. Parties secret share local centroid of each group to each other.

5.2 Lloyd's iteration

Lloyd's iteration can be divided into four steps:

(1) Calculate the distance between each data point and cluster centers using the Euclidean squared distance

(2) Assign each data point to the closest cluster center

(3) Recalculate the new cluster center by taking the average of the points assigned to that cluster.

(4) Repeat steps 1, 2 and 3 iteratively either a given number of times, or until clusters can no longer change.

We notice that the data points are fixed during the training while the cluster centers can be changed between two iterations. Thus, our SESD protocol (ref. Section 4.2) can be directly applied to Step (1) of Lloyd's iteration.

5.2.1 Approximation of Euclidean distance

In Machine Learning, Euclidean distance (norm-2) is the most common distance measure used in K-means clustering. However, its main drawback is the high computational cost due to the multiplication operator. Thus, Manhattan metric (norm-1) and Chessboard metric (norm-∞) are often considered as alternatives. The Manhattan distance between two points x and y is the sum of the absolute differences of their coordinates (e.g. $\sum_{i=1}^{d} |x_i - y_i|$). The Chessboard distance between two points is the greatest of their absolute differences along any coordinate dimension (i.e. $\max\limits_{i \in [d]} |x_i - y_i|$). We denote the Manhattan and Chessboard distance between x and y by $z \leftarrow \mathcal{F}_{\mathsf{MDist}}(x, y)$, $z \leftarrow \mathcal{F}_{\mathsf{CDist}}(x, y)$, respectively.

We implement secure Manhattan and Chessboard distance, and report their runtime in Section 7.3. We calculate the absolute differences of two values, and find the greatest of these differences using a garbled circuit.

5.2.2 Assigning data point to clusters

From Step (1), parties have arithmetic secret shared value $[\![X_{ik}]\!]$ of the distance from each point $\mathbf{P}_i, i \in [n]$, to the cluster center $\boldsymbol{\phi}_k, k \in [K]$. For each data point $\mathbf{P}_i$, we find its nearest cluster by invoking our $\mathcal{F}_{\mathsf{min}}^{K}$ protocol (as described in Section 4.3). The index vector output $[\![\mathbf{C}_i]\!]_{\oplus} \leftarrow \mathcal{F}_{\mathsf{min}}^{K}([\![X_{i1}]\!], \ldots, [\![X_{iK}]\!])$ indicates which cluster center this data point is assigned to.

5.2.3 Updating Cluster Centers

We form a matrix $\mathcal{C}$ of size $n \times K$, where each row is index vector $\mathbf{C}_i$ obtained from Step (2). Let $\mathbf{M}_k, k \in [K]$, be the row of the matrix transposition of $\mathcal{C}$ (see Figure 9 in Appendix C) . It is easy to see that the i^{th} element of $\mathbf{M}_k$ is set to be 1 if and only if the data point $\mathbf{P}_i$ is assigned to the cluster k. Therefore, we can calculate the new centroid by taking the mean:

$$\boldsymbol{\varphi}_k = \frac{\sum_{i=1}^{n} \mathbf{M}_k[i]\mathbf{P}_i}{\sum_{i=1}^{n} \mathbf{M}_k[i]} = \frac{\sum_{i=1}^{n} (\mathbf{M}_k^A[i] \oplus \mathbf{M}_k^B[i])(\mathbf{P}_i^A + \mathbf{P}_i^B)}{\sum_{i=1}^{n} (\mathbf{M}_k^A[i] \oplus \mathbf{M}_k^B[i])} \tag{3}$$

To compute the secret sharing of the updated cluster $\boldsymbol{\varphi}_k$, parties first compute the numerator and denominator and then calculate the reminder using a division garbled circuit. Similar trick used in determining minimum of two shared numbers in Section 4.3, the numerator can be implemented using $4n$ OT invocations. Since the same bits $\mathbf{M}_k^A[i]$ and $\mathbf{M}_k^B[i]$ are used in both numerator and denominator computation, we can reuse the OT instances to computing the denominator. Therefore, updating the centroid $\boldsymbol{\varphi}_k$ requires $4n$ OT instances and one division garbled circuit.

5.2.4 Checking the stopping criterion

After obtaining the secret sharing $[\![e_k]\!]$ of the Euclidean squared distance between the new cluster centroid $\boldsymbol{\varphi}_k$ and $\boldsymbol{\phi}_k$, parties locally sum up these shares and invoke a 'min' garbled circuit $\mathcal{GC}(\sum_{k=1}^{K} [\![e_k]\!], \epsilon, min) = 1$ to check the stopping criterion.

5.3 Main Construction

We describe the main construction of K-means clustering protocol in Figure 6. It closely follows and formalizes these above steps presented in sections 5.1 and 5.2. We note that the input and output of each Lloyd's steps are secret shares of corresponding variables. Our theorem statement and its security proof is given in Appendix B.

Parameters:

- Number of clusters K; number of data points n', n; dimension d
- Ideal $\mathcal{F}_{\mathsf{EDist}}, \mathcal{F}^{K}_{\mathsf{min}}$ primitives defined in Figure 3 and Figure 4, respectively.
- Garbled circuit $z \leftarrow \mathcal{GC}(x, y, f)$ described in Section 3.5, which takes x and y as inputs, and computes $z = f(x, y)$.

Input of Alice: $\{\mathbf{P}_1, \mathbf{P}_2, \dots, \mathbf{P}_{n'}\} \subseteq \mathbb{F}_p^d$
Input of Bob: $\{\mathbf{P}_{n'+1}, \mathbf{P}_2, \dots, \mathbf{P}_n\} \subseteq \mathbb{F}_p^d$

Protocol:

I. **Initialization:**

(a) Alice locally runs the plain-text K-means clustering algorithm to group her dataset into $\lceil K/2 \rceil$ cluster centers $\{\boldsymbol{\phi}_1, \dots, \boldsymbol{\phi}_{\lceil K/2 \rceil}\}$. She secret shares $\{\llbracket \boldsymbol{\phi}_1 \rrbracket, \dots, \llbracket \boldsymbol{\phi}_{\lceil K/2 \rceil} \rrbracket\}$ and $\{\llbracket \mathbf{P}_1 \rrbracket, \dots, \llbracket \mathbf{P}_{n'} \rrbracket\}$ to Bob.

(b) Bob locally runs the plain-text K-means clustering algorithm to group his dataset into $K - \lceil K/2 \rceil$ cluster centers $\{\boldsymbol{\phi}_{\lceil K/2 \rceil+1}, \dots, \boldsymbol{\phi}_K\}$. He secret shares $\{\llbracket \boldsymbol{\phi}_{\lceil K/2 \rceil+1} \rrbracket, \dots, \llbracket \boldsymbol{\phi}_K \rrbracket\}$ and $\{\llbracket \mathbf{P}_{n'+1} \rrbracket, \dots, \llbracket \mathbf{P}_n \rrbracket\}$ to Alice.

II. **Lloyd's Step:** Repeat the following until the stopping criterion.

(a) For $i \in [n], k \in [K]$, Alice and Bob jointly compute secret sharing of the Euclidean squared distance:

$$\llbracket X_{ik} \rrbracket = \mathcal{F}_{\mathsf{EDist}}(\llbracket \mathbf{P}_i \rrbracket, \llbracket \boldsymbol{\phi}_k \rrbracket)$$

(b) For $i \in [n]$, Alice and Bob jointly compute secret sharing of the nearest cluster:

$$\llbracket C_i \rrbracket_{\oplus} = \mathcal{F}^{K}_{\mathsf{min}}(\llbracket X_{i,1} \rrbracket, \dots, \llbracket X_{i,K} \rrbracket)$$

Parties forms $K \times n$ matrix $\mathcal{C}$ such that the i^{th} column of $\mathcal{C}$ is the vector $\llbracket C_i \rrbracket_{\oplus}$. Let $\llbracket M_k \rrbracket_{\oplus}$ denote the k^{th} row of $\mathcal{C}$.

(c) For $k \in [K]$, Alice and Bob jointly calculate secret sharing $\llbracket \boldsymbol{\varphi}_k \rrbracket$ of the new cluster centers as follows:

- Computing secret sharing of the numerator $\llbracket \mathbf{M} \rrbracket = \sum_{i=1}^{n} (\mathbf{M}_k^A[i] \oplus \mathbf{M}_k^B[i])(\mathbf{P}_i^A + \mathbf{P}_i^B)$
- Computing secret sharing of the denominator $\llbracket D \rrbracket = \sum_{i=1}^{n} (\mathbf{M}_k^A[i] \oplus \mathbf{M}_k^B[i])$
- Invoking a division garbled circuit $\llbracket \boldsymbol{\varphi}_k \rrbracket \leftarrow \mathcal{GC}(\llbracket \mathbf{M} \rrbracket, \llbracket D \rrbracket, \frac{\mathbf{M}}{D})$.

and then, parties also compute secret sharing of Euclidean squared distance:

$$\llbracket e_k \rrbracket = \mathcal{F}_{\mathsf{EDist}}(\llbracket \boldsymbol{\varphi}_k \rrbracket, \llbracket \boldsymbol{\phi}_k \rrbracket)$$

(d) Alice and Bob jointly check the stopping criterion: If $\mathcal{GC}(\llbracket e \rrbracket, \epsilon, \min(e, \epsilon)) = 1$, where $\llbracket e \rrbracket = \sum_{k=1}^{K} \llbracket e_k \rrbracket$, then stop the criterion, parties reveal $\boldsymbol{\varphi}_k$. Otherwise, replacing $\llbracket \boldsymbol{\phi}_k \rrbracket = \llbracket \boldsymbol{\varphi}_k \rrbracket$

Figure 6: Our Privacy-preserving K-Means Clustering Framework.

6 Multi-party Clustering

In this section, we extend our two-party clustering scheme to support a set of users $U_0, \ldots, U_m$ who want to train a clustering model on their joint data. We consider two following models:

1. Server-aided model: Given a set of users with private datasets, server-aided model allows the clients to outsource the computation to two untrusted but non-colluding servers.

2. Multi-party computation: users jointly train the model on their joint data without requiring a trusted/untrusted additional party.

6.1 Server-aided model

The server-aided setting has been formalized, utilized in various previous work [46], and in privacy-preserving machine learning model [58, 59]. Given a semi-honest adversary $\mathcal{A}$ who can corrupt any subset of the users and at most one of the two untrusted servers, the security definition of this model requires that such an adversary only learns the data of the corrupted users and the final model, but nothing else about the remaining honest non-corrupted users' data. It is easy to see that our K-means clustering scheme (described in Section 5) can be directly applied to this model where users can secret share their inputs among the two untrusted servers. This distribution step can be done in a setup phase. Therefore, the advantage of this model is that it does not require the users to be involved throughout the protocol computation.

6.2 Multi-party computation model

The data stream model has attracted attention in machine learning and data analysis, and is used to analyze very large datasets. Popular clustering data stream algorithms are CURE [34], BIRCH [84], and STREAM [33] which achieves a constant factor approximation algorithm for the k-Median problem. A clustering data stream is a divide-and-conquer algorithm that divides the whole data into small pieces, and clusters each one of them using K-means, then clusters the resulting centers. Inspired by this technique, we propose a secure clustering scheme in multi-party setting. This model provides a weaker security guarantee where we assume that we know user U_0 who does not collude with other users.

A solution is to perform a secure two-party computation where each user plays the role of one party in our privacy-preserving clustering scheme (ref. Section 5). Concretely, two users U_0 and $U_i, i \neq 0$, perform 2-party secure K-means clustering. As a result, users receive the shared value of the cluster centroids (denote them as $\boldsymbol{\phi}_k^{U_0}$ and $\boldsymbol{\phi}_k^{U_i}$). Next step is that user U_i sends these obtained shared values $\boldsymbol{\phi}_k^{U_i}$ to user U_{i+1} in the clear (this captures the property that users $U_i, i \neq 0$, are not colluding). Users U_0 and U_{i+1} now can use the values $\boldsymbol{\phi}_k^{U_0}$ and $\boldsymbol{\phi}_k^{U_i}$ as the initial centroids for training model on their data.

7 Experimental Results

We implement a privacy-preserving clustering system based on our proposed protocols and report the experimental results in this section. We also compare the performance of our scheme with the state-of-the-art privacy-preserving clustering protocols in [74] and [40].

Dataset	# of points n	# of clusters K	dimension d
Lsun [40]	400	3	2
arff [2]	1000	2	4
S1 [75]	5000	15	2
DIM-sets [47]	10000	9	$\{2, 4, 6, 8, 10\}$
self-generated data	$\{10000, 100000\}$	$\{2, 5\}$	2

Table 1: Descriptions of the Datasets.

7.1 Experimental Setup

To understand the scalability of our protocol, we evaluate it on a single server which has 2x 36-core Intel Xeon 2.30GHz CPU and 256GB of RAM. Although there are many cores, each party does their computation only on a single thread. We run all parties in the same network, but simulate a network connection using the Linux tc command: a LAN setting with 0.02ms round-trip latency, 10 Gbps network bandwidth. We observe that running times on WAN can be computed with the linear cost model as the overall running time is equal to the sum of computation time and data transfer time. Moreover, the previous work has conducted expermentials numbers in LAN setting only. Thus, we will focus on the LAN setting in all the experiments below.

For the most direct comparison to the work of Jäschke and Armknecht [40], we matched the test system's computational performance to that of [40]. We evaluate our protocol on a machine Intel Core i7 2.60GHz with 12GB RAM.

In our protocol, the base-OT is implemented using Naor-Pinkas construction. The system is implemented in C++, and builds on use the primitives provided by Ivory Runtime library [4] for garbled circuits (free XOR [51],half-gate [82], fixed-key AES garbling optimizations [11]), and libOTe [66] for OT extension of [37]. All evaluations were performed with statistical security parameter $\lambda = 40$ and computational security parameter $\kappa = 128$.

7.2 Datasets

For fair comparison, we use two datasets, each of which was evaluated in some relevant previous works:

- The first dataset is Lsun dataset [5], which consists of 400 data points of 2 dimensions and 3 clusters. This dataset was evaluated in [40]
- The second dataset is a 2D synthetic dataset S1 [26], which was experimented in [74] in the Differentially Privacy setting. The S1 dataset contains 5,000 data points and 15 Gaussian clusters.

We also consider arff [2] dataset for a visual accuracy comparison. The dataset consists of 1000 data points of 4 clusters and 2 dimensions. Furthermore, to verify the performance of the proposed protocol for datasets with different dimensions, we use DIM-sets [47] that contain 10000 vectors in 9 clusters in $d \in \{2, 4, 6, 8, 10\}$-dimensional space. Finally, we generate synthetic 2D datasets with sizes of $\{10000, 100000\}$ and $K \in \{2, 5\}$. Table 1 summarizes the datasets used in our experiments.

7.3 Experiments for Distance Metric

We start with the experimental results for the secure Euclidean squared distance protocol (its functionality described in Figure 3), and compare it with previous privacy preserving solutions [21, 44].

7.3.1 Secure Euclidean Squared Distance

To examine how our SESD protocol scales, we run experiments on datasets with size $n \in \{2^{12}, 2^{16}\}$, number of clusters $K \in \{4, 8, 16\}$, and number of iterations $T \in \{10, 20\}$. The field size is set to $\ell = 2^{32}$ and the dimensions of the data is fixed to be $d = 2$. We note that ℓ and d do not affect the comparison with previous works.

Table 2 shows the running time (in millisecond) to perform a SESD, and the number of OT instances needed. Recall that SESD from the i^{th} point to the k^{th} cluster is equal to

$$\begin{aligned}\mathcal{F}_{\mathsf{EDist}}(\llbracket \mathbf{P}_i \rrbracket, \llbracket \boldsymbol{\phi}_k \rrbracket) = &\sum_{\rho=1}^{d} (\mathbf{P}_i^A[\rho] - \boldsymbol{\phi}_k^A[\rho])^2 + \sum_{\rho=1}^{d} (\mathbf{P}_i^B[\rho] - \boldsymbol{\phi}_k^B[\rho])^2 \\ &+ 2 \sum_{\rho=1}^{d} (\mathbf{P}_i^A[\rho] - \boldsymbol{\phi}_k^A[\rho])(\mathbf{P}_i^B[\rho] - \boldsymbol{\phi}_k^B[\rho])) \end{aligned} \tag{4}$$

All previous privacy preserving clustering protocols [15, 38, 40] use a standard secure multiplication (based on garbled circuit or homomorphic encryption) to compute the mixed term of the above equation. Recently, [44] proposed and implemented SESD using the state-of-the-art secure multiplication [21]. The baseline in Figure 3 shows the performance of [44]. We obtain the baseline measurements by running the implementation of ABY [3, 44]. We note that in the baseline the running time per SESD does not depend on n, K, T since this solution computes the mixed term independently from other SESD instances. Therefore, this solution requires $\ell dnKT$ instances of OTs in total. For a database of size $n = 2^{16}$, $K = 16$, and $T = 20$, this baseline requires around 2^{30} OT instances, which does not scale well.

The mixed term can be written as the formula (2). We observe that in each t^{th} iteration, parties perform K secure multiplications with the same factor $\mathbf{P}_i^A$ or $\mathbf{P}_i^B$, thus, we compute our new SESD formula (2) by using the technique of [57]. Concretely, all OT sender messages can be concatenated before encrypting and sending them to other party. The column "Our Amortized" in Table 2 and Table 7 (in Appendix) present the performance of this optimization. For $n = 2^{16}$, $K = 4$, and $T = 20$, we obtain an overall running time of 0.172 ms per SESD in the amortized setting. Increasing the number of cluster from 4 to 16, our protocol shows a factor of 3.44× improvement in terms of running time, due to the fact that it amortizes well. This solution requires $\ell d(2n + K)T$ instances of OTs in total. Note that the technique of [57] cannot be directly applied to compute SESD without breaking down $\mathcal{F}_{\mathsf{EDist}}(\llbracket \mathbf{P} \rrbracket, \llbracket \boldsymbol{\phi} \rrbracket)$ into our formula (2).

Moreover, [57] only supports multiplying one *fixed* value by many *fixed* values. In the K-means application, the centroids are changing after each iteration, thus our final secure multiplication protocol allows these values to change during the execution. The column "Adaptive Amortized" shows its performance (described in Section 4.2), where parties can reused OT instances across all iterations. Our experiments show that our SESD is highly scalable. For a database of size $n = 2^{16}$, $K = 16$, and $T = 20$, our protocol requires around 2^{23} OT instances, which is 159× lower than that of the baseline. In terms of running time, our protocol requires only 0.135 ms to compute a SESD in the adaptive amortized setting with $n = 2^{16}$, $K = 4$, and $T = 10$. For the same n, when

Parameters			RunTime(ms) per SESD			
n	K	T	Baseline [44]	Our Amortized	Our Adaptive Amortized	Improved Factor
2^{12}	4	10	4.398	0.212	0.094	47×
		20		0.152	0.079	55.7×
	16	10		0.062	0.036	122.5×
		20		0.061	0.031	142.5×
2^{16}	4	10		0.235	0.135	32.5×
		20		0.172	0.093	47.5×
	16	10		0.051	0.033	134.2×
		20		0.05	0.03	148.1×

Table 2: Running time in millisecond per SESD (described in Figure 3), where n, K is the size of database, T is number of iterations, dimension $d = 2$, and bit-length $\ell = 32$.

Distance Metric		Dimension d			
		2	3	4	10
Manhattan		1.163	1.623	1.96	4.763
Chessboard		1.222	1.711	2.294	5.791
SESD	$\{K = 4, T = 10\}$	0.094	0.155	0.219	0.474
	$\{K = 4, T = 20\}$	0.079	0.123	0.164	0.398
	$\{K = 16, T = 10\}$	0.036	0.043	0.066	0.172
	$\{K = 16, T = 20\}$	0.031	0.042	0.063	0.163

Table 3: Running time in millisecond per a distance metric with d dimension, and bit-length $\ell = 32$. In our SESD protocol, $n = 2^{12}, K$ are the size of database, T is number of iterations.

increasing the number of cluster to $K = 16$, and the number of iteration to $T = 20$, our protocol running time is 0.03 ms per SESD.

Of particular interest is the column "Improved Factor", which presents the ratio between the runtime and the number OT required of the baseline and our scheme. Our protocol yields a better speedup when the dataset size and number of iterations are larger. For smallest dataset size of $n = 2^{12}, K = 4$ and $T = 10$, the protocol achieves a speed up of about 47×. When considering the larger database size $n = 2^{16}, K = 16$, the speed up of 134.1 × is obtained for $T = 10$ and 148.1× at $T = 20$ iterations.

7.3.2 Approximation of Euclidean distance

As discussed in Section 5.2.1, Manhattan metric and Chessboard metric (norm-∞) are considered as alternative distance metrics in some ML applications. We implement these distance metric by employing a generic secure computation, and compare their performance with our SESD. We note that Manhattan metric is used in the privacy-preserving clustering protocol of [40]. We benchmark these distance protocols and present their runtime in Table 3. It is not clear how to compute these distance metrics in the amortized setting. Thus, the parameters n, K, T do not affect their cost.

The running time to measure Manhattan and Chessboard distance is similar in the low-dimension space. It dues to the fact that secure Chessboard distance computation requires a small number of the "maximum" gabled circuits. Computing Manhattan or Chessboard distance between two 3-dimensional vectors takes around 1.7ms. Increasing the dimension from $d = 3$ to 10, the Manhattan distance computation costs 4.7 ms while the Chessboard distance computation requires 5.7ms.

It is easy to see from Table 3, our amortized SESD cost is $8.9 \times - 38.5\times$ faster than the cost of computing a Manhattan distance, and $10.5 \times - 40.5\times$ faster than that of Chessboard distance. We note that our SESD is amortized well in both adaptive and non-adaptive setting. When executing more and more SESD (between one fix point and other points), the cost drops dramatically to few microseconds per SESD. Therefore, we use SESD in our experiments for privacy-preserving clustering.

7.4 Experiments for Clustering

In this section we present our experimental results of our privacy-preserving clustering protocol. We ran our experiments on a large number of synthetic data sets to show the practicality and scalability. We also benchmark our scheme on the real dataset for comparison with previous work.

The offline phase includes the base OTs, n instances of OT extensions. We generate garbled circuits and OT extensions needed for $\mathcal{F}^K_{\mathsf{min}}$ executed in the online phase (even these steps can be performed in the offline phase).

7.4.1 Scalability

Experiments with generated dataset We generate 2-dimensional synthetic data sets on the range of set sizes $n \in \{10000, 100000\}$. Our synthetic data set generator takes a number of cluster $K \in \{2, 5\}$. There exist various criteria to stop iterations in K-means. In this experiment, we simply set the number of iterations to a fixed value (say, $T \in \{10, 20\}$).

We report both the running time and the communication cost of our scheme in Table 4. We recall that our scheme consists of three major phases, which are distance phase, assignment phase, and update phase (as described in Figure 6). To understand the performance of each phase, we also report their empirical results in Table 4. The main cost of our scheme comes from the second phase, where we need to evaluate $(n-1)$ "less than" garble circuits. To save time in evaluating this phase a case of $n = 10^5$ and $K = 5$, instead of running it in every iteration, we measure its runtime for one round iteration, and multiply by the number of iterations T.

As shown in Table 4, our scheme is practical. Small-size problems are few minutes; and larger size problems ($n = 100,000$) is under 2 hours, all *single-threaded*. In particular, it only takes 1.92 minutes to train a clustering model securely on 10,000 data samples with 2 clusters. From 1.92 minutes needed for privacy preserving training, only a small portion is spent on the distance and update phases. Our scheme is mostly based on symmetric-key operations, it introduces a overhead on the communication, namely 2.5GB for $n = 10,000$. When $n = 100,000$ and $K = 5$, our protocol takes 115.78 minutes to train the model, in which 81.5% of the total runtime comes from the assignment phase.

It is worth noting that our protocol is amenable to parallelization. Specifically, the computing SESD and updating the centroids steps can be parallelized. Moreover, one can compute $\mathcal{F}^K_{\mathsf{min}}$ in parallel per branch of the tree (see Figure 5). Therefore, we expect that our protocol can securely cluster a large dataset of $n = 100,000$ under 10 minutes using 16 threads. In addition, we estimate that our scheme can deal with datasets of millions points by using several servers (e.g. 10 servers).

Parameters			RunTime (minute)				Communication (MB)			
n	K	T	Distance (SESD)	Assign Points to Clusters	Update Centroids	Total	Distance (SESD)	Assign Points to Clusters	Update Centroids	Total
10^4	2	10	0.65	1.14	0.13	1.92	200	2330	10	2559
		20	0.95	2.29	0.26	3.5	398	4660	20	5097
	5	10	0.73	4.61	0.47	5.81	496	8760	40	9316
		20	1.18	9.23	0.94	11.35	989	17520	80	18609
10^5	2	10	5.69	11.12	1.2	18.02	1932	21400	140	23671
		20	10.38	22.25	2.4	35.04	3985	42800	280	47264
	5	10	5.77	47.18	5.13	58.09	9927	85630	340	96096
		20	11.13	94.35	10.27	115.78	4969	171260	680	177108

Table 4: Total running time in minute and communication cost of our privacy-preserving clustering protocol, where n, K is the size of database and the number of clusters, respectively, T is number of iterations, dimension $d = 2$, and bit-length $\ell = 32$.

Dataset	Setting	Lsun			S1
		Exact ($T = 15$)	Stabilized ($T = 40$)	Approximate ($T = 40$)	($T = 30$)
[75]	DP	-	-	-	23.18**s**
[40]	SH	545.91**d**	15.56**h**	15.47**h**	-
Ours		22.21**s**	48.9**s**	48.9**s**	1472.6**s**

Table 5: Comparison of total runtime between our protocol and [75, 40] on dataset Lsun and S1. T is number of iterations. "DP" and "SH" denote differential-privacy and semi-honest setting. "**s**", "**h**", and "**d**" denote second, hour and day, respectively. Cells with "-" denote the runtime not given.

Experiments with different dimensions From the breakdown performance of our scheme in Table 4, we can observe that the majority (70-80%) of the total cost is for the "Assign Points to Cluster" step, which is independent of the dimension because distance metric is a number. Additionally, from Table 2, it can seen that the cost of computing SESD increases 5× as the dimension increases from 2 to 10, which implies that the running time of SESD is almost linear in the dimension.

To understand the impact of different dimensions on the total cost of our scheme, we evaluate our protocol using DIM-set with different values of $d \in \{2, 4, 6, 8, 10\}$. The number of iterations T is set to be 15. The numerical results are reported in Figure 7. It can be observed that increasing d affects only the SESD and "Centroid Update" phases in the clustering algorithm. For smaller d, these two phases account for only a small portion of total running-time of our scheme. When increasing dimension to 10, these phases takes 30% of the total cost.

7.4.2 Comparison with Prior Work

We compare our prototype to the state-of-art privacy-preserving clustering protocols of Jäschke and Armknecht [40], and differential privacy clustering protocols of Su *et al.* [74]. Since implementation of the work [40] is not publicly available, we use their reported experimental numbers.

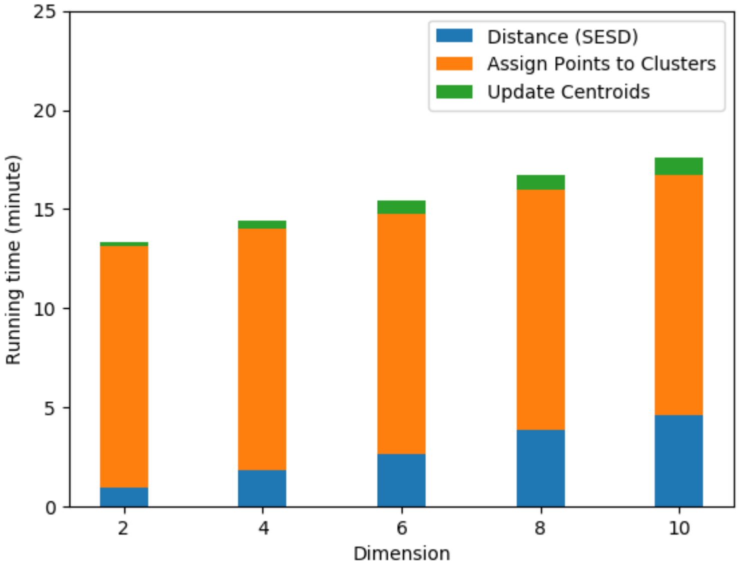


Figure 7: Running time (in minute) of our privacy-preserving clustering protocol on dataset DIM-set, where dataset size is 10,000 points, dimension is $d \in \{2, 4, 6, 8, 10\}$, the number of cluster and iterations are 9 and 15, resp.

Comparison with Jäschke and Armknecht's scheme [40] For the most direct comparison, we perform a comparison on the Lsun dataset [5] to match the dataset used in [[40], Table 2]. We also matched the test system's computational performance to that of [40]. Since [40] ran experiments on Intel i7-3770, 3.4 GHz, 20GB RAM; we use a similar (1.32× slower) machine as reported in Section 7.1. Table 5 presents the running time of our protocol compared with [40]. The work of [40] evaluate three different versions of privacy-preserving K-means clustering algorithm.

The first scheme [40] is exact K-means algorithm, in which the authors use TFHE library [6] to implement ciphertext division $\frac{c_1}{c_2}$, where both c_1 and c_2 are ciphertexts. This is a needed operation in the update phase which recalculates the new cluster center by taking the average of the values of the point's attributes that are part of the cluster. The authors encoded each data entry with 35 bits, in which 20 bits are used for the numbers after the decimal point. In our experiment, we use 32 bits to encode the data entry and use garble circuits to implement the ciphertext division operation. We fix the number of iterations to be $T = 15$ rounds, which is the same as in the experiment in [40]. As shown in Table 5, the protocol [40] costs 545.91 days to train Lsun dataset while our scheme requires only 22.21 seconds (i.e, five orders of magnitude faster than [40]).

Since the main computational cost of their exact version comes from the division operation where both numerator and denominator are ciphertext, the authors modify the update phase of K-means algorithm to have denominator to be a constant number. Concretely, their new k^{th} cluster center can be computed by $\frac{\sum_{i=1}^{n} \mathbf{P}'_i}{n}$, where $\mathbf{P}'_i$ is exactly the data entry $\mathbf{P}_i$ if this data entry is assigned to the k^{th} centroid, otherwise, $\mathbf{P}'_i$ is equal to the old centroid value $\boldsymbol{\phi}_k$. They call this algorithm the stabilized K-means. Since the centroids move more slowly in this scheme, the experiment [40] chooses $T = 40$ iterations which is also used in our experiment. Section 7.4.1 shows that our update phase takes only a small portion of the total runtime, therefore, we do not apply the stabilized technique [40] in our protocol (which is in favor of [40]). As shown in Table 5, the protocol [40] costs 15.56 hours to train Lsun dataset while our scheme requires 48.9 seconds, an approximate 1145× improvement.

The third scheme [40] is approximate K-means algorithm, where Euclidean distance is replaced by Manhattan distance. This modification speeds up the runtime of the protocol [40]. However, as discussed in Section 5.2.1, the amortized cost of our SESD is much better than that of Manhattan, thus we use SESD in our experiment. We fix the number of iterations to be $T = 40$ rounds, which is also used in the experiment [40]. Our experimental results show that our clustering scheme is 1138× faster than the third version of privacy-preserving K-mean clustering algorithm [40].

Comparison with Su *et al.* scheme [75] We note that this work focus on MPC-based methods, which have a very different security definition and requirement and design objective compared to the extensively-studied differential privacy (DP) based techniques. In DP, privacy and security are defined in terms of statistical and information theory senses. We do not intend to give a detailed comparison between MPC-based and DP-based methods because the settings and design goals are different (comparing apples and oranges). We only briefly compare running time of these two methods in Table 5 to examine the performance gap between our semi-honest scheme and DP security model. The experimental results on 2D synthetic dataset S1 [26] show that our privacy-preserving K-means clustering scheme is only 63.5× slower than the differential privacy model [75]. We include more discussion in Appendix B.1.

7.4.3 Accuracy

The accuracy is the percentage of entities in the evaluation set grouped correctly. In this section, we compare the accuracy of the produced models using our proposed approach and the plain-text K-means clustering algorithm (i.e., without privacy). For a visual comparison, we use the 2D dataset from arff [2] and S1 [26], which have the actual labels or ground truth centroids shown in Appendix C (Figure 10a and Figure 11a, respectively). We evaluate the plain-text algorithm and our privacy-preserving scheme, and present the obtained groupscentroids in Appendix C (Figure 10b, 11b, 11c).

All functions employed in our framework is the same as the original functions used the plain-text K-means clustering, except the update phase (step 3c in Figure 6), where we truncate the fractional part of the new cluster centroid to obtain an integer. Note that we use the truncation technique mentioned in [57]. The experimental results show that the truncation has a negligible impact on model accuracy compared to the original function. Our scheme with truncation reaches the same accuracy compared to a plain-text K-means clustering on decimal numbers. When training dataset arff using both our privacy preserving approach and plain-text K-mean algorithm, 95% of entities have been grouped correctly compared to the ground truth model.

Indeed, the K-means algorithm itself already has certain errors. A well-known disadvantage of the K-means algorithm is that its performance lacks of consistency. A random choice of cluster centers at the initialization step may result in different clusters since the algorithm can be stuck in a local optimum and may not converge to the global optimum. Therefore, in practice, we often run the algorithm with different initializations of centroids, and then pick the result of the run that yields the lowest sum of squared distance. Hence, we also compare our privacy-preserving model to the plaintext k-mean algorithm on S1.

Given the ground-truth of dataset S1, we find the best matching from each obtained centroid to them. We calculate the Euclidean distance between each obtained centroid and all ground truth centroids, map each obtained centroid to the ground truth centroid whose distance is the minimum among all the ground truth centroids. We note that a ground-truth model is often not available in practice since clustering is an unsupervised learning method. As shown in Figure 11 (Appendix C), both models produce the same accuracy ranging from 86% to 99% depending on the initially chosen centroids.

In all experiments, our privacy-preserving model achieves the same accuracy that the plaintext algorithm does. It can be explained as follows: the loss of accuracy of our protocol compared to the plaintext algorithm is solely due to the garble-circuit based-division operation as mentioned above. Moreover, the roundoff/truncation error is very small (e.g. 10^{-4}). Therefore, the loss of accuracy does not occur in our experiments. However, in the case this error is not very small, we would like to note that the accuracy of our model can be improved by increasing the number of digits used in the division operation. Specifically, during the truncation step, if we keep more digits, the accuracy of our scheme will increase, however, it requires more computation/communication cost. There is a tradeoff between computational time and accuracy of the division operation.

8 Conclusion

In this paper, we presented a novel privacy-preserving K-means clustering scheme with an efficient batched secure squared Euclidean distance and a customized garbled circuit to compute the binary secret sharing of the minimum value among a list of secret shared values.

Although our proposed protocol currently takes about 2 hours for clustering 100,000 records, it is practical for many real-world scenarios where privacy is critical to the parties (e.g., medical records of patients/hospitals, customer databases of companies) and the clustering algorithm runs periodically (i.e., no need to be real-time). For example, two hospitals/companies usually just want to train a model in a weekly, monthly, or even quarterly or yearly basis to update knowledge about patients/customers/items in their joint database, to have efficient and dedicated treatments/marketing schemes targeted/tailored for each subgroup (i.e., each cluster). More importantly, it is worth emphasizing that the proposed algorithm is already *five orders of magnitude faster* than the state-of-the-art work. We believe our work can serve as an important step to facilitate the development of faster privacy-preserving clustering algorithms in the future, especially through the lens of MPC.

We finally describe three directions for future work: improving scalability, other applications of secure squared Euclidean distance, and extension to malicious adversaries.

- The current implementation of our scheme only uses single-thread while our proposed protocol can be implemented in a parallel fashion. To enhance scalability, we plan to implement each step of the protocol (Figure 6) in parallel, which allows the scheme to deal with a big dataset (e.g. million points) in minutes.

- We believe that the construction of our SSED is of independent interests, and can be used in many applications such as k-nearest neighbors or face recognition. However, we usually need to tailor the construction for each specific problem.

- Another direction is to extend our scheme to the malicious adversarial setting where the combination of malicious secure computation protocols is a non-trivial problem. For example, it is not known how to efficiently verify parties that use the same shares from the previous k-mean step to the next steps. One promising direction is to investigate the malicious secure SPDZ protocol [20] which uses MACs to achieve malicious security.

References

[1] http://mint.sbg.ac.at/.

[2] https://github.com/deric/clustering-benchmark.

[3] https://github.com/encryptogroup/aby/.

[4] https://github.com/ladnir/ivory-runtime.

[5] http://www.uni-marburg.de/fb12/datenbionik/downloads/fcps.

[6] Tfhe library: https://tfhe.github.io/tfhe.

[7] Gilad Asharov, Yehuda Lindell, Thomas Schneider, and Michael Zohner. More efficient oblivious transfer and extensions for faster secure computation. In Ahmad-Reza Sadeghi, Virgil D. Gligor, and Moti Yung, editors, *ACM CCS 13*, pages 535–548. ACM Press, November 2013.

[8] Maria-Florina Balcan, Travis Dick, Yingyu Liang, Wenlong Mou, and Hongyang Zhang. Differentially private clustering in high-dimensional Euclidean spaces. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 322–331, International Convention Centre, Sydney, Australia, 06–11 Aug 2017. PMLR.

[9] Assi Barak, Daniel Escudero, Anders Dalskov, and Marcel Keller. Secure evaluation of quantized neural networks. Cryptology ePrint Archive, Report 2019/131, 2019. https://eprint.iacr.org/2019/131.

[10] Donald Beaver. Correlated pseudorandomness and the complexity of private computations. In *28th ACM STOC*, pages 479–488. ACM Press, May 1996.

[11] Mihir Bellare, Viet Tung Hoang, Sriram Keelveedhi, and Phillip Rogaway. Efficient garbling from a fixed-key blockcipher. In *2013 IEEE Symposium on Security and Privacy*, pages 478–492. IEEE Computer Society Press, May 2013.

[12] Assaf Ben-David, Noam Nisan, and Benny Pinkas. FairplayMP: a system for secure multi-party computation. In Peng Ning, Paul F. Syverson, and Somesh Jha, editors, *ACM CCS 08*, pages 257–266. ACM Press, October 2008.

[13] Michael W. Berry, Umeshwar Dayal, Chandrika Kamath, and David B. Skillicorn, editors. *Proceedings of the Fourth SIAM International Conference on Data Mining, Lake Buena Vista, Florida, USA, April 22-24, 2004*. SIAM, 2004.

[14] Marina Blanton and Paolo Gasti. Secure and efficient protocols for iris and fingerprint identification. In Vijay Atluri and Claudia Díaz, editors, *ESORICS 2011*, volume 6879 of *LNCS*, pages 190–209. Springer, Heidelberg, 2011.

[15] Paul Bunn and Rafail Ostrovsky. Secure two-party k-means clustering. In Peng Ning, Sabrina De Capitani di Vimercati, and Paul F. Syverson, editors, *ACM CCS 07*, pages 486–497. ACM Press, October 2007.

[16] Ran Canetti, Ivan Damgαrd, Stefan Dziembowski, Yuval Ishai, and Tal Malkin. Adaptive versus non-adaptive security of multi-party protocols. *Journal of Cryptology*, 17(3):153–207, June 2004.

[17] Hervé Chabanne, Amaury de Wargny, Jonathan Milgram, Constance Morel, and Emmanuel Prouff. Privacy-preserving classification on deep neural network. Cryptology ePrint Archive, Report 2017/035, 2017. http://eprint.iacr.org/2017/035.

[18] Hao Chen, Ilaria Chillotti, Yihe Dong, Oxana Poburinnaya, Ilya Razenshteyn, and M. Sadegh Riazi. Sanns: Scaling up secure approximate k-nearest neighbors search. Cryptology ePrint Archive, Report 2019/359, 2019. https://eprint.iacr.org/2019/359.

[19] Hao Chen, Ran Gilad-Bachrach, Kyoohyung Han, Zhicong Huang, Amir Jalali, Kim Laine, and Kristin Lauter. Logistic regression over encrypted data from fully homomorphic encryption. Cryptology ePrint Archive, Report 2018/462, 2018. https://eprint.iacr.org/2018/462.

[20] Ivan Damgαrd, Valerio Pastro, Nigel P. Smart, and Sarah Zakarias. Multiparty computation from somewhat homomorphic encryption. In Reihaneh Safavi-Naini and Ran Canetti, editors, *CRYPTO 2012*, volume 7417 of *LNCS*, pages 643–662. Springer, Heidelberg, August 2012.

[21] Daniel Demmler, Thomas Schneider, and Michael Zohner. ABY - A framework for efficient mixed-protocol secure two-party computation. In *NDSS 2015*. The Internet Society, February 2015.

[22] Ghada Dessouky, Farinaz Koushanfar, Ahmad-Reza Sadeghi, Thomas Schneider, Shaza Zeitouni, and Michael Zohner. Pushing the communication barrier in secure computation using lookup tables. In *NDSS 2017*. The Internet Society, February / March 2017.

[23] Cynthia Dwork. Differential privacy (invited paper). In Michele Bugliesi, Bart Preneel, Vladimiro Sassone, and Ingo Wegener, editors, *ICALP 2006, Part II*, volume 4052 of *LNCS*, pages 1–12. Springer, Heidelberg, July 2006.

[24] Cynthia Dwork, Moni Naor, Guy N. Rothblum, and Vinod Vaikuntanathan. How efficient can memory checking be? In Omer Reingold, editor, *TCC 2009*, volume 5444 of *LNCS*, pages 503–520. Springer, Heidelberg, March 2009.

[25] Alexandre Evfimievski, Ramakrishnan Srikant, Rakesh Agrawal, and Johannes Gehrke. Privacy preserving mining of association rules. In *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '02, pages 217–228, New York, NY, USA, 2002. ACM.

[26] Pasi Fränti and Sami Sieranoja. K-means properties on six clustering benchmark datasets, 2018.

[27] Craig Gentry. *A Fully Homomorphic Encryption Scheme.* PhD thesis, Stanford, CA, USA, 2009. AAI3382729.

[28] Z. Gheid and Y. Challal. Efficient and privacy-preserving k-means clustering for big data mining. In *2016 IEEE Trustcom/BigDataSE/ISPA*, pages 791–798, Aug 2016.

[29] Ran Gilad-Bachrach, Kim Laine, Kristin Lauter, Peter Rindal, and Mike Rosulek. Secure data exchange: A marketplace in the cloud. CCSW, 2019.

[30] Niv Gilboa. Two party RSA key generation. In Michael J. Wiener, editor, *CRYPTO'99*, volume 1666 of *LNCS*, pages 116–129. Springer, Heidelberg, August 1999.

[31] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or A completeness theorem for protocols with honest majority. In Alfred Aho, editor, *19th ACM STOC*, pages 218–229. ACM Press, May 1987.

[32] S. Dov Gordon, Jonathan Katz, Vladimir Kolesnikov, Fernando Krell, Tal Malkin, Mariana Raykova, and Yevgeniy Vahlis. Secure two-party computation in sublinear (amortized) time. In Ting Yu, George Danezis, and Virgil D. Gligor, editors, *ACM CCS 12*, pages 513–524. ACM Press, October 2012.

[33] Sudipto Guha, Adam Meyerson, Nina Mishra, Rajeev Motwani, and Liadan O'Callaghan. Clustering data streams: Theory and practice. *IEEE Trans. on Knowl. and Data Eng.*, 15(3):515–528, March 2003.

[34] Sudipto Guha, Rajeev Rastogi, and Kyuseok Shim. Cure: An efficient clustering algorithm for large databases. In *Proceedings of the 1998 ACM SIGMOD International Conference on Management of Data*, SIGMOD '98, pages 73–84, New York, NY, USA, 1998. ACM.

[35] Rob Hall, Yuval Nardi, and Stephen E. Fienberg. Achieving both valid and secure logistic regression analysis on aggregated data from different private sources. *CoRR*, abs/1111.7277, 2011.

[36] Yan Huang, Lior Malka, David Evans, and Jonathan Katz. Efficient privacy-preserving biometric identification. In *NDSS 2011*. The Internet Society, February 2011.

[37] Yuval Ishai, Joe Kilian, Kobbi Nissim, and Erez Petrank. Extending oblivious transfers efficiently. In Dan Boneh, editor, *CRYPTO 2003*, volume 2729 of *LNCS*, pages 145–161. Springer, Heidelberg, August 2003.

[38] Geetha Jagannathan, Krishnan Pillaipakkamnatt, Rebecca N. Wright, and Daryl Umano. Communication-efficient privacy-preserving clustering. *Trans. Data Privacy*, 3(1):1–25, April 2010.

[39] Geetha Jagannathan and Rebecca N. Wright. Privacy-preserving distributed k-means clustering over arbitrarily partitioned data. In *Proceedings of the Eleventh ACM SIGKDD International Conference on Knowledge Discovery in Data Mining*, KDD '05, pages 593–599, New York, NY, USA, 2005. ACM.

[40] Angela Jäschke and Frederik Armknecht. Unsupervised machine learning on encrypted data. In Carlos Cid and Michael J. Jacobson Jr., editors, *Selected Areas in Cryptography – SAC 2018*, pages 453–478, Cham, 2019. Springer International Publishing.

[41] Somesh Jha, Luis Kruger, and Patrick McDaniel. Privacy preserving clustering. In Sabrina de Capitani di Vimercati, Paul Syverson, and Dieter Gollmann, editors, *Computer Security – ESORICS 2005*, pages 397–417, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.

[42] Xiaoqian Jiang, Miran Kim, Kristin Lauter, and Yongsoo Song. Secure outsourced matrix computation and application to neural networks. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, CCS '18, pages 1209–1222, New York, NY, USA, 2018. ACM.

[43] Zoe Jiang, Ning Guo, Yabin Jin, Jiazhuo Lv, Yulin Wu, Yating Yu, Xuan Wang, Sm Yiu, and Junbin Fang. Efficient two-party privacy preserving collaborative k-means clustering protocol supporting both storage and computation outsourcing: 18th international conference, ica3pp 2018, guangzhou, china, november 15-17, 2018, proceedings, part iv. pages 447–460, 11 2018.

[44] K. Järvinen, H. Leppäkoski, E. Lohan, P. Richter, T. Schneider, O. Tkachenko, and Z. Yang. Pilot: Practical privacy-preserving indoor localization using outsourcing. In *2019 IEEE European Symposium on Security and Privacy (EuroS P)*, pages 448–463, June 2019.

[45] Chiraag Juvekar, Vinod Vaikuntanathan, and Anantha Chandrakasan. Gazelle: A low latency framework for secure neural network inference. In *Proceedings of the 27th USENIX Conference on Security Symposium*, SEC'18, pages 1651–1668, Berkeley, CA, USA, 2018. USENIX Association.

[46] Seny Kamara, Payman Mohassel, and Mariana Raykova. Outsourcing multi-party computation. Cryptology ePrint Archive, Report 2011/272, 2011. http://eprint.iacr.org/2011/272.

[47] I. Kärkkäinen and P. Fränti. Gradual model generator for single-pass clustering. *Pattern Recognition*, 40(3):784–795, 2007.

[48] Vladimir Kolesnikov and Ranjit Kumaresan. Improved OT extension for transferring short secrets. In Ran Canetti and Juan A. Garay, editors, *CRYPTO 2013, Part II*, volume 8043 of *LNCS*, pages 54–70. Springer, Heidelberg, August 2013.

[49] Vladimir Kolesnikov, Ranjit Kumaresan, Mike Rosulek, and Ni Trieu. Efficient batched oblivious PRF with applications to private set intersection. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *ACM CCS 16*, pages 818–829. ACM Press, October 2016.

[50] Vladimir Kolesnikov, Jesper Buus Nielsen, Mike Rosulek, Ni Trieu, and Roberto Trifiletti. DUPLO: Unifying cut-and-choose for garbled circuits. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *ACM CCS 17*, pages 3–20. ACM Press, October / November 2017.

[51] Vladimir Kolesnikov and Thomas Schneider. Improved garbled circuit: Free xor gates and applications. In Luca Aceto, Ivan Damgαrd, Leslie Ann Goldberg, Magnús M. Halldórsson, Anna Ingólfsdóttir, and Igor Walukiewicz, editors, *Automata, Languages and Programming*, pages 486–498, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.

[52] Yehuda Lindell and Benny Pinkas. Privacy preserving data mining. In Mihir Bellare, editor, *CRYPTO 2000*, volume 1880 of *LNCS*, pages 36–54. Springer, Heidelberg, August 2000.

[53] X. Liu, Z. L. Jiang, S. M. Yiu, X. Wang, C. Tan, Y. Li, Z. Liu, Y. Jin, and J. Fang. Outsourcing two-party privacy preserving k-means clustering protocol in wireless sensor networks. In *2015 11th International Conference on Mobile Ad-hoc and Sensor Networks (MSN)*, pages 124–133, Dec 2015.

[54] S. Lloyd. Least squares quantization in pcm. *IEEE Trans. Inf. Theor.*, 28(2):129–137, September 2006.

[55] Xianrui Meng, Dimitrios Papadopoulos, Alina Oprea, and Nikos Triandopoulos. Privacy-preserving hierarchical clustering: Formal security and efficient approximation. *CoRR*, abs/1904.04475, 2019.

[56] Payman Mohassel and Peter Rindal. Aby3: A mixed protocol framework for machine learning. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, CCS '18, pages 35–52, New York, NY, USA, 2018. ACM.

[57] Payman Mohassel and Yupeng Zhang. SecureML: A system for scalable privacy-preserving machine learning. In *2017 IEEE Symposium on Security and Privacy*, pages 19–38. IEEE Computer Society Press, May 2017.

[58] Valeria Nikolaenko, Stratis Ioannidis, Udi Weinsberg, Marc Joye, Nina Taft, and Dan Boneh. Privacy-preserving matrix factorization. In Ahmad-Reza Sadeghi, Virgil D. Gligor, and Moti Yung, editors, *ACM CCS 13*, pages 801–812. ACM Press, November 2013.

[59] Valeria Nikolaenko, Udi Weinsberg, Stratis Ioannidis, Marc Joye, Dan Boneh, and Nina Taft. Privacy-preserving ridge regression on hundreds of millions of records. In *2013 IEEE Symposium on Security and Privacy*, pages 334–348. IEEE Computer Society Press, May 2013.

[60] Michele Orru, Emmanuela Orsini, and Peter Schol. Actively secure 1-out-of-n ot extension with application to private set intersection. In *CT-RSA*, 2017.

[61] Sankita Patel, Sweta Garasia, and Devesh Jinwala. An efficient approach for privacy preserving distributed k-means clustering based on shamir's secret sharing scheme. In Theo Dimitrakos, Rajat Moona, Dhiren Patel, and D. Harrison McKnight, editors, *Trust Management VI*, pages 129–141, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[62] Benny Pinkas, Thomas Schneider, and Michael Zohner. Scalable private set intersection based on ot extension. In *ACM TOPS*, 2018.

[63] Michael O. Rabin. How to exchange secrets with oblivious transfer. Cryptology ePrint Archive, Report 2005/187, 2005. http://eprint.iacr.org/2005/187.

[64] F. Rao, B. K. Samanthula, E. Bertino, X. Yi, and D. Liu. Privacy-preserving and outsourced multi-user k-means clustering. In *2015 IEEE Conference on Collaboration and Internet Computing (CIC)*, pages 80–89, Oct 2015.

[65] M. Sadegh Riazi, Mohammad Samragh, Hao Chen, Kim Laine, Kristin Lauter, and Farinaz Koushanfar. Xonn: Xnor-based oblivious deep neural network inference. Cryptology ePrint Archive, Report 2019/171, 2019. https://eprint.iacr.org/2019/171.

[66] Peter Rindal. libOTe: an efficient, portable, and easy to use Oblivious Transfer Library. https://github.com/osu-crypto/libOTe.

[67] Boris Rozenberg and Ehud Gudes. Association rules mining in vertically partitioned databases. *Data Knowl. Eng.*, 59(2):378–396, November 2006.

[68] Ahmad-Reza Sadeghi, Thomas Schneider, and Immo Wehrenberg. Efficient privacy-preserving face recognition. In *Proceedings of the 12th International Conference on Information Security and Cryptology*, ICISC'09, pages 229–244, Berlin, Heidelberg, 2010. Springer-Verlag.

[69] Ashish P. Sanil, Alan F. Karr, Xiaodong Lin, and Jerome P. Reiter. Privacy preserving regression modelling via distributed computation. In *Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '04, pages 677–682, New York, NY, USA, 2004. ACM.

[70] Phillipp Schoppmann, Adrià Gascón, and Borja Balle. Private nearest neighbors classification in federated databases. Cryptology ePrint Archive, Report 2018/289, 2018. https://eprint.iacr.org/2018/289.

[71] Phillipp Schoppmann, Adria Gascon, Mariana Raykova, and Benny Pinkas. Make some room for the zeros: Data sparsity in secure distributed machine learning. ACM CCS, 2019.

[72] Arlei Silva and Gowtham Bellala. Privacy-preserving multi-party clustering: An empirical study. pages 326–333, 06 2017.

[73] Jinhyun So, Basak Guler, A. Salman Avestimehr, and Payman Mohassel. Codedprivateml: A fast and privacy-preserving framework for distributed machine learning. Cryptology ePrint Archive, Report 2019/140, 2019. https://eprint.iacr.org/2019/140.

[74] Dong Su, Jianneng Cao, Ninghui Li, Elisa Bertino, and Hongxia Jin. Differentially private k-means clustering. In *Proceedings of the Sixth ACM Conference on Data and Application Security and Privacy*, CODASPY '16, pages 26–37, New York, NY, USA, 2016. ACM.

[75] Dong Su, Jianneng Cao, Ninghui Li, Elisa Bertino, Min Lyu, and Hongxia Jin. Differentially private k-means clustering and a hybrid approach to private optimization. *ACM Trans. Priv. Secur.*, 20(4):16:1–16:33, October 2017.

[76] Jaideep Vaidya and Chris Clifton. Privacy-preserving k-means clustering over vertically partitioned data. In *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '03, pages 206–215, New York, NY, USA, 2003. ACM.

[77] Xiao Wang, Alex J. Malozemoff, and Jonathan Katz. EMP-toolkit: Efficient MultiParty computation toolkit. https://github.com/emp-toolkit/emp-tool/blob/master/emp-tool/circuits/float32_circuit.hpp#L37.

[78] Xiao Wang, Samuel Ranellucci, and Jonathan Katz. Authenticated garbling and efficient maliciously secure two-party computation. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *ACM CCS 17*, pages 21–37. ACM Press, October / November 2017.

[79] K. Xing, C. Hu, J. Yu, X. Cheng, and F. Zhang. Mutual privacy preserving k -means clustering in social participatory sensing. *IEEE Transactions on Industrial Informatics*, 13(4):2066–2076, Aug 2017.

[80] Andrew Chi-Chih Yao. How to generate and exchange secrets (extended abstract). In *27th FOCS*, pages 162–167. IEEE Computer Society Press, October 1986.

[81] J. Yuan and Y. Tian. Practical privacy-preserving mapreduce based k-means clustering over large-scale dataset. *IEEE Transactions on Cloud Computing*, 7(2):568–579, April 2019.

[82] Samee Zahur, Mike Rosulek, and David Evans. Two halves make a whole - reducing data transfer in garbled circuits using half gates. In Elisabeth Oswald and Marc Fischlin, editors, *EUROCRYPT 2015, Part II*, volume 9057 of *LNCS*, pages 220–250. Springer, Heidelberg, April 2015.

[83] Jun Zhang, Xiaokui Xiao, and Xing Xie. Privtree: A differentially private algorithm for hierarchical decompositions. In *Proceedings of the 2016 International Conference on Management of Data*, SIGMOD '16, pages 155–170, New York, NY, USA, 2016. ACM.

[84] Tian Zhang, Raghu Ramakrishnan, and Miron Livny. Birch: A new data clustering algorithm and its applications. *Data Mining and Knowledge Discovery*, 1(2):141–182, Jun 1997.

A Details of Our Building Blocks

A.1 Revising Communication-Efficient Secure multiplication Based on 1-out-of-N OT

Recently, several works [48, 49, 62, 60] have proposed efficient protocols to generalize 1-out-of-2 OT extension to 1-out-of-N OT, in which the the receiver learns one of the sender's N messages. To achieve 1-out-of-N OT, the main modification compared to the original IKNP scheme is the different kinds of encoding used to construct the IKNP OT extension matrices. While IKNP use a 128-bit repetition code, Kolesnikov and Kumaresan [48] employ 256-bit Walsh-Hadamard error-correcting code and achieve 1-out-of-N OT on random strings, for N up to approximately 256. The works [49, 62, 60] use either pseudo-random code or linear BCH code to achieve 1-out-of-N OT for large N. It is important to notice in the 1-out-of-N OT that the number of base OTs have to increase to the codeword length of the underlying code in order to obtain the same computational security level $\kappa = 128$ as in the original 1-out-of-2 OT IKNP. The reason is that the Hamming distance of two codewords has to be at least κ. For arbitrarily large N and arbitrarily bit length ℓ of OT messages, the best 1-out-of-N OT protocol [49] uses 424-448 bits codeword length, which requires 424-448 bits of communication per OT and N hash evaluations. For smaller ℓ, the best protocols [62, 60] use linear BCH code, in which codeword length depends on ℓ.

With recent improvement to 1-out-of-N OT, several works proposed to replace 1-out-of-2 OT with 1-out-of-N OT in some specific problems (e.g. Private Set Intersection [49, 62, 60]) to improve their performance. The work [22] proposed a communication efficient Beaver's triple generation which implicitly improves secure multiplication protocol. Our Section 4.1.1 explicitly described their construction. We now discuss the choice of OT variants, and parameter N.

There are two noteworthy aspects of the 1-out-of-N OT based protocol. First, the 1-out-of-N OT protocol of [49, 62, 60] is on random strings, in which the protocol itself "chooses" the OT messages $r_{i\in[N]}$ randomly, gives them to the sender and gives one chosen message r_b to the receiver. In this secure multiplication protocol, we need a standard 1-out-of-N OT protocol where the OT messages $m_{i\in[N]}$ are given by the sender. To achieve this OT variant, the sender requires to correct

				N			Improved
		2^1	2^2	2^3	2^4	2^8	Factor
Codeword length		128	192	224	240	255	
Comm. per OT	$\ell = 8$	136	108	105	**90**	288	1.51
	$\ell = 16$	288	**240**	252	**240**	1085	1.2
	$\ell = 32$	640	**576**	616	720	4209	1.11
	$\ell = 64$	**1536**	**1536**	1848	2400	16575	1

Table 6: Bit-length (in bit) of Linear Error Correcting Code (OT width) and the communication cost of secure multiplication (in byte) for 1-out-of-N OT of ℓ-bit strings.

the OT random messages by sending $c_i = r_i + m_i$ to the receiver, who later obtains the correct choice message m_b by subtracting r_b from the received c_i. This needed step increases the bandwidth requirement of the protocol. Thus, it is necessary to analyze what is the best value for N. Second, 1-out-of-2 OT-based protocol can use Correlated OT extension [7] since the sender's OT inputs $m_{i,0}, m_{i,1}$ are chosen randomly subject to $m_{i,0} + m_{i,1} = 2^i y$. Doing so reduces the communicational cost from the sender to the receiver by a factor of $\frac{\kappa+\ell}{\kappa+2\ell}$. This Correlated OT idea can be used in the 1-out-of-N OT-based protocol. As a result, we reduce the bandwidth requirement by a factor of $\frac{\kappa+(N-1)\ell}{\kappa+N\ell}$.

Table 6 presents the communication cost the the 1-out-of-N OT-based secure multiplication of two ℓ-bit strings. The required codeword length and the best error-correcting code are chosen according to [1] to achieve Hamming distance of two codewords at least κ. For short bit-length $\ell = 8$ or $\ell = 16$, Table 6 shows that using 1-out-of-2^4 OT gives us the best communication cost for secure multiplication, which is $1.2 - 1.51\times$ lower bandwidth requirement than the original 1-out-of-2 OT-based one. For bigger ℓ, an incremental improvement is achieved by employing 1-out-of-2^2 OT in the secure multiplication protocol.

A.2 Secure Euclidean Squared Distance (SESD) Details

The formal details of our secure Euclidean squared distance protocol are given in Figure 8. It closely follows and formalizes the detail presented in sections 4.2 and 4.1.2. The security of our construction follows in a straightforward way from the security of its building block (e.g. oblivious transfer) and the encryption scheme. Thus, we omit the proof of the following theorem.

Theorem 1. *The protocol in Figure 8 securely computes the Secure Euclidean Squared Distance (SESD) functionality (Figure 3) in semi-honest setting, given the ideal Oblivious Transfer (OT) defined Figure 1.*

B Privacy Preserving K-means Security Proof

Theorem 2. *The protocol in Figure 6 securely computes the K-means clustering functionality (Figure 2) in semi-honest setting, given the ideal Oblivious Transfer (OT), Euclidean Squared Distance (SESD), and Garbled Circuit (GC) primitives defined Figure 1, Figure 3, and Section 3.5, respectively.*

Parameters:

- Number of iterations K; number of clusters K; number of data points n; dimension d; value N
- Ideal OT primitive defined in Figure 1
- An encryption/decryption scheme $\mathsf{Enc}, \mathsf{Dec}$ (e.g. AES)

Input of Alice: Arithmetic secret shares $\mathcal{P}^A = \{\mathbf{P}_1^A, \mathbf{P}_2^A, \ldots, \mathbf{P}_n^A\}$ of n points $\mathcal{P} = \{\mathbf{P}_1, \mathbf{P}_2, \ldots, \mathbf{P}_n\}$
Input of Bob: Arithmetic secret shares $\mathcal{P}^B = \{\mathbf{P}_1^B, \mathbf{P}_2^B, \ldots, \mathbf{P}_n^B\}$ of n points $\mathcal{P}$

Protocol:

1. For each secret shares $\mathbf{P}_i^A \in \mathcal{P}^A$:
   - Alice uses an N-based representation and rewrites $\mathbf{P}_i^A = \sum_{j=0}^{\lceil \ell/\log(N)\rceil - 1} N^j \mathbf{P}_i^A[j]$
   - Parties invoke $\lceil \ell/\log(N)\rceil$ instances of 1-out-of-N OT as follows:
     - Bob acts as an OT sender with input sequence $(\Delta_{i,j,0}^B, \ldots, \Delta_{i,j,N-1}^B)$ which are randomly chosen from $\mathbb{Z}_{2^\ell}$
     - Alice acts as an OT receiver with choice value $\mathbf{P}_j^A[i]$, and obtains $\Delta_{i,j,\mathbf{P}_i^A[j]}^B$
2. For each secret shares $\mathbf{P}_i^B \in \mathcal{P}^B$:
   - Bob uses an N-based representation and rewrites $\mathbf{P}_i^B = \sum_{j=0}^{\lceil \ell/\log(N)\rceil - 1} N^j \mathbf{P}_i^B[j]$
   - Parties invoke $\lceil \ell/\log(N)\rceil$ instances of 1-out-of-N OT as follows:
     - Alice acts as an OT sender with input sequence $(\Delta_{i,j,0}^A, \ldots, \Delta_{i,j,N-1}^A)$ which are randomly chosen from $\mathbb{Z}_{2^\ell}$
     - Bob acts as an OT receiver with choice value $\mathbf{P}_j^B[i]$, and obtains $\Delta_{i,j,\mathbf{P}_i^B[j]}^A$
3. For each iteration $t \in T$:

   Input of Alice: Arithmetic secret shares $\{\boldsymbol{\phi}_{t1}^A, \boldsymbol{\phi}_{t2}^A, \ldots, \boldsymbol{\phi}_{tK}^A\}$ of K centroids $\{\boldsymbol{\phi}_{t1}, \ldots, \boldsymbol{\phi}_{tK}\}$
   Input of Bob: Arithmetic secret shares $\{\boldsymbol{\phi}_{t1}^B, \boldsymbol{\phi}_{t2}^B, \ldots, \boldsymbol{\phi}_{tK}^B\}$ of K centroids $\{\boldsymbol{\phi}_{t1}, \ldots, \boldsymbol{\phi}_{tK}\}$

   I. Sub-protocol: computing the first term of the mixed term in formula (2)

   (1) For each $i \in [N]$:
   - For each $k \in K$, Bob defines $y_{i,k} := \mathbf{P}_i^B[\rho] - \boldsymbol{\phi}_{tk}^B[\rho]$
   - For $j \in [\lceil \ell/\log(N)\rceil]$ and $\theta \in [N]$, Bob computes encryptions $e_{i,j,\theta} = \mathsf{Enc}(\Delta_{i,j,\theta}^B, m_{j,\theta,1} || m_{j,\theta,2} || \ldots || m_{j,\theta,K})$ where for all $m_{j,0,k\in[K]}$ is randomly chosen from $\mathbb{Z}_{2^\ell}$; for $1 \le \theta \le N-1$, $m_{j,\theta,k} = (N^j \theta y_{i,k} - m_{j,0,k}) \mod 2^\ell$

   (2) Bob sends to Alice the ciphertexts $e_{i,j,\theta}$ in order.

   (3) For each $i \in [N]$ and $j \in [\lceil \ell/\log(N)\rceil]$, Alice decrypts the ciphertexts $e_{i,j,\widetilde{\theta}}$ using the decrypted key $\Delta_{i,j,\widetilde{\theta}}^B$ where $\widetilde{\theta}_i := \mathbf{P}_i^A[j]$, and obtains $m_{j,\widetilde{\theta}_i,1} || m_{j,\widetilde{\theta}_i,2} || \ldots || m_{j,\widetilde{\theta}_i,K}$

   (4) For each $i \in [N]$ and $k \in [K]$, Alice locally computes $z_{i,k}^A := \sum_{j=0}^{\lceil \ell/\log(N)\rceil - 1} m_{j,\widetilde{\theta}_i,k} \mod 2^\ell$

   (5) For each $i \in [N]$ and $k \in [K]$, Bob locally computes $z_{i,k}^B := \sum_{j=0}^{\lceil \ell/\log(N)\rceil - 1} m_{j,0,k} \mod 2^\ell$ respectively.

   II. Sub-protocol: computing the second term of the mixed term in formula (2)

   (1) For each $i \in [N]$:
   - For each $k \in K$, Alice defines $y_{i,k} := \boldsymbol{\phi}_{tk}^A[\rho]$
   - For $j \in [\lceil \ell/\log(N)\rceil]$ and $\theta \in [N]$, Alice computes encryptions $e_{i,j,\theta} = \mathsf{Enc}(\Delta_{i,j,\theta}^A, m_{j,\theta,1} || m_{j,\theta,2} || \ldots || m_{j,\theta,K})$ where for all $m_{j,0,k\in[K]}$ is randomly chosen from $\mathbb{Z}_{2^\ell}$; for $1 \le \theta \le N-1$, $m_{j,\theta,k} = (N^j \theta y_{i,k} - m_{j,0,k}) \mod 2^\ell$

   (2) Alice sends to Bob the ciphertexts $e_{i,j,\theta}$ in order.

   (3) For each $i \in [N]$ and $j \in [\lceil \ell/\log(N)\rceil]$, Bob decrypts the ciphertexts $e_{i,j,\widetilde{\theta}}$ using the decrypted key $\Delta_{i,j,\widetilde{\theta}}^A$ where $\widetilde{\theta}_i := \mathbf{P}_i^B[j]$, and obtains $m_{j,\widetilde{\theta}_i,1} || m_{j,\widetilde{\theta}_i,2} || \ldots || m_{j,\widetilde{\theta}_i,K}$

   (4) For each $i \in [N]$ and $k \in [K]$, Bob locally computes $u_{i,k}^B := \sum_{j=0}^{\lceil \ell/\log(N)\rceil - 1} m_{j,\widetilde{\theta}_i,k} \mod 2^\ell$

   (5) For each $i \in [N]$ and $k \in [K]$, Alice locally computes $u_{i,k}^A := \sum_{j=0}^{\lceil \ell/\log(N)\rceil - 1} m_{j,0,k} \mod 2^\ell$ respectively.

   [noitemsep,topsep=0pt]

   III. Sub-protocol: outputting arithmetic secret sharings of the output $\mathcal{F}_{\mathsf{EDist}}(\mathbf{P}_i, \boldsymbol{\phi}_{tk})$

   (1) For each $k \in [K]$, Parties invoke a standard secure multiplication to compute the third terms $\boldsymbol{\phi}_{tk}^A[\rho]\boldsymbol{\phi}_{tk}^B[\rho]$ of Eq. (2), and obtains an output under a secret share form as v_k^A and v_k^B, respectively.

   (2) For each $i \in [N], k \in [K]$, Alice outputs $\sum_{\rho=1}^d (\mathbf{P}_i^A[\rho] - \boldsymbol{\phi}_{tk}^A[\rho])^2 + z_{i,k}^A + u_{i,k}^A + v_k^A$ as a secret share of $\mathcal{F}_{\mathsf{EDist}}(\mathbf{P}_i, \boldsymbol{\phi}_{tk})$

   (3) For each $i \in [N], k \in [K]$, Bob outputs $\sum_{\rho=1}^d (\mathbf{P}_i^B[\rho] - \boldsymbol{\phi}_{tk}^B[\rho])^2 + z_{i,k}^B + u_{i,k}^B + v_k^B$ as a secret share of $\mathcal{F}_{\mathsf{EDist}}(\mathbf{P}_i, \boldsymbol{\phi}_{tk})$

Figure 8: Our Secure Euclidean Squared Distance (SESD) Construction in the Adaptive Amortized Setting.

Proof. We exhibit a simulator Sim for simulating a corrupt party Alice. The simulator for Bob should be the same.

Sim simulates the view of corrupt Alice, which consists of her input/output and received messages. Sim proceeds as follows. It calls $\mathcal{F}_{\mathsf{EDist}}$ simulator $\mathsf{Sim}_{\mathcal{F}_{\mathsf{EDist}}}(\llbracket \mathbf{P}_i \rrbracket, \llbracket \boldsymbol{\phi}_k \rrbracket), \forall i \in [n], k \in [K]$, and then simulates step (II.b) by calling $\mathsf{Sim}_{\mathcal{F}^K_{\mathsf{min}}}(\llbracket X_{i,1} \rrbracket, \ldots, \llbracket X_{i,K} \rrbracket), \forall i \in [n]$, and appends its output to the general view. For step (II.c), Sim first computes the numerator/denominator using OT, runs simulator $\mathcal{GC}$, and appends its output to the view. We now argue the indistinguishability of the produced transcript from the real execution. For this, we formally show the simulation by proceeding the sequence of hybrid transcripts $T_0, \ldots, T_4$, where T_0 is real view of $\mathcal{C}$, and T_4 is the output of Sim.

Hybrid 1. Let T_1 be the same as T_0, except the $\mathcal{F}_{\mathsf{EDist}}$ execution is replaced with running the simulator $\mathsf{Sim}_{\mathcal{F}_{\mathsf{EDist}}}(\llbracket \mathbf{P}_i \rrbracket, \llbracket \boldsymbol{\phi}_k \rrbracket)$, $\forall i \in [n], k \in [K]$. Because $\mathsf{Sim}_{\mathcal{F}_{\mathsf{EDist}}}$ is guaranteed to produce output indistinguishable from real, T_0 and T_1 are indistinguishable.

Hybrid 2. Let T_2 be the same as T_1, except the $\mathcal{F}^K_{\mathsf{min}}$ execution is replaced with running the simulator $\forall i \in [n]$, $\mathsf{Sim}_{\mathcal{F}^K_{\mathsf{min}}}(\llbracket X_{i,1} \rrbracket, \ldots, \llbracket X_{i,K} \rrbracket)$. $\mathcal{F}^K_{\mathsf{min}}$ takes the secret share value of the Euclidean squared distance between point $\mathbf{P}_i$ and cluster $\boldsymbol{\phi}_k$, which does not reveal any information (e.g. the distance). Moreover, the output of $\mathsf{Sim}_{\mathcal{F}^K_{\mathsf{min}}}$ is indistinguishable from real execution, thus T_2 and T_1 are indistinguishable.

Hybrid 3. Let T_3 be the same as T_2, except the execution step (II.b) is replaced as follows.

- Numerator: one can view the numerator computation as a scalar product of two vectors $\mathbf{M}_k$ and $\mathbf{P}_i$, which is implemented using OT. As long as the OT used is secure, so is this computation.
- Denominator: computing secret sharing of the denominator is indeed a scalar product between $\mathbf{M}_k$ and a vector of one. Thus, the simulation is same as above.
- Division: the properties of the $\mathcal{GC}$ allow to replace the division's outputs with random.

In summary, T_3 and T_2 are indistinguishable.

Hybrid 4. Let T_4 be the same as T_3, except the $\mathcal{GC}$ execution is replaced with running the simulator $\mathsf{Sim}_{\mathcal{GC}}(\llbracket e \rrbracket, \epsilon, \min(e, \epsilon)$. Because pseudorandomness guarantees of the underlying simulator, T_4 and T_3 are indistinguishable.

□

B.1 Comparison with Su *et al.* scheme [75]

We evaluate our prototype on 2D synthetic dataset S1 [26], which was experimented in [75] for differentially privacy setting. We obtained the implementations of Su *et al.* scheme [75] from the authors's website, and evaluate their protocol on our own machine, described in Section 7.1 (a single server with 2x 36-core Intel Xeon 2.30GHz CPU and 256GB of RAM). We note that the implementation [75] is in Python.

We recall that differential privacy requires the output of a data analysis mechanism approximately the same, even if any single entity of the input database is arbitrarily added or removed. Formally, a randomized mechanism $\mathcal{A}$ gives ϵ-differential privacy [23, 24, 75] if for any pair of

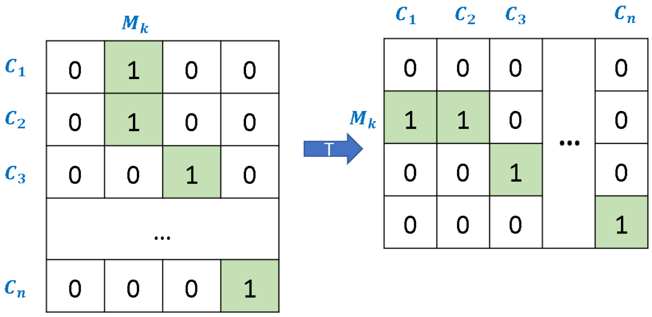
Parameters			#OT ($\times \ell d$)			
n	K	T	Baseline [44]	Our Amortized	Our Adaptive Amortized	Improved Factor
2^{12}	4	10	163840	81960	8232	19.9×
		20	327680	163920	8272	39.6×
	16	10	655360	82080	8352	78.5×
		20	1310720	164160	8512	154×
2^{16}	4	10	2621440	1310760	131112	20×
		20	5242880	2621520	131152	40×
	16	10	10485760	1310880	131232	79.9×
		20	20971520	2621760	131392	159.6×

Table 7: The number of OT instances needed for SESD protocol (described in Figure 3), where n, K is the size of database, T is number of iterations, dimension $d = 2$, and bit-length $\ell = 32$.

neighboring datasets D and D', and any $S \in Range(\mathcal{A}), Pr[\mathcal{A}(D) = S] \leq e^{\epsilon} Pr[\mathcal{A}(D') = S]$. Differentially privacy is used in machine learning (ML) context such that the server has full access to the data in plaintext but wants to guarantee that the released model cannot be used to infer the data used during the training. A common technique used in differentially private ML is to introduce an additive Laplacian noise [23] to the data or the iteration of updating function scaled with the sensitivity. In the experiment of Su *et al.*, we set $\epsilon = 1$ and $T = 30$.

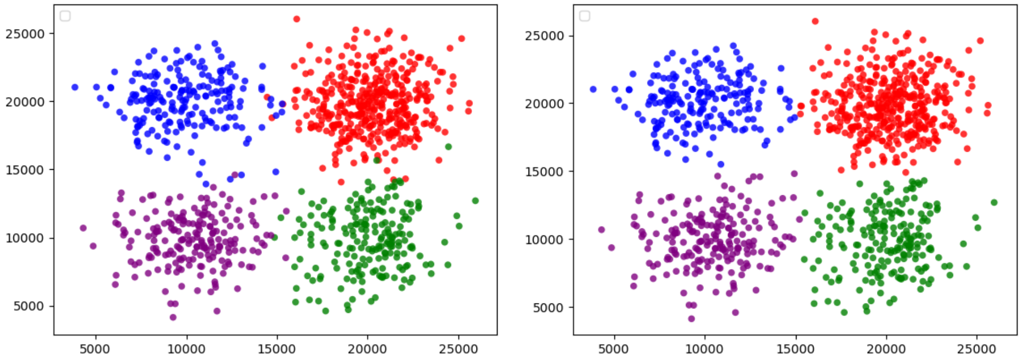
Typically, training a differentially privacy ML model is much faster than training semi-honest ML. We are interested to examine the performance gap between our scheme and this security model. The experimental results in Table 5 show that our privacy-preserving K-means clustering scheme is only 63.5× slower than the differential privacy model [75]. Concretely, our protocol requires 1472.6 seconds to evaluate the model on 2D synthetic dataset S1 while the differential privacy model [75] requires 23.8 seconds.

C Figure Details



$$\begin{array}{c|cccc} & & M_k & & \\ \hline C_1 & 0 & 1 & 0 & 0 \\ C_2 & 0 & 1 & 0 & 0 \\ C_3 & 0 & 0 & 1 & 0 \\ & & \cdots & & \\ C_n & 0 & 0 & 0 & 1 \end{array} \xrightarrow{T} \begin{array}{c|ccccc} & C_1 & C_2 & C_3 & & C_n \\ \hline & 0 & 0 & 0 & & 0 \\ M_k & 1 & 1 & 0 & \cdots & 0 \\ & 0 & 0 & 1 & & 0 \\ & 0 & 0 & 0 & & 1 \end{array}$$

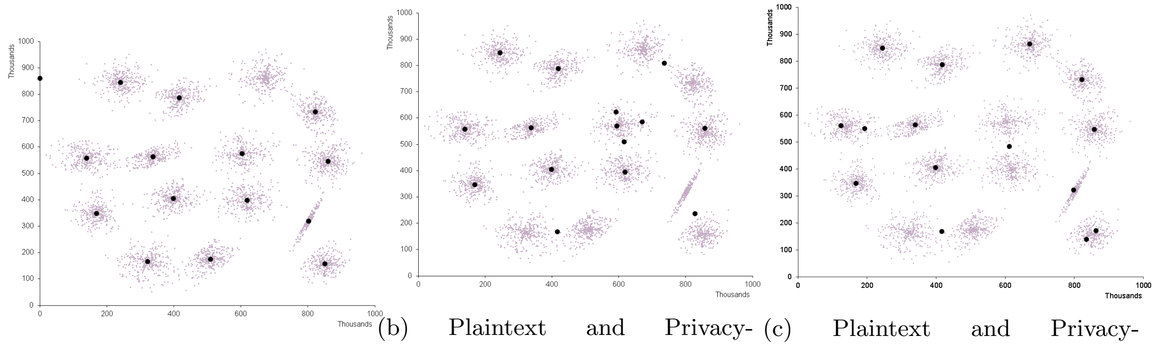
Figure 9: Matrix transposition of a matrix $\mathcal{C}$.



(a) Ground Truth Model [2]

(b) Plaintext and Privacy-Preserving K-means Model.

Figure 10: Comparison of accuracy for privacy-preserving, plain-text, and ground truth model. Our privacy-preserving model achieves the same accuracy as the plain-text model, which reaches 95% accuracy compared to the expected ideal clusters



(a) Ground Truth Model [26]

(b) Plaintext and Privacy-Preserving K-means Model (91.68% accuracy)

(c) Plaintext and Privacy-Preserving K-means Model (86.84% accuracy)

Figure 11: Comparison of accuracy for privacy-preserving, plain-text, and ground truth model. Our privacy-preserving model achieves the same accuracy as the plain-text model, which reaches 86.84% and 91.68% accuracy compared to the expected ideal clusters. The 99% accuracy privacy-preserving model is almost exactly the ground truth model in Figure 11a.