Privacy-Preserving K-means Clustering with Multiple Data Owners

Jung Hee Cheon, Jinhyuck Jeong, Dohyeong Ki, Jiseung Kim, Joohee Lee, and Seok Won Lee

Seoul National University, Seoul 08826, South Korea
{jhcheon,wlsyrlekd,wooki7098,tory154,skfro6360,cvpndrv}@snu.ac.kr

Abstract. Recently with the advent of technology, a lot of data are stored and mined in cloud servers. Since most of the data contain potential private information, it has become necessary to preserve the privacy in data mining. In this paper, we propose a protocol for collaboratively performing the K-means clustering algorithm on the data distributed among multiple data owners, while protecting the sensitive private data. We employ two service providers in our scenario, namely a main service provider and a key manager. Under the assumption that the cryptosystems used in our protocol are secure and that the two service providers do not collude, we provide a perfect secrecy in the sense that the cluster centroids and data are not leaked to any party including the two service providers. Also, we implement the scenario using recently proposed leveled homomorphic encryption called HEAAN. With our construction, the privacy-preserving K-means clustering can be done in less than one minute while maintaining 80-bit security in a situation with 10,000 data, 8 features and 4 clusters.

Keywords: K-means Clustering, Machine Learning, Privacy-Preserving, Fully Homomorphic Encryption, HEAAN

1 Introduction

Recently, the development of information and communication technology has made it possible to integrate a large amount of data that is difficult for an individual to handle. Thus, delegating information process to agencies that can manage and treat large data sets has become a natural phenomenon. Consequently, service providers that professionally process large volumes of data have emerged. Data mining is a major method for data analysis, and one of the fundamental techniques of data mining is clustering.

Clustering analysis is one of the most vigorously studied methods in the unsupervised machine learning field. Among many clustering analysis methods, it is well-known that K-means clustering is easy to implement, apply and interpret with fast and efficient algorithm in terms of a computational cost. In the signal processing field from which K-means clustering originated, this algorithm is used for the task of an image segmentation or a color quantization [1,2]. Furthermore, K-means clustering can be used to find correlations between different

types of cancers [3], myocardial infarction prediction [4], weather prediction [5] and fraud or abuse detection of medical claims [6]. In many cases, the datasets are sensitive, so it is necessary to pay particular attention in privacy. For example, we can assume a situation that patient data are spread among multiple hospitals and they should aggregate all data to predict a specific disease or to obtain meaningful information about diseases. In such case, we have to perform K-means clustering over multiple data owners while preserving privacy of given data.

Our Contribution. In this paper, we design and implement a two-server protocol for privacy preserving K-means clustering using a fully homomorphic encryption scheme, recently proposed HEAAN [7]. By using HEAAN, we can encode the real number data into the message space without adopting any strategies to inject them into the integer space. Previously this can not be done, because most of the fully homomorphic encryption schemes support operations over a fixed integer modulus space only, and it was necessary to encode real data into integers in the bounded ranges. Hence, especially for a large real data set, any encoding strategies would easily get failed to fit in, which causes a blow-up of the parameter sizes. We avoid this hurdle using HEAAN, which results in a fast implementation. We further optimize the implementation using the batching technique for HEAAN to parallelize the computation in the algorithm and to pack a number of data into a single ciphertext, and it allows us to gain considerable efficiency in both time complexity and communication cost. We demonstrate the full process in detail, especially focusing on how to use HEAAN efficiently. To curb unnecessary burdens, we also employed the additive homomorphic encryption, Paillier encryption, when encrypting a weight vector of clusters. Compared to the result of Rao *et al.* [8] which takes 337 minutes per iteration for the same goal when clustering 10000 data with 8 features, our protocol is much faster. It takes less than one minute per iteration for the same data set and security level.

We also gain the perfect secrecy on the data owner's perspective assuming the honest-but-curious (HBC) model. In other words,

- Each data owners can not obtain any information of the data of other data owners.
- Both service providers can not achieve any information of the data assuming that the service providers do not collude with each other.

Therefore, not only the raw data itself, but also any of the cluster centroids and the number of data in each cluster are remained as secret, if the hired cryptosystems are secure and the service providers do not collude with each other.

Related Works There have been many researches on collaboratively performing K-means clustering algorithm in a privacy-preserving way. Earlier studies are reviewed in a proper manner in a survey paper written by Meskine *et al.* [9]. The paper classifies the researches into three groups according to the way data are distributed: researches on vertically partitioned data [10,11], where each data

owner has some portion of features of all data, researches on horizontally partitioned data [12,13], where each data owner has some portion of data with full features, and researches on arbitrarily partitioned data [14,15,16,17], where each data owner has arbitrary portion of data and features. They all share one thing in common: multiple data owners cooperatively perform K-means clustering without revealing information of their own data set to each other.

With the advent of cloud services, recent studies [18,19,20] focus more on an outsourced environment, where service providers are commissioned to collect a lot of data from data owners who have restricted computational resources and compute cluster centroids for them. In this scenario, data are assumed to be horizontally distributed among data owners and the service provider should not achieve any confidential information from the data while performing the algorithm.

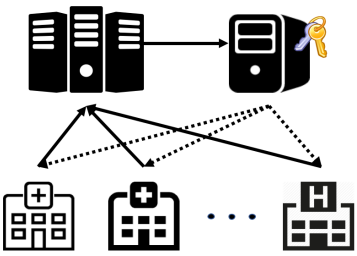
In general, there are two approaches in Honest but Curious (HBC) model that consists of service providers and data owners for this problem: one is having a single service provider [21,22,23], and the other consists of two or more service providers [18,24,8].

The former one is relatively simple, Xing *et al.* [23] recently suggested an algorithm which is secure even if multiple parties collude, but the line of work in this approach has a drawback in common that the cluster centroids and indices of clusters to which each data is assigned are revealed to the service provider.

The latter can be considered if one needs to hide all the information from the service providers. We take this approach with two service providers and intend to be conservative for any leakage of information in our scenario. Rao *et al.* [8] also presented a protocol with two service providers and their goal is the closest to ours. In their algorithms, they eliminated data owners' burden in computing cluster centroids with a satisfactory security level by using Paillier cryptosystem and a set of privacy-preserving primitives. However, it took quite a lot of time to run a single iteration in their algorithm: clustering 10,000 10-dimensional data into 4 clusters took 337 minutes per iteration.

Our Scenario. Our scenario consists of two service providers, called a main service provider and a key manager, and multiple data owners. For the sake of simplicity, we denote the main service provider by MSP, the key manager by KM and i-th data owner by DO_i. The KM generates public keys for the homomorphic encryption schemes and distributes them to MSP and DO_i's. Multiple data owners delegate clustering process to the main service provider, and the main service provider performs computation with encrypted data using homomorphic properties of cryptosystems. More precisely, KM generates public key of Paillier encryption, which is an additive homomorphic encryption and HEAAN, which is a fully homomorphic encryption, and disseminates them to MSP [25], [7]. At the same time, MSP generate secret keys of symmetric key encryption like AES-128 and deliver them to each data owner DO_i. DO_i first encrypt their data employing HEAAN and sends them to a MSP. MSP then calculate encrypted distance between data and clusters using homomorphic properties of HEAAN and sends

Fig. 1: Our Scenario



them back to the DO_i. DO_i classifies decrypted data by clusters using distance, and return the number of data contained in each cluster using a Paillier encryption. With the additive property of Paillier encryption, MSP can obtain how many data included in each cluster. All participant repeat the protocol until the number of data contained in each cluster has not changed.

The scenario is based on an Honest but Curious (HBC) model, where each participant honestly follows instructions that are given. We also assume that the key manager does not collude with the main service provider.

Moreover, all of the communication between MSP and DO_i's should be encrypted with a symmetric key encryption such as AES-128 because KM has secret key of Paillier encryption and HEAAN. Otherwise, KM can observation all secret information of DO_i's.

Organization. In Section 2, we introduce some preliminaries related to homomorphic encryption schemes and K-means clustering. In Section 3, we present our new privacy-preserving K-means clustering algorithm using HEAAN and Paillier cryptosystem with its security analysis. In Section 4, we display our experimental results and compare it with previous works.

2 Preliminaries

2.1 Additive Homomorphic Encryption

Additive homomorphic encryption (AHE) is a cryptosystem which supports the additive operation of plaintext without decryption process. Among various AHEs, we adopt the Paillier cryptosystem proposed first by [**?**]. Here is a brief description for algorithms of Paillier system.

- $(pk_{\mathsf{P}}, sk_{\mathsf{P}}) \leftarrow \mathsf{Paillier.KeyGen}_{\mathsf{P}}(1^{\lambda})$: Set $n = pq$ and $\ell = \mathrm{lcm}(p-1, q-1)$ with two large primes p and q having the same bitsize. Choose en element $g \in \mathbb{Z}_{n^2}^{\times}$ of order divided by n and compute

$$\mu = \left(\left\lfloor \frac{(g^{\lambda} \bmod n^2) - 1}{n} \right\rfloor \right)^{-1} \pmod{n}.$$

Output $pk_{\mathsf{P}} = (n, g)$ and $sk_{\mathsf{P}} = (\ell, \mu)$.

- $c \leftarrow \mathsf{Paillier.Enc}_{pk_{\mathsf{P}}}(m)$: For a message $0 \leq m < n$ and a public key $pk_{\mathsf{P}} = (n, g)$, choose a random $r \in \mathbb{Z}_n$ and compute a ciphertext $c = g^m \cdot r^n \bmod n^2$. Output c.

- $m \leftarrow \mathsf{Paillier.Dec}_{sk_{\mathsf{P}}}(c)$: For a ciphertext c and a secret key $sk_{pa} = (\ell, \mu)$, compute
$$m = \left\lfloor \frac{(c^\lambda \bmod n^2) - 1}{n} \right\rfloor \cdot \mu \pmod{n^2}$$
and output m.

2.2 Homomorphic Encryption for Arithmetic of Approximate Numbers

Homomorphic Encryption (HE) is a cryptographic scheme which aims to enable homomorphic operations such as additions and multiplications on encrypted data. After Gentry's blueprint [26,27], there have been many following works [28,29,30,31,32,33,34,35,36,37,38,39,40,41,42,43,44,45], and many applications of HE also emerged for various usage in medical, genomic, or financial fields of studies [46,47,48,49,50,51]. Though remarkable improvements in the area of HE arose and the HE libraries such as HElib [37,52,53] or YASHE [41] show some good implementation results in their applications [47,51], they only support operations over a fixed integer modulus space so that there are some difficulties to adapt the real data sets in most cases. In fact, encoding strategy for the real data affects the real time of the implementations and especially for the huge data set with much entropy, any hired encoding strategy can cause blow-up for the parameters and it might be very costly to implement.

Recently, a homomorphic encryption scheme which focus on arithmetic of approximate numbers, namely HEAAN [7], appeared to mitigate this problem. Indeed, they suggest a kind of converting technique to turn a HE scheme with certain hardness assumptions into a HE scheme that carries out approximate computations efficiently. The main idea of HEAAN is to treat a noise from the hardness assumption of a scheme as part of error occurred in approximate computations.

We describe the algorithms in the (leveled) HE scheme HEAAN of depth L here. Let $\Phi_{M'}(x)$ be an M'-th cyclotomic polynomial of degree $\phi(M')$: we will set $\phi(M')$ to be a power-of-two and $\Phi_{M'}(x) = x^M + 1$ where $\phi(M') = M$. We also set the parameters M', an integer h, an integer P, and a real σ to make the hard problem $\mathsf{RLWE}_{P \cdot q_L, \sigma}$ with certain secret distribution (the distribution of s) achieve λ-bit security for targeted λ. Let $\mathcal{R} = \mathbb{Z}[x]/(\Phi_{M'}(X))$ and $\mathcal{R}_q := \mathcal{R}/q\mathcal{R}$. First, we need to set the sequence of moduli $\{q_i\}_{i=0}^{L}$ for the ciphetext space in each depth by letting $p, q_0 > 0$ be fixed integers and $q_i = p^i \cdot q_0$ for $0 < i \leq L$. Then, the ciphertext space of level ℓ is $\mathcal{R}_{q_\ell}^k$ for fixed k.

One major difference in HEAAN compared to other HE schemes is that a plaintext can be an arbitrary complex vector, and it is encoded into the space

$\mathcal{R}$. Let S be a subgroup of $\mathbb{Z}_{M'}^*$ satisfying $\mathbb{Z}_{M'}^*/S = \{\pm 1\}$, and $\mathbb{H} = \{z = (z_j) \in \mathbb{Z}_{M'}^* : z_j = \overline{z_{-j}}, \forall j \in \mathbb{Z}_{M'}^*\}$. Let $\sigma : \mathbb{Q}[X]/\Phi_{M'}(X)) \to \mathcal{C}^N$ be the canonical embedding map. Then, the specifications of their algorithms are as follows.

- $\mathsf{KeyGen}(1^\lambda) \to (pk, sk, evk)$:
  - Sample random $a \leftarrow \mathcal{R}_{q_L}$, sparse signed binary polynomial s in $\mathcal{R}$ of h non-zero coefficients, and $e \in \mathcal{R}_{P \cdot q_L}$ of small sizes of which sizes depend on σ^2. Calculate $b \leftarrow -as + e \bmod q_L$, and set the secret key $sk \leftarrow (1, s)$ and the public key $pk \leftarrow (b, a) \in \mathcal{R}_{q_L}^2$.
  - Sample random $a' \leftarrow R_{P \cdot q_L}$ and $e' \in R_{P \cdot q_L}$ of small sizes of which sizes depend on σ^2. Calculate $b' \leftarrow -a's + e' + Ps' \bmod P \cdot q_L$, where $s' \leftarrow s^2$. Set the evaluation key $evk \leftarrow (b', a') \in \mathcal{R}_{P \cdot q_L}^2$.

- $\mathsf{Encode}(z) \to m$: For an $(N/2)$-dimensional vector $z = (z_j)_{j \in S} \in \mathbb{Z}[i]^{N/2}$, first transform z into $\mathbb{H}$ using the map ι defined by

$$\iota(z)[j] = \begin{cases} z_j & \text{if } j \in S \\ \overline{z_{-j}} & \text{if } j \notin S \end{cases},$$

  and compute $\mathbf{m} = \lfloor \Delta \cdot \iota(z) \rceil_{\sigma(\mathcal{R})}$ in $\mathbb{H}$ for the scale factor Δ. Return the polynomial $m(X) = \sigma^{-1}(\mathbf{m}) \in \mathcal{R}$.

- $\mathsf{Decode}(m) \to z$: Let $\mathbf{m} = \sigma(m(X)) \in \mathbb{H}$ and output the corresponding vector $z = \Delta^{-1} \cdot \iota^{-1}(\mathbf{m}) \in \mathcal{C}^{N/2}$, that is, $z_i = \Delta^{-1} \cdot m(\delta_M^i)$ for $i \in S$.

- $\mathsf{Encrypt}(pk, m) \to \mathbf{c}$: Sample $\mathbf{r} \in \mathcal{R}^2$ for which coefficients of each component would be zero with probability $1/2$, and ± 1 with probability $1/4$, respectively, and $e_0, e_1 \leftarrow \mathcal{DG}_{q_L}(\sigma^2)$. Output $\mathbf{c} \leftarrow \mathbf{r} \cdot pk + (m + e_0,\ e_1) \in \mathcal{R}_{q_L}^2$.

- $\mathsf{Decrypt}(sk, \mathbf{c} = (c_0, c_1)) \to m$: Output $m \leftarrow c_0 + c_1 \cdot s \bmod q_\ell$.

- $\mathsf{Add}(\mathbf{c}_1, \mathbf{c}_2) \to c_{\mathsf{add}}$: Output $\mathbf{c}_{\mathsf{add}} \leftarrow \mathbf{c}_1 + \mathbf{c}_2 \bmod q_\ell$.

- $\mathsf{Mult}(\mathbf{c}_1, \mathbf{c}_2) \to \mathbf{c}_{\mathsf{mult}}$: Set $c_1 = (b_1, a_1)$ and $c_2 = (b_2, a_2)$. Let $(d_0, d_1, d_2) \leftarrow (b_1 b_2, a_1 b_2 + a_2 b_1, a_1 a_2) \in \mathcal{R}_{q_\ell}^3$. Output $\mathbf{c}_{\mathsf{mult}} \leftarrow (d_0, d_1) + \lfloor \frac{1}{P}(d_2 \cdot evk \bmod P q_\ell) \rceil \in \mathcal{R}_{q_\ell}^2$.

- $\mathsf{Rescaling}_{\ell \to \ell'}(\mathbf{c}) \to \mathbf{c}'$: For a level ℓ ciphertext $\mathbf{c}$, output $\mathbf{c}' \leftarrow \lfloor \frac{q_{\ell'}}{q_\ell} \mathbf{c} \rceil \in \mathcal{R}_{q_{\ell'}}^2$ at level ℓ'.

For more details, we recommend to see [7]. Additionally, we make some notes on functions in the open library of HEAAN [54] of the version reported on Aug 7, 2017. We will use the following functions in the library for our implementation.

- modEmbedOne : We can convert a ciphertext into a ciphertext in the next level with this function, e.g. we make $[x_{i,t}]_2$ from $[x_{i,t}]_1$ with this function.

– modSwitchOne : This function is used for increasing FHE level by 1 after multiplying ciphertexts.
– leftRotateByPo2 (= $shift$ Operator) : This function used for shifting ciphertext slots by 1 to the left.

2.3 K-means Clustering

Given N data $\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_N \in \mathbb{R}^m$ and the number of clusters K, K-means clustering algorithm clusters data into K clusters based on an Euclidean distance. It is performed by assigning each data to its nearest cluster and computing centroids of clusters repeatedly.

Initializing cluster centroids is done by randomly choosing K data from $\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_N$. After initialize the centroids $\mu_1, \cdots, \mu_K$, an iteration which consists of two steps starts:

(i) Compute $c_i := \arg\min_j \|\mathbf{x}_i - \mu_j\|$ for each $\mathbf{x}_i$.
(ii) Update cluster centroids by setting

$$\mu_j = (\sum_{i \in A_j} x_i)/|A_j|,$$

where $A_j = \{i | c_i = j\}$.

There are several termination criterion that are widely used. One of them which we will use is to check each data's assignment between (i) and (ii) of every iteration. If a newly computed c_i is the same as the former c_i for all $i = 1, 2, \cdots, N$, then the iteration is terminated.

It can be proved that $J = \sum_{i=1}^{N} \|\mathbf{x}_i - \mu_{c_i}\|^2$, a *cost function* that we want to minimize, decreases in every iteration. While J attains the function's local minimum when the algorithm finishes, it cannot be assured that the value is global minimum. Thus it is recommended to repeat this algorithm multiple times and choose the clustering result with the minimum value of the cost function.[55]

2.4 Notation

The followings are notations that will be used for the rest of this paper:

3 Privacy-Preserving K-means Clustering with Multiple Data Owners

In this section, we present an algorithm that clusters data which are horizontally distributed among multiple data owners. More precisely, our scenario consists of a main service provider MSP, who is in charge of heavy computations, a key manager KM, who is in charge of key distributions and decryptions of HEAAN

N	the number of total data
N_i	the number of data of DO_i
m	the dimension of each data
n	the number of data owners
K	the number of clusters
M	the parameter in HEAAN from $\Phi_{M'}(x) = x^M + 1$
$\mathbf{x}_{i,t}$	t-th data of DO_i
$\mu_i^{(j)}$	i-th cluster centroid in j-th iteration
k_i^{MSP}	a symmetric key between MSP and DO_i
k_i^{KM}	a symmetric key between KM and DO_i
$(pk_{\mathsf{H}}, sk_{\mathsf{H}})$	a key pair of HEAAN
$(pk_{\mathsf{P}}, sk_{\mathsf{P}})$	a key pair of Paillier
$\mathsf{Enc}_{\mathsf{MSP},i}(x)$	x encrypted with a symmetric key k_i^{MSP}
$\mathsf{Enc}_{\mathsf{KM},i}(x)$	x encrypted with a symmetric key k_i^{KM}
$\mathsf{H.Enc}(x)$	x encrypted with HEAAN
$\mathsf{P.Enc}(x)$	x encrypted with Paillier

and Paillier cryptosystem, and multiple data owners DO_i who owns N_i data with full m feature for $i = 1, \ldots, n$. (Let $N = N_1 + \cdots + N_n$.)

Our algorithm consists of 3 steps;

- In the first step (Setup), symmetric keys are distributed to create secure channels via a Public key infrastructure, and HEAAN and Paillier cryptosystem are initialized.
- In the second step (Initialization for Iterations), each DO_i sends encrypted data to MSP and MSP initializes cluster centroids.
- In the third step (Iteration), an iteration is started. MSP computes the encrypted distances between each data and centroids and returns the distances to each DO_i via KM. Then each DO_i compares the distances and construct weight vectors, which will be explained later, using additive homomorphic encryption scheme; Paillier. Finally, MSP updates new encrypted cluster centroids with weight vectors received from DO_i. Additionally, a termination criterion is checked inside Step 3 to decide whether or not to terminate the iteration phase.[1]

[1] Of course, fully homomorphic encryption makes it possible to operate this third step without data owner's help. However, each iteration requires one computation for

3.1 Description

Here we give a detailed description for each of three step.

Step 1. Setup

First of all, each DO_i exchanges a symmetric key k_i^{MSP} and k_i^{KM} with MSP and KM, respectively. Then KM generates public keys and secret keys of HEAAN and Paillier cryptosystems. We denote them by $(pk_{\mathsf{H}}, sk_{\mathsf{H}})$ and $(pk_{\mathsf{P}}, sk_{\mathsf{P}})$ respectively. After generating keys, KM disseminates two public keys to MSP and all DO_i.

Step 2. Initialization for Iterations

(i) DO Side. Through this process, each DO_i will encrypt their data $\{\mathbf{x}_{i,t}\}_{t=1}^{N_i}$ with HEAAN at level 1 using batch processing and send it to MSP after encrypting again with k_i^{MSP}. Considering the trivial embedding of $\mathbb{R}^m$ into $\mathbb{R}^{\widetilde{m}}$, where $\widetilde{m} = 2^{\lceil \log_2 m \rceil}$, we consider $\mathbf{x}_{i,t}$ as an $\widetilde{m}$-dimensional vector. The data owner DO_i encrypts as [2]

$$[\mathbf{x}_{i,t}]_1 = \mathsf{H.Enc}(\mathbf{x}_{i,t} \otimes \bar{1}) \qquad \forall\ 1 \le i \le N,\ 1 \le t \le N_i,$$

where $\bar{1} = (1, \ldots, 1) \in \mathbb{R}^{\widetilde{K}}$ with $\widetilde{K} = 2^{\lceil \log_2 K \rceil}$, and send $\mathsf{Enc}_{\mathsf{MSP},i}([\mathbf{x}_{i,t}]_1)$ to MSP.

For the initialization of cluster centroids, each DO_i also sends $s = \lceil K/n \rceil$ data which are randomly chosen from its data set to MSP. We temporarily denote them by $[\mathbf{y}_{i,t}]_1$ where $1 \le i \le n, 1 \le t \le s$. The encryption and communication process is the same as the above.

(ii) SP Side. MSP decrypts all data received and obtains $[\mathbf{x}_{i,t}]_1$ for all $1 \le i \le n$, $1 \le t \le N_i$, and $[\mathbf{y}_{i,t}]_1$ for all $1 \le i \le n, 1 \le t \le s$.

MSP firstly computes

$$[\delta_i]_1 = \mathsf{H.Enc}(\bar{1} \otimes e_i),$$

for $1 \le i \le K$ where $\bar{1} = (1, \ldots, 1) \in \mathbb{R}^{\widetilde{m}}$ and $\{e_i\}_{i=0}^{K}$ is the first K elements of the standard basis of $\mathbb{R}^{\widetilde{K}}$. Note that $\bar{1} \otimes e_i$ is a vector containing ones from $(\widetilde{m}(i-1)+1)$-th component to $\widetilde{m}i$-th component and zeros in others.

---

maximum argument per each data and this computation has homomorphic comparisons as a subroutine. Note that homomorphic comparison circuit consumes asymptotically $\log(d)$ levels of encryption where d is the bit size of input[56]. It means that it is impractical to operate the third step using a comparison circuit without decryption.

[2] For the convenience, we describe this algorithm with the assumption that one ciphertext contains one data. We can optimize this if we use one ciphertext for multiple data which will be explained in the end of this section.

Among $[\mathbf{y}_{i,t}]_1$ received, MSP randomly chooses K ciphertexts, namely $[\mathbf{z}_1]_1$, $\cdots, [\mathbf{z}_K]_1$. Now MSP computes the following:

1. $[\mu_i^{'}]_1 = [\mathbf{z}_i]_1 \times [\delta_i]_1, \qquad \forall\ 1 \leq i \leq K,$
2. $[\mu_i^{'}]_2 \xleftarrow{\text{Increase 1-level}} [\mu_i^{'}]_1,$
3. $[\mu^{(0)}]_2 = \sum_{i=1}^{K} [\mu_i^{'}]_2.$

In this way, MSP only obtains encrypted data and encrypted initial cluster centroids. Also, we can guarantee that centroids are well-spread among diverse data owners by the K random choices. In addition, MSP obtains $[\mathbf{x}_{i,t}]_2$ by increasing the level of $[\mathbf{x}_{i,t}]_1$ and stores it with $[\mathbf{x}_{i,t}]_1$ for further computations.

Step 3. Iteration

Before we start, this step can be summarized as follows:

(i) MSP computes the encrypted distance from each data to centroids, and adds a random error to it. MSP sends the ciphertext to KM. KM decrypts the ciphertext, encrypts it with k_i^{KM} and sends it to DO_i. MSP also encrypts the error with k_i^{MSP} and sends it to DO_i.

(ii) DO_i can get distances between its data and centroids in plaintext. DO_i computes the number of its data that are assigned to each cluster. DO_i then encrypts it with pk_{P} and k_i^{MSP}, and sends it to MSP.

(iii) MSP sums those data and gets the number of total data that are assigned to each cluster, along with another random error. MSP sends the ciphertext to KM. KM decrypts the ciphertext, encrypts it with k_i^{KM} and sends it to DO_i. MSP also encrypts the error with k_i^{MSP} and sends it to DO_i.

(iv) DO_i computes a weight vector, which will soon be explained, of each data and encrypts it with pk_{H} and k_i^{MSP}, and sends it to MSP.

(v) MSP performs multiplications between an encrypted data and an encrypted weight vector of the same index and sums them all up, thereby obtaining new encrypted centroids.

Throughout this step, we assume that we are in the j-th iteration.

(i) SP Side 1. In the first sub-step, MSP performs the following computations for each $i = 1, \cdots, N$ and $t = 1, \cdots, N_i$:

1. $\left[\left(\mathbf{x}_{i,t} - \mu^{(j)}\right)\right]_2 = [\mathbf{x}_{i,t}]_2 - [\mu^{(j)}]_2$
2. $\left[\left(\mathbf{x}_{i,t} - \mu^{(j)}\right)^2\right]_2 = \left(\left[\mathbf{x}_{i,t} - \mu^{(j)}\right]_2\right)^2$
3. $\left[\left(\mathbf{x}_{i,t} - \mu^{(j)}\right)^2\right]_3 \xleftarrow{\text{Increase 1-level}} \left[\left(\mathbf{x}_{i,t} - \mu^{(j)}\right)^2\right]_2$
4. $\left[\mathbf{D}_{i,t}^{(j)}\right]_3 = \sum_{s=1}^{m} (shift)^s \left(\left[\left(\mathbf{x}_{i,t} - \mu^{(j)}\right)^2\right]_3\right)$

As a result, one can easily check $((l-1)\widetilde{m}+1)$-th slot of $[\mathbf{D}_{i,t}^{(j)}]_3$ has an encrypted data of a squared Euclidean distance between $\mathbf{x}_{i,t}$ and $\mu_l^{(j)}$ for $l = 1, \ldots, K$. Note that only 1-st, $(\widetilde{m}+1)$-th, $\cdots$, and $((K-1)\widetilde{m}+1)$-th slots are meaningful in $[\mathbf{D}_{i,t}^{(j)}]_3$.

MSP generates a random error $\mathbf{r}_{i,t}$ for each $[\mathbf{D}_{i,t}^{(j)}]_3$ and makes $[\mathbf{r}_{i,t}]_3$. MSP then sends $[\mathbf{D}_{i,t}^{(j)} + \mathbf{r}_{i,t}]_3$ to KM and sends $\mathsf{Enc}_{\mathsf{MSP},i}(\mathbf{r}_{i,t})$ to DO_i. After KM sending $\mathsf{Enc}_{\mathsf{KM},i}(\mathbf{D}_{i,t}^{(j)} + \mathbf{r}_{i,t})$ to DO_i, DO_i can obtain $\mathbf{D}_{i,t}^{(j)}$.

(ii) DO Side 1. Now DO_i can obtain distances between the t-th data and cluster centroids and can decide to which cluster the t-th data should be assigned. We denote the index of the cluster to which DO_i's t-th data is assigned in j-th iteration by $c_{i,t}^{(j)}$. After DO_i figuring out all N_i data's assignments, DO_i creates a vector

$$\mathbf{v}_i^{(j)} = (a_{i,1}^{(j)}, \cdots, a_{i,K}^{(j)})$$

where $a_{i,t}^{(j)}$ is the number of data which are assigned to the t-th cluster.

Before continuing, a termination criterion is checked here. Each DO_i informs MSP about whether or not $c_{i,t}^{(j)}$ is the same as $c_{i,t}^{(j-1)}$ for all $1 \le t \le N_i$. Upon receiving those information, MSP announces the termination of iterations if all DO_i said that their $c_{i,t}^{(j)}$ and $c_{i,t}^{(j-1)}$ are the same. If not, MSP announces that the iteration should go on.

To get the value of the *cost function*, the i-th data owner also computes

$$J_i^{(j)} = \sum_{t=1}^{N_i} \|\mathbf{x}_{i,t} - \mu_{c_{i,t}^{(j)}}^{(j)}\|^2,$$

a sum of squared Euclidean distances between DO_i's data and their closest cluster centroid. Although DO_i does not know cluster centroids, $J_i^{(j)}$ can be computed by finding and adding values from corresponding coordinates of $\mathbf{D}_{i,t}^{(j)}$ where $t = 1, \ldots, N_i$. Then DO_i encrypts each component of $\mathbf{v}_i^{(j)}$ and $J_i^{(j)}$ with pk_{P} and k_i^{MSP} and sends $\mathsf{Enc}_{\mathsf{MSP},i}(\mathsf{P.Enc}(a_{i,t}^{(j)}))$, $\mathsf{Enc}_{\mathsf{MSP},i}(\mathsf{P.Enc}(J_i^{(j)}))$ to MSP.

(iii) SP Side 2. MSP decrypts all data that each DO_i sent in **(ii)**, thereby obtaining $\mathsf{P.Enc}(a_{i,t}^{(j)})$ and $\mathsf{P.Enc}(J_i^{(j)})$ for all i and t. Using these encrypted data and the additive homomorphic cryptosystem, MSP computes

$$1. \quad \mathsf{P.Enc}(a_t^{(j)}) = \sum_{i=1}^{n} \mathsf{P.Enc}(a_{i,t}^{(j)}), \qquad 1 \le t \le K,$$

$$2. \quad \mathsf{P.Enc}(J^{(j)}) = \sum_{i=1}^{n} \mathsf{P.Enc}(J_i^{(j)}).$$

MSP also generates random errors $s_{1,t}$ $(1 \le t \le K)$ and s_2, and computes $\mathsf{P.Enc}(a_t^{(j)} + s_{1,t})$ and $\mathsf{P.Enc}(J^{(j)} + s_2)$. Similar to **(i)**, KM decrypts them for DO_i and MSP sends $\mathsf{Enc}_{\mathsf{MSP},i}(s_{1,i})$ and $\mathsf{Enc}_{\mathsf{MSP},i}(s_2)$ to DO_i.

(iv) DO Side 2. Upon receiving data from MSP and KM, each DO_i obtains $\mathbf{v} = (a_1, \cdots, a_K)$. It means that among total data, each DO_i knows how many data are assigned to each cluster. As DO_i knows the index of the cluster to which its each data is assigned, DO_i can now create a weight vector $\mathbf{w}_{i,t}^{(j)}$ for all $1 \le t \le N_i$ and encrypted with pk_{H} at level 1 with batch processing as follows:

$$[\mathbf{w}_{i,t}^{(j)}]_1 = \mathsf{H.Enc}\left(\frac{1}{a_{c_{i,t}^{(j)}}} \cdot \bar{1} \otimes e_{c_{i,t}^{(j)}}\right),$$

where $\bar{1} = (1, \ldots, 1) \in \mathbb{R}^{\widetilde{m}}$. Each then sends $\mathsf{Enc}_{\mathsf{MSP},i}([\mathbf{w}_{i,t}^{(j)}]_1)$ to MSP.

(v) SP Side 3. MSP obtains $[\mathbf{w}_{i,t}^{(j)}]_1$ after decryption. To update encrypted cluster centroids, MSP computes

$$1. \quad [\mu_{i,t}^{''}]_1 = [\mathbf{x}_{i,t}]_1 \times [\mathbf{w}_{i,t}^{(j)}]_1,$$
$$2. \quad [\mu_{i,t}^{''}]_2 \xleftarrow{\text{Increase 1-level}} [\mu_{i,t}^{''}]_1,$$
$$3. \quad [\mu^{(j+1)}]_2 = \sum_{i,t} [\mu_{i,t}^{''}]_2.$$

Here $[\mu^{(j+1)}]_2$ contains data of updated cluster centroids encrypted with HEAAN at level 2. Then MSP goes back to **(i)** of Step 3 and enter $(j+1)$-th iteration with these updated centroids.

After Step 3 is finished by meeting the termination criterion[3], each DO_i obtains $c_{i,t}$, an index of the cluster to which t-th data is assigned. DO_i also obtains J, the value of the *cost function*, and can decide whether to run the whole algorithm again or not.

Finally, Step 3 is summarized in Algorithm 1.

[3] It can be easily proved that the algorithm terminates within finite iterations.

Algorithm 1 The Iteration to Update Cluster Centroids

Input: $\{[\mathbf{x}_{i,t}]_1, [\mathbf{x}_{i,t}]_2\}$ $\forall\ i = 1, \cdots, n,\ \ t = 1, \cdots, N_i$ and $[\mu^{(0)}]_2$ of MSP
Output: $c_{i,t}$ $\forall\ t = 1, \cdots, N_i$ to DO_i for $i = 1, \cdots, n$

for $i = 1$ **upto** n **do**
  for $t = 1$ **upto** N_i **do**
    MSP generates $\mathbf{r}_{i,t}$ and computes $[\mathbf{D}_{i,t}^{(j)}]_3$, $[\mathbf{r}_{i,t}]_3$.
    MSP sends $[\mathbf{D}_{i,t}^{(j)} + \mathbf{r}_{i,t}]_3$ to KM and sends $\mathsf{Enc}_{\mathsf{MSP},i}(\mathbf{r}_{i,t})$ to DO_i.
    KM sends $\mathsf{Enc}_{\mathsf{KM},i}(\mathbf{D}_{i,t}^{(j)} + \mathbf{r}_{i,t})$ to DO_i.
    DO_i computes $c_{i,t}^{(j)}$.
  end for
  DO_i computes $\mathbf{v}_i^{(j)}$.
end for
if $c_{i,t}^{(j)} = c_{i,t}^{(j-1)}$ $\forall\ i = 1, \cdots, n,\ \ t = 1, \cdots, N_i$ **then**
  $c_{i,t} = c_{i,t}^{(j)}$ $\forall\ i = 1, \cdots, n,\ \ t = 1, \cdots, N_i$
  goto return.
end if
for $i = 1$ **upto** n **do**
  DO_i computes $J_i^{(j)}$, and sends $\mathsf{Enc}_{\mathsf{MSP},i}(\mathsf{P.Enc}(J_i^{(j)}))$ to MSP.
  for $t = 1$ **upto** N_i **do**
    DO_i sends $\mathsf{Enc}_{\mathsf{MSP},i}(\mathsf{P.Enc}(a_{i,t}^{(j)}))$ to MSP.
  end for
end for
MSP computes $\mathsf{P.Enc}(a_t^{(j)})$ and $\mathsf{P.Enc}(J^{(j)})$.
for $i = 1$ **upto** n **do**
  MSP generates $s_{1,t}$ and s_2, and sends $\mathsf{Enc}_{\mathsf{MSP},i}(s_{1,i})$ and $\mathsf{Enc}_{\mathsf{MSP},i}(s_2)$ to DO_i $\forall\ t = 1, \cdots, K$.
  MSP sends $\mathsf{P.Enc}(a_t^{(j)} + s_{1,t})$ and $\mathsf{P.Enc}(J^{(j)} + s_2)$ to KM $\forall\ t = 1, \cdots, K$.
  KM sends $\mathsf{Enc}_{\mathsf{KM},i}(a_t^{(j)} + s_{1,t})$ and $\mathsf{Enc}_{\mathsf{KM},i}(J^{(j)} + s_2)$ to DO_i $\forall\ t = 1, \cdots, K$.
end for
for $i = 1$ **upto** n **do**
  for $t = 1$ **upto** N_i **do**
    DO_i computes $[\mathbf{w}_{i,t}^{(j)}]_1$ and sends it to MSP.
  end for
end for
MSP computes $[\mu^{(j+1)}]_2$ and **goto** 1.
return $c_{i,t}$ $\forall\ t = 1, \cdots, N_i$ to DO_i for $i = 1, \cdots, n$.

Optimization. Although we stated our algorithm under the setting that one ciphertext contains only one data, we can optimize this by packing multiple data in one ciphertext with batching technique. More precisely, as aforementioned, each m-dimensional data should be extended into an $\widetilde{m}$-dimensional data and be repeated $\widetilde{K}$ times to be encrypted. Therefore, we need $\widetilde{m}\widetilde{K}$ slots to encrypt one data. Furthermore, because one slot of ciphertext in HEAAN consists of not only a real part but an imaginary part, we can utilize this that packing two coordinates of a data vector in one slot is possible. The number of slots per data now becomes $\frac{\widetilde{m}\widetilde{K}}{2}$ and it is far smaller than $\frac{M}{2}$, the number of slots available per ciphertext. Thus it is possible to pack $M/\widetilde{m}\widetilde{K}$ data in one ciphertext. Since time elapsed for encryption and decryption scarcely change when we increase the number of slots in one ciphertext, we can increase efficiency by packing multiple data in one ciphertext. In section refexperi we will show that this optimization drastically decreases time consumption of the algorithm with detailed experimental results.

3.2 Security Analysis

In this paper, we assume that all the participants are honest but curious because if some malicious participant manipulates data, everyone cannot obtain correct results. Moreover, the scenario for running the K-means clustering algorithm while preserving the private information is based on the security of the three cryptographic systems such as the Fully Homomorphic Encryption (HEAAN), Additive Homomorphic Encryption (Paillier), and symmetric encryptions involved. In other words, if the three systems are secure against existing attacks, this scenario is sufficiently secure.

MSP only achieves the encrypted data of DOs with HEAAN. However, since MSP does not have the secret key of the HEAAN due to the assumption that MSP does not collude with KM, MSP cannot know the secret information of the DO only with accessing to ciphertexts from the semantic security of the HEAAN. Similarly, since each DO_i encrypts the data using the HEAAN and symmetric encryption when sending data to the MSP, so even if KM see the communication between MSP and DO_i, it is impossible that KM obtains additional private information. Although the attacker colludes with the DO's, additional information can not be obtained except for the secret information of the data owners who conspired since the shared symmetric key between each of the colluded DO and MSP is different.

Also, in Step 3, when the MSP encrypts the distance $[\mathbf{D}_{i,t}]_3$ by injecting random error and sends it to KM.Therefore, even if KM decrypts the ciphertext, it cannot get the original message due to the error.

4 Experimental Results

In this section, we analyze running times of our algorithm. We first state our implementation environment and then explain experimental results of the algorithm.

A. Implementation Environment

- The experiments were conducted on a single computer with Intel(R) CORE$^{\text{TM}}$ i7-7500 at 2.7GHz and 8GB of memory.
- We checked the running time of computations with various parameter settings of data set, *i.e.* N, m or K.
- We ignore the time consumption of encryption and decryption with symmetric keys since it is too small.
- We used KEGG Metabolic Reaction Network (Undirected) Data Set [57] for our K-means clustering. The original data set contains 65554 data with 29 attributes. We chose m attributes from the front and randomly selected N data from the data set for each m and N.

B. Performances

Firstly, we set $N = 10,000$, $m = 8$ and $K = 4$ as our standard parameters and analyzed the running time of each part in Step 2 and 3, respectively. The timing results for each part are presented in Table 1, 2. As targeting 80-bit security, we set the parameters for HEAAN cryptosystem as $M = 13$, $\log_2 q = 154$ and $\log_2 p = 48$, thereby attaining correctness for the outputs of the whole process except negligible probabilities, respectively.

Table 1: Timing Results for Each Part in Step 2 with 80-bit security ($N = 10,000$, $m = 8$, $K = 4$)

Data Encryption	Centroid Initialization		Total
DOs	DOs	MSP	
5196 ms	212 ms	458 ms	5866 ms
88.6 %	3.6 %	7.8 %	100 %

Table 2: Timing Results for Each Part in Step 3 with 80-bit security ($N = 10,000$, $m = 8$, $K = 4$)

SP Side 1		DO Side 1	SP Side 2		DO Side 2	SP Side 3	Total
MSP	KM	DOs	MSP	KM	DOs	MSP	
11433 ms	4381 ms	8 ms	8 ms	6 ms	4546 ms	11687 ms	32069 ms
35.7 %	13.7 %	≈ 0 %	≈ 0 %	≈ 0 %	14.2 %	36.4 %	100 %

In Step 2, which is a preparation phase, most of the time is spent on encrypting data with HEAAN. But it only needs to be done once so the time

consumption is amortizable. In Step 3, the first thing to notice is that the time consumption of Paillier cryptosystem is negligible. We can also see that SP Side 1 and SP Side 3 take up most of the time. These two steps take up 86% of the total time per iteration while MSP taking up 72% alone. DO_i is in charge of only 14% of the total time.

If the number of data owners increases while N remains the same, each data owner's time consumption of Data Encryption in Step 2 and DO Side 2 in Step 3 decreases while time consumption of Paillier cryptosystem hardly increases. Thus, the running time decreases as the number of data owners increases.

Next, we focus on 4 parts which take the most of time consumption. We present the analysis of how each parameter affects on the time consumption.

- MSP of SP Side 1:
  - The most time consuming part is computing distances.
  - The number of homomorphic subtraction, multiplication and conjugation is linear in N, $\widetilde{m}$ and $\widetilde{K}$.
  - The number of *shift* operations and homomorphic additions exactly equals to m.
  - Thus the time consumption is linear in N, $\widetilde{K}$, and asymptotically linear in m.

- KM of SP Side 1:
  - The most time consuming part is decrypting distances.
  - The number of ciphertexts of distances to decrypt is linear in N, $\widetilde{m}$ and $\widetilde{K}$
  - Thus the time consumption is linear in N, $\widetilde{m}$ and $\widetilde{K}$.

- DO_i of DO Side 2:
  - The most time consuming part is encrypting weight vectors.
  - The number of weight vectors to encrypt is linear in N, $\widetilde{m}$ and $\widetilde{K}$.
  - Thus time consumption is linear in N, $\widetilde{m}$ and $\widetilde{K}$.

- MSP of SP Side 3:
  - The most time consuming part is computing new centroids.
  - The number of homomorphic additions and multiplications is linear in N, $\widetilde{m}$ and $\widetilde{K}$.
  - Due to the optimization, we need additional *shift* operators and homomorphic additions as many as the number of data packed in one ciphertext, which is inversely proportional to $\widetilde{m}$ and $\widetilde{K}$.

For the last, we changed only one parameter among N, m and K in consecutive order to get more analyses. It can be checked that total time consumption is asymptotically linear in N, m, and K. Also it is notable that while the algorithm suggested in Rao *et al.* [8] took 337 minutes to finish one iteration with $N = 10,000$, $m = 10$ and $K = 4$, it took only 52 seconds for our algorithm to finish one iteration with the same dataset and parameters N, m and K.

The results are neatly visualized in the following Table 3 and Figure 2.

Table 3: Results with varying N, m and K with 80-bit security, respectively. All the timing results are reported in seconds.

N	Step 2		Step 3			
($m = 8$, $K = 4$)	DO_i	MSP	MSP	KM	DO_i	Total
5000	3.1	0.2	15.6	2.2	2.3	20.1
10000	5.7	0.2	22.8	4.4	4.5	21.7
15000	8.1	0.2	29.5	6.5	6.6	42.6
20000	10.9	0.2	36.8	8.5	8.8	54.1

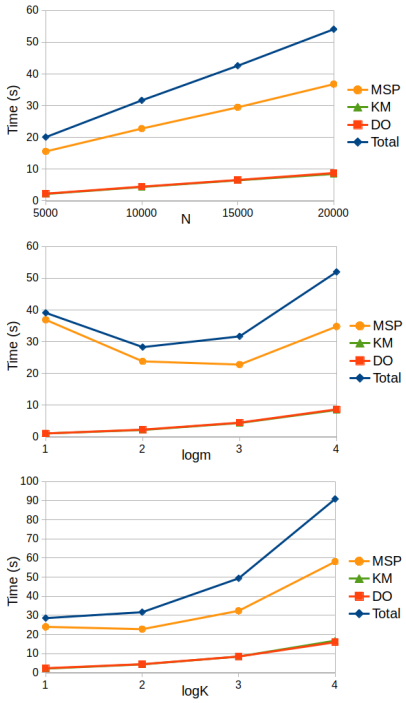
m	Step 2		Step 3			
($K = 4$, $N = 10^4$)	DO_i	MSP	MSP	KM	DO_i	Total
2	1.8	0.2	36.9	1.1	1.1	39.1
4	3.1	0.2	23.8	2.2	2.3	28.3
8	5.7	0.2	22.8	4.4	4.5	31.7
10	8.0	0.2	34.8	8.5	8.7	52.0
16	10.8	0.2	34.8	8.5	8.7	52.0

K	Step 2		Step 3			
($N = 10^4$, $m = 8$)	DO_i	MSP	MSP	KM	DO_i	Total
2	2.9	0.1	24.0	2.2	2.4	28.6
4	5.7	0.2	22.8	4.4	4.5	31.7
8	11.1	0.4	32.4	8.5	8.5	49.4
16	21.8	0.8	58.1	16.8	16.0	90.9

5 Conclusion

In this paper, we propose a privacy-preserving K-means clustering algorithm employing a fully homomorphic encryption HEAAN and Paillier encryption. We use a two-server model to prevent any information leakage about centroids. In the single server model, it is inevitable that the server knows the results of the clustering algorithm Also, we use interaction between users MSP for each iteration because the cost of comparison with encrypted data such as sorting and argmax is too expensive. Indeed, we implement our scenario with 80-bit security parameter of HEAAN and Paillier encryption, and obtain the following an experimental result: clustering 10,000 data with 8 features into 4 clusters takes less than a minute per iteration.

Fig. 2: Results with varying N, m and K, respectively. All the timing results are reported in seconds.



References

1. B. S. Everitt, S. Landau, and M. Leese, "Clustering analysis," *Arnold, London*, 2001.
2. J. A. Hartigan and J. Hartigan, *Clustering algorithms*. Wiley New York, 1975, vol. 209.
3. Z. Kakushadze and W. Yu, "*k-means and cluster models for cancer signatures," *Biomolecular Detection and Quantification*, vol. 13, no. Supplement C, pp. 7 – 31, 2017. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S2214753517302061
4. M. Umamaheswari and P. Isakki alias Devi, "Myocardial infarction prediction using k-means clustering algorithm," *International Journal of Innovative Research in Computer and Communication Engineering*, vol. 5, 2017.
5. S. Chakraborty, N. Nagwai, and L. Dey, "Weather forecasting using incremental k-means clustering," vol. 4, 06 2012.
6. L. W. Wakoli, A. Orto, and S. Mageto, "Application of the k-means clustering algorithm in medical claims fraud / abuse detection," *International Journal of Application or Innovation in Engineering & Management*, vol. 3, 2014.
7. J. H. Cheon, A. Kim, M. Kim, and Y. Song, "Homomorphic encryption for arithmetic of approximate numbers," IACR Cryptology ePrint Archive, 2016: 421, Tech. Rep., 2016, to appear in the proceedings of Asiacrypt 2017.

8. F.-Y. Rao, B. K. Samanthula, E. Bertino, X. Yi, and D. Liu, "Privacy-preserving and outsourced multi-user k-means clustering," *2015 IEEE Conference on Collaboration and Internet Computing (CIC)*, vol. 00, pp. 80–89, 2015.
9. F. Meskine and S. Nait-Bahloul, "Privacy preserving k-means clustering: A survey research," vol. 9, 03 2012.
10. J. Vaidya and C. 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*, ser. KDD '03. New York, NY, USA: ACM, 2003, pp. 206–215. [Online]. Available: http://doi.acm.org/10.1145/956750.956776
11. M. C. Doganay, T. B. Pedersen, Y. Saygin, E. Savaş, and A. Levi, "Distributed privacy preserving k-means clustering with additive secret sharing," in *Proceedings of the 2008 International Workshop on Privacy and Anonymity in Information Society*, ser. PAIS '08. New York, NY, USA: ACM, 2008, pp. 3–11. [Online]. Available: http://doi.acm.org/10.1145/1379287.1379291
12. S. Jha, L. Kruger, and P. McDaniel, *Privacy Preserving Clustering*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 397–417. [Online]. Available: https://doi.org/10.1007/11555827_23
13. S. Samet, A. Miri, and L. Orozco-Barbosa, "Privacy preserving k-means clustering in multi-party environment." pp. 381–385, 01 2007.
14. G. Jagannathan and R. 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*, ser. KDD '05. New York, NY, USA: ACM, 2005, pp. 593–599. [Online]. Available: http://doi.acm.org/10.1145/1081870.1081942
15. C. Su, F. Bao, J. Zhou, T. Takagi, and K. Sakurai, "Privacy-preserving two-party k-means clustering via secure approximation," in *Proceedings of the 21st International Conference on Advanced Information Networking and Applications Workshops - Volume 01*, ser. AINAW '07. Washington, DC, USA: IEEE Computer Society, 2007, pp. 385–391. [Online]. Available: http://dx.doi.org/10.1109/AINAW.2007.295
16. P. Bunn and R. Ostrovsky, "Secure two-party k-means clustering," in *Proceedings of the 14th ACM Conference on Computer and Communications Security*, ser. CCS '07. New York, NY, USA: ACM, 2007, pp. 486–497. [Online]. Available: http://doi.acm.org/10.1145/1315245.1315306
17. J. Sakuma and S. Kobayashi, "Large-scale k-means clustering with user-centric privacy preservation," in *Proceedings of the 12th Pacific-Asia Conference on Advances in Knowledge Discovery and Data Mining*, ser. PAKDD'08. Berlin, Heidelberg: Springer-Verlag, 2008, pp. 320–332. [Online]. Available: http://dl.acm.org/citation.cfm?id=1786574.1786606
18. M. Upmanyu, A. M. Namboodiri, K. Srinathan, and C. V. Jawahar, *Efficient Privacy Preserving K-Means Clustering*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 154–166. [Online]. Available: https://doi.org/10.1007/978-3-642-13601-6_17
19. D. Liu, E. Bertino, and X. Yi, "Privacy of outsourced k-means clustering," in *Proceedings of the 9th ACM Symposium on Information, Computer and Communications Security*, ser. ASIA CCS '14. New York, NY, USA: ACM, 2014, pp. 123–134. [Online]. Available: http://doi.acm.org/10.1145/2590296.2590332
20. "Privacy-preserving kernel k-means outsourcing with randomized kernels," *2013 IEEE 13th International Conference on Data Mining Workshops*, vol. 00, pp. 860–866, 2013.

21. A. İnan, S. V. Kaya, Y. Saygın, E. Savaş, A. A. Hintoğlu, and A. Levi, "Privacy preserving clustering on horizontally partitioned data," *Data & Knowledge Engineering*, vol. 63, no. 3, pp. 646 – 666, 2007, 25th International Conference on Conceptual Modeling (ER 2006). [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0169023X0700047X
22. Z. Erkin, T. Veugen, T. Toft, and R. L. Lagendijk, "Privacy-preserving distributed clustering," *EURASIP Journal on Information Security*, vol. 2013, no. 1, p. 4, Nov 2013. [Online]. Available: https://doi.org/10.1186/1687-417X-2013-4
23. 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*, vol. 13, no. 4, pp. 2066–2076, Aug 2017.
24. M. Beye, Z. Erkin, and R. L. Lagendijk, "Efficient privacy preserving k-means clustering in a three-party setting," in *2011 IEEE International Workshop on Information Forensics and Security*, Nov 2011, pp. 1–6.
25. P. Paillier *et al.*, "Public-key cryptosystems based on composite degree residuosity classes," in *Eurocrypt*, vol. 99. Springer, 1999, pp. 223–238.
26. C. Gentry, "A fully homomorphic encryption scheme," Ph.D. dissertation, Stanford University, 2009, crypto.stanford.edu/craig.
27. ——, "Fully homomorphic encryption using ideal lattices," in *Proceedings of the 41st annual ACM symposium on Symposium on theory of computing-STOC'09*. ACM Press, 2009, pp. 169–169.
28. M. van Dijk, C. Gentry, S. Halevi, and V. Vaikuntanathan, "Fully homomorphic encryption over the integers," in *Proc. of EUROCRYPT*, ser. LNCS, vol. 6110. Springer, 2010, pp. 24–43.
29. J.-S. Coron, A. Mandal, D. Naccache, and M. Tibouchi, "Fully homomorphic encryption over the integers with shorter public keys," in *Annual Cryptology Conference*. Springer, 2011, pp. 487–504.
30. J.-S. Coron, D. Naccache, and M. Tibouchi, "Public key compression and modulus switching for fully homomorphic encryption over the integers," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2012, pp. 446–464.
31. J. H. Cheon, J.-S. Coron, J. Kim, M. S. Lee, T. Lepoint, M. Tibouchi, and A. Yun, "Batch fully homomorphic encryption over the integers," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2013, pp. 315–335.
32. J.-S. Coron, T. Lepoint, and M. Tibouchi, "Cryptanalysis of two candidate fixes of multilinear maps over the integers," *IACR Cryptology ePrint Archive*, vol. 2014, p. 975, 2014.
33. J. H. Cheon, J. Kim, M. S. Lee, and A. Yun, "CRT-based fully homomorphic encryption over the integers," *Information Sciences*, vol. 310, pp. 149–162, 2015.
34. Z. Brakerski and V. Vaikuntanathan, "Efficient fully homomorphic encryption from (standard) lwe," *SIAM Journal on Computing*, vol. 43, no. 2, pp. 831–871, 2014.
35. ——, "Fully homomorphic encryption from ring-lwe and security for key dependent messages," in *Annual cryptology conference*. Springer, 2011, pp. 505–524.
36. Z. Brakerski, "Fully homomorphic encryption without modulus switching from classical gapsvp." in *CRYPTO*, vol. 7417. Springer, 2012, pp. 868–886.
37. Z. Brakerski, C. Gentry, and V. Vaikuntanathan, "(leveled) fully homomorphic encryption without bootstrapping," *ACM Transactions on Computation Theory (TOCT)*, vol. 6, no. 3, p. 13, 2014.

38. C. Gentry, S. Halevi, and N. P. Smart, "Fully homomorphic encryption with polylog overhead," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2012, pp. 465–482.
39. ——, "Homomorphic evaluation of the aes circuit," in *Advances in Cryptology–CRYPTO 2012*. Springer, 2012, pp. 850–867.
40. A. López-Alt, E. Tromer, and V. Vaikuntanathan, "On-the-fly multiparty computation on the cloud via multikey fully homomorphic encryption," in *Proceedings of the forty-fourth annual ACM symposium on Theory of computing*. ACM, 2012, pp. 1219–1234.
41. J. W. Bos, K. E. Lauter, J. Loftus, and M. Naehrig, "Improved security for a ring-based fully homomorphic encryption scheme." in *IMA Int. Conf.* Springer, 2013, pp. 45–64.
42. C. Gentry, A. Sahai, and B. Waters, "Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based," in *Advances in Cryptology–CRYPTO 2013*. Springer, 2013, pp. 75–92.
43. J. H. Cheon and D. Stehlé, "Fully homomophic encryption over the integers revisited." *EUROCRYPT (1)*, vol. 9056, pp. 513–536, 2015.
44. Y. Doröz, Y. Hu, and B. Sunar, "Homomorphic aes evaluation using the modified LTV scheme," *Designs, Codes and Cryptography*, vol. 80, no. 2, pp. 333–358, 2016.
45. L. Ducas and D. Micciancio, "FHEW: Bootstrapping homomorphic encryption in less than a second." *EUROCRYPT (1)*, vol. 9056, pp. 617–640, 2015.
46. M. Naehrig, K. Lauter, and V. Vaikuntanathan, "Can homomorphic encryption be practical?" in *Proceedings of the 3rd ACM workshop on Cloud computing security workshop*. ACM, 2011, pp. 113–124.
47. J. W. Bos, K. Lauter, and M. Naehrig, "Private predictive analysis on encrypted medical data," *Journal of biomedical informatics*, vol. 50, pp. 234–243, 2014.
48. K. Lauter, A. López-Alt, and M. Naehrig, "Private computation on encrypted genomic data," in *International Conference on Cryptology and Information Security in Latin America*. Springer, 2014, pp. 3–27.
49. J. H. Cheon, M. Kim, and K. Lauter, "Homomorphic computation of edit distance," in *International Conference on Financial Cryptography and Data Security*. Springer, 2015, pp. 194–212.
50. S. Wang, Y. Zhang, W. Dai, K. Lauter, M. Kim, Y. Tang, H. Xiong, and X. Jiang, "Healer: Homomorphic computation of exact logistic regression for secure rare disease variants analysis in gwas," *Bioinformatics*, vol. 32, no. 2, pp. 211–218, 2015.
51. J. H. Cheon, J. Jeong, J. Lee, and K. Lee, "Privacy-Preserving Computations of Predictive Medical Models with Minimax Approximation and Non-Adjacent Form," to appear in International Conference on Financial Cryptography and Data Security. Springer, 2017.
52. S. Halevi and V. Shoup, "Design and implementation of a homomorphic-encryption library," *IBM Research (Manuscript)*, 2013.
53. ——, "Algorithms in HElib," in *International Cryptology Conference*. Springer, 2014, pp. 554–571.
54. "Homomorphic Encryption for Arithmetic of Approximate Numbers," https://github.com/kimandrik/HEAAN.
55. K. Fukunaga, *Introduction to Statistical Pattern Recognition.*, ser. Computer Science and Scientific Computing. Academic Press, 1990, vol. 2nd ed. [Online]. Available: http://search.ebscohost.com/login.aspx?direct=true&db=nlebk&AN=196121&site=ehost-live

56. J. H. Cheon, M. Kim, and M. Kim, "Search-and-compute on encrypted data," in *International Conference on Financial Cryptography and Data Security*. Springer, 2015, pp. 142–159.
57. M. Naeem, S. Asghar, and M. Lichman, "UCI machine learning repository," 2011. [Online]. Available: http://archive.ics.uci.edu/ml