Implementing Private K-Means Clustering Using a LWE-based Cryptosystem

Anastasia Theodouli
Dept. of Informatics
Aristotle Univ. of Thessaloniki
Thessaloniki, Greece
Email: anastath@csd.auth.gr

Konstantinos A. Draziotis
Dept. of Informatics
Aristotle Univ. of Thessaloniki
Thessaloniki, Greece
Email: drazioti@csd.auth.gr

Anastasios Gounaris
Dept. of Informatics
Aristotle Univ. of Thessaloniki
Thessaloniki, Greece
Email: gounaria@csd.auth.gr

***Abstract*—Combining data analytics with homomorphic encryption is an interesting topic which enables clients with low computational and/or storage capacity to outsource the analysis of potentially large datasets to the cloud while protecting sensitive data from unwanted access. In this work, we propose a framework for evaluating k-means clustering using a cryptosystem based on the Learning With Errors (LWE) problem. We implemented three variants of this framework in the computer algebra system Sagemath and executed many experiments to test the performance for various values of k-means and LWE parameters.**

1. Introduction

The volume of data that is nowadays produced exceeds our capacity to store and analyze them by orders of magnitude; e.g., the LHC experiment at CERN keeps only a tiny fraction (at the order of 10^{-6}) of the produced data to allow for economical storage and analysis [1]. Similar problems are faced by most top organizations and enterprises in the world. In both scientific and enterprise settings, the main direction followed is to employ shared computing infrastructures, which has led to the establishment of public cloud computing solutions. According to a report(1), "*at year-end 2016, more than 50% of Global 1000 companies will have stored customer-sensitive data in the public cloud.*" Albeit, resorting to cloud infrastructures raises concerns regarding the loss of control over data and the consequent compromise of security and privacy. One technique widely used to secure the data transferred to and analyzed on the cloud is called *homomorphic encryption (HE)*. As we argue in this paper, HE is of interest for storing and analyzing data from (medical) IoT devices as well.

In 2009, Gentry in his landmark paper [2] provided a specific construction of a fully homomorphic scheme (FHS). The idea of FHS first appeared in 1978 by Rivest, Shamir and Dertouzos [3]. Taking as an example RSA-cryptosystem which is multiplicative homomorphic, they posed a more general question, if we could really compute (any) function over encrypted data in a homomorphic way.

(1). https://www.gartner.com/doc/1991317/information-innovation-innovation-key-initiative

This work is based on the client-server setting. Data belongs *exclusively* to the client, who does not want to share them. On the other hand, the client has limited computational and/or storage resources and thus needs to outsource the computation of a specific analysis, namely $k-$means clustering, to the cloud. For instance, consider an IoT medical device that collects data from a patient, but cannot keep it due to limited memory. In such a case, the device should offload the data and the computation load to a cloud-hosted trusted server that needs also to perform some type of analysis in order physicians to take informed decisions. Note that data sizes need not be big but larger than the storage capacities of the IoT device, something that is quite common. Therefore, combining data mining with HE is an interesting topic, since it enables clients with low computational and/or storage capacity to outsource the analysis of potentially large datasets to the cloud while protecting sensitive data from unwanted access, and has started been explored recently [4], [5].

In this paper, we consider the k-means clustering algorithm as the operation to be done securely on the data on the server side. We use the system in [6], which base its security on the learning with errors problem (LWE) of Regev [7]. This system belongs to the class of somewhat homomorphic schemes (SHS) rather than to FHSs. SHSs allow us to make as many additions as we want over the encrypted data, but multiplications only of specific depth. For instance, we can not make a multiplication of 20 integers, since, in that case, the *noise* we add to the system does not allow us to correctly decrypt the data. In FHS, Gentry suggested the bootstrapping step to *refresh* the encrypted data and decrease the *noise* over the data. The strong point of SHS is that it is more efficient than FHS in practical applications (e.g. see [5]), given that several real-world data science and data mining algorithms require only a limited number of multiplications on each data record rather than the full the power that a FHS provides.

In the specific case of k-means, the choice of a SHS comes at the expense of allocating some workload to the client. More specifically, the main contribution of this work is that we present three collaboration protocols between the client and the server that make different trade-offs between security, client load and minimum client resources required.

The reasons that we do not fully implement k-means on the server's side is twofold. First, the algorithm requires computation of operations that compare numbers and find the minimum from a list. The circuit that we need to implement in order to compute the minimum of some data is not very efficient, i.e., has very large depth, (see [5]) and so it cannot be efficiently implemented in a SHS. Second, say we have a relation of the form $Enc(a) > Enc(b)$ (i.e. an order over the encrypted data), then this will reveal *some* information to the server, which may destroy the semantic security of the system, i.e. it may reveal some information for the unencrypted messages a or/and b.

In summary, our work aims to make another step towards implementing advanced data analytics according to a HE scheme. Our current results highlight bottlenecks and provide strong insights into the issues that need to be resolved in order such techniques to scale to large datasets.

Structure of the remainder of this paper. In the next section, we provide the necessary background. We give the exact definition of FHS, our security model, a description of the SHS BV-11 cryptosystem [6] and the underlying Learning with errors (LWE) problem. In section 3, we provide the related work on the secure implementation of k-means. In section 4, we present our framework for implementing k-means in a homomorphic way. Furthermore, in Section 5, we provide some experimental results concerning the performance of our framework as implemented in Sagemath [8]. Finally, in the last section we provide some concluding remarks and some possible extensions of this work.

2. Homomorphic Encryption and LWE Basics

We start with the definition of a FHS.

Definition *1.* (FHS) A quadruple consisting of four algorithms $H = \{KeyGen, Enc, Dec, Eval\}$ is called *FHS* if it has the following properties:
Homomorphic. $KeyGen$ generates a pair of public and secret key (pk, sk). Let f be any function defined over the message space and m_i $(i = 1, 2, ..., t)$ are some messages and $c_i = Enc_{pk}(m_i)$ are the corresponding ciphertexts. Enc (resp. Dec) is the encryption (resp. decryption) function. If

$$c^* = Eval(f, c_1, c_2, ..., c_t),$$

then

$$Dec_{sk}(c^*) = f(m_1, m_2, ..., m_t).$$

Security. We assume that the scheme is semantically secure (with the usual notion in public key cryptosystems).
Compact. c^* is independent on the complexity of f (i.e. decrypting c^* is easier than computing f).

Security Model We assume that the cloud provider is *honest-but-curious*. Honest means that the cloud-hosted server-side processing will follow the protocol as defined by the data owner and will return the results. Curious means that the cloud can look at the data; thus the data owner on the client side needs to devise the processing protocols in such a way that no or minimal information leakage takes place.

2.1. LWE and BV-11

In 2005, Regev defined the *learning with errors* (LWE) problem [7]. Before we formally define it, we need the definition of the LWE distribution. Let an integer $N \geq 1$, a prime $q \geq 2$ and $\mathbf{s} \in \mathbb{Z}_q^N$ (where $\mathbb{Z}_q$ is the finite field with q elements, with addition and multiplication $\bmod q$). We define the LWE distribution $\mathcal{A}_{\mathbf{s},\chi}$ over $\mathbb{Z}_q^N \times \mathbb{Z}_q$ by the following process:
• pick a vector $\mathbf{a}$ uniformly from $\mathbb{Z}_q^N$ and e from $\mathbb{Z}_q$ according to a (discrete) distribution χ over $\mathbb{Z}$.
• return $(\mathbf{a}, \mathbf{a} \cdot \mathbf{s} + e)$, i.e. the inner product of $\mathbf{s}$ and $\mathbf{a}$ plus a *noise* e and then reduce by $\bmod q$.

Definition *2.* (LWE problem) Let $\mathbf{s}$ be arbitrary in $\mathbb{Z}_q^N$. Having arbitrary many pairs $\{(\mathbf{a}_i, \mathbf{a}_i \cdot \mathbf{s} + e_i)\}_i$ compute $\mathbf{s}$.

If we choose χ as the discrete Gaussian distribution over $\mathbb{Z}_q$ with mean zero and $\sigma = \alpha q/\sqrt{2\pi}$ (where $\alpha \in (0, 1)$) and $q = O(poly(n))$ then, this problem has been proved that it is as hard as a standard hard lattice problem [9].

In [6], the authors provide a cryptosystem based on the LWE problem, which we call it **BV-11**. The cryptosystem is briefly described as follows:
(i) Choose an integer $t \ll q$ and an integer number $m \in (-t/2, t/2)$ to encrypt.
(ii) Pick a random vector of $\mathbb{Z}_q^N$, say $\mathbf{a}$. Then the encryption function is $\mathbf{c} = enc_{\mathbf{s}}(m) = (\mathbf{a}, \mathbf{a} \cdot \mathbf{s} + te + m) = (\mathbf{a}, b)$ (all the operations are $\bmod q$), where $\mathbf{s}$ is the secret key.
(iii) In order to decrypt, construct the polynomial $f_{\mathbf{a},b}(\mathbf{x}) = b - \mathbf{a} \cdot \mathbf{x} \in \mathbb{Z}_q[x]$. Then, the decryption function is $dec_{\mathbf{s}}(\mathbf{c}) = f_{\mathbf{a},b}(\mathbf{s}) \pmod t = m$.

This system is symmetrical (same encryption-decryption key : $\mathbf{s}$). To see that is homomorphic with respect addition and multiplication, we choose two messages m_1, m_2 and let $\mathbf{c}_1 = (\mathbf{a}_1, \mathbf{a}_1 \cdot \mathbf{s} + te_1 + m_1) = (\mathbf{a}_1, b_1)$, $\mathbf{c}_2 = (\mathbf{a}_2, \mathbf{a}_2 \cdot \mathbf{s} + te_2 + m_2) = (\mathbf{a}_2, b_2)$. Then someone that does not have the secret key, for the addition, computes the coefficients of the polynomial $f_{\mathbf{a}_1,b_1}(\mathbf{x}) + f_{\mathbf{a}_2,b_2}(\mathbf{x})$ and form a vector say $\mathbf{c}_{add}$; similarly, for the multiplication, computes the coefficients of the polynomial $f_{\mathbf{a}_1,b_1}(\mathbf{x}) f_{\mathbf{a}_2,b_2}(\mathbf{x})$ and form a vector say $\mathbf{c}_{mult}$. Now, someone that has the secret key and having $\mathbf{c}_{add}$ and $\mathbf{c}_{mult}$ constructs the two previous polynomials and set $\mathbf{x} = \mathbf{s}$. Then the result will be $(m_1 + m_2) \pmod t$ and $m_1 m_2 \pmod t$, respectively. Choosing carefully the parameter t i.e. $m_1 + m_2, m_1 m_2 < t$ we get $m_1 + m_2$ and $m_1 m_2$, respectively.

To choose secure parameters for LWE we follow [10] (see also Section 4.4).

3. Related Work

Here, we briefly discuss other proposals for securely implementing k-means. Vaidya and Clifton proposed a privacy-preserving k-means clustering algorithm for vertically partitioned data [11]. Their protocol involves more than two

parties, i.e., it does not fit into the client-sever model. Another approach to designing a secure k-means clustering algorithm in arbitrarily partitioned data for a two-party setting has been presented by Bunn and Ostrovsky in [12]. In this work, the authors have combined cryptographic security tools with several secure multi-party communication (SMC) protocols able to be readily implemented only using the so-called Scalar Product Protocol, which has been extensively explored, see e.g. [13]. Protocols for SMC have been proposed by the authors in [14] as well. Finally, based on the work of [15], Biswas et. al devised and implemented a prototype which allows multiple parties to compute a k-means algorithm using their disjoint, horizontally partitioned datasets as input while retaining privacy of each party's input [16]. Compared to the above proposals, apart from focusing on a client-server setting, we differ in that we use as a security tool for encryption, the BV-11 cryptosystem which is based on the LWE problem, while most of the related work uses the Pallier's cryptosystems which is partially homomorphic and base its security in hard problems of number theory. Furthermore, Pallier's cryptosystem is not quantum resistant, since the *Decisional Composite Residuosity* problem on which Pallier's cryptosystem is based is an easy one if factorization is easy, and the factorization problem in quantum computers is an easy problem indeed.

Further application of HE to data mining problems are discussed in works, such as [4], [5], which do not deal with k-means.

4. The proposed framework for securely implementing $k-$means

Here, we describe three variants of a *secure* protocol for k-means. Each one has some positive and negative points with regards to the performance and the security of the system and different trade-offs are made regarding the load on each side.

In all three variants, the client and server agree on some secure parameters of BV-11 system. The basic steps of $k-$means algorithm, as considered in this work, are presented below.

Input: A dataset of n d-dimensional integer points, $\{P_1, \ldots, P_n\} \subset \mathbb{Z}^d$, and a positive integer k, which denotes the number of clusters, and a threshold $d_{threshold}$. It is safely assumed that $k \ll n$). The threshold is used in the stopping criterion.

Output: k d-dimensional points of $\mathbb{R}^d$ denoting the cluster centers.

1) Consider k random points from the dataset as the initial centers.
2) Assign each point to the group with the closest euclidean distance from the center of the group.
3) When all points have been assigned to a cluster, recalculate the positions of the k centers by computing the *average* of the points belonging to each group.
4) Repeat steps 2 and 3 if, for at least one of the k centers, the distance of the previous and the new center is greater than $d_{threshold}$; the difference is computed by Euclidean distance.(2)

In all variants, we assume that the available memory on the server is too small to hold the nd point data, but is adequate to hold both the intermediate and final result, which consists of kd values; actually the minimum required memory is the amount needed to store the centers in two iterations, while holding the sum per dimension and the amount of data per cluster separately, which amounts to $2k(d+1)$ values. The 2nd step is always computed via a collaboration between the server and the client, and the 4th step is always computed on the client. The type of the collaboration and the execution of the 3rd step is where the variants differ. The collaboration is needed because the server can compute homomorphically the distance between the data points and the centers in an efficient manner, but cannot establish which distance is the smallest. This also implies that the client needs additional memory to hold k more values, so that it can check the minimum distance from a point to a center. Therefore, the minimum client memory required is space for $2k(d+1)+k$ integers.

In the following, we use the notation $Q_i = Enc(P_i)$ for the homomorphic encryption of the point P_i, i.e. $Enc(P_i) = (Enc(P_i^{(1)}), \ldots, Enc(P_i^{(d)}))$. The black dots correspond to operations on the client side, whereas white dots correspond to operations on the server side.

4.1. Variant I: computing the new centers on the client

The details of the first variant are as follows:

1) *Initialization.*
   • The client encrypts each of the $d-$entries of $P_1, \ldots, P_n$ and sends them to the server.

   ◦ The server permanently stores the encrypted data in a matrix A. It also picks the first $kd-$encrypted entries to form the initial centers C_i, $i = 1, 2, ..., k$.

2) *Finding the closest cluster to each data point-Assignment step.*
   ◦ The server (homomorphically) computes the k squares of distances $dist(Q_j, C_i)^2$ for $i = 1, 2, \ldots, k$ and $j \in \{1, \ldots, n\}$. Finally, the server sends the distances back to the client along with Q_j (3).

   • The client decrypts all the distances and computes their minimum. The client keeps the sum for each dimension for each cluster along

(2). It is also common to allow steps 2 and 3 to run for a predefined number of iterations.

(3). In the first round, j starts from $k+1$ since $C_i = Enc(P_i)(1 \leq i \leq k)$

with the number of the points of each cluster in each iteration; the dimension values of P_j are retrieved through decrypting Q_j.

3) *Re-calculate the positions of the k centers- Update step.*
• The client calculates the new centers, i.e., it divides each of the d numbers with the number of the points in each cluster and keeps the nearest integer. If at least one new center differs from its previous value by at least $d_{threshold}$, it encrypts the new centers, sends them to the server, and the assignment step is re-executed.

An analysis of the 1st variant is as follows. In each iteration, $n(d+k)$ encrypted values are sent back from the server to the client. The client needs some extra memory to hold this data. Assuming that the transmission is in batches, the smallest batch corresponds to the data of a single point and comprises $d+k$ encrypted values. Thus, the client needs to have available memory for at least $2k(d+1)+d+k$ values. Also, the client performs $n(d+1)$ additions and d divisions when storing the intermediate center information and the final center coordinates, respectively.

4.2. Variant II: computing the new centers on the server with information leakage

The second variant aims to alleviate the load on the client side and perform the computation of the sum of each dimension per cluster on the server side. This also makes the need to send Q_j back to the client obsolete. More specifically, the assignment and update steps are modified as follows (see also Figure 1):

1) *Finding the closest cluster to each data point- Assignment step.*
◦ The server (homomorphically) computes the k squares of distances $dist(Q_j, C_i)^2$ for $i = 1, 2, \ldots, k$ and $j \in \{1, \ldots, n\}$ and sends the distances back to the client for each Q_j without sending Q_j per se.

• The client decrypts all the k distances and computes their minimum. The client sends the identifier i of the minimum distance back to the server in an unencrypted form. The client only keeps a counter for the number of data points assigned to each cluster in the current iteration.

2) *Re-calculate the positions of the k centers- Update step.*
◦ The server homomorphically computes the sum for each coordinate for each center and sends the result to the client.

• The client decrypts the information received and calculates the new centers, i.e., it divides each of the d numbers with the number of the points in each cluster and keeps the nearest integer. If at least one new center differs from its previous values by at least $d_{threshold}$, it encrypts the new centers, sends them to the server, and the assignment step is re-executed.

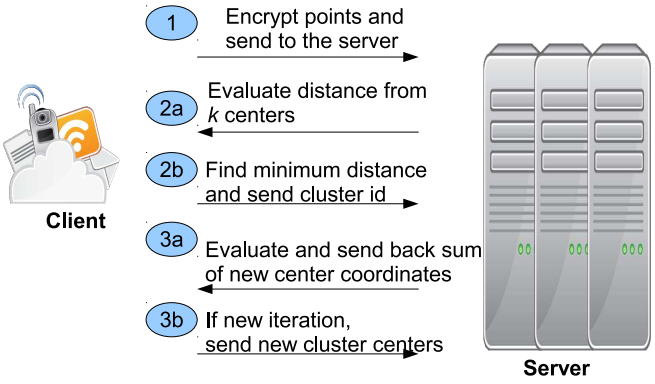

Fig. 1 A summary of the 2nd variant.

The memory requirements on the client are very slightly affected. The main advantage of this variant is that the data communicated during the assignment step drops to nk encrypted values and only n additions are performed on the client. This comes at the expense of a) $O(nd)$ homomorphic additions on the server, and b) some security leakage, in the sense that the server knows about the cluster identifiers for each encrypted point.

Note that we can avoid sending the cluster identifier in plaintext using AES (or some other secure symmetric encryption algorithm). This can be done easily if we have a Pubic key infrastructure. In more details, the server must have a digital certificate signed by a trusted authority. Then before the initialization (step 1), the client and server can agree to a common key, say 128 bits, which can be used by AES. So, in this way we can avoid a possible *Man in the Middle* attack. Furthermore, not keeping secret the cluster identifier, gives rise to a potential threat of someone changing its value; in this way, the results will be false. But through encrypting the cluster identifiers and assuming that the server is trusted in the sense that will execute the framework as presented, our results will be correct.

4.3. Variant III: computing the new centers on the server without information leakage

This variant aims to address the main limitation of the previous one, namely to reveal the cluster identifiers for each point to the server. To this end, the assignment and update steps are modified as follows:

1) *Finding the closest cluster to each data point- Assignment step.*
◦ The server (homomorphically) computes the k squares of distances $dist(Q_j, C_i)^2$ for $i = 1, 2, \ldots, k$ and $j \in \{1, \ldots, n\}$ and sends the

Points in matrix A				Clustering in matrix B		
d_1	d_2	d_3	d_4	c_1	c_2	c_3
2	8	5	3	0	1	0
6	6	2	1	1	0	0
2	1	5	2	0	0	1
5	2	7	6	1	0	0
3	4	9	6	1	0	0

TABLE 1: Example of the (encrypted) information on the server according to the 3rd variant.

	Variant I	Variant II	Variant III
encryptions	kd	kd	$(n+k)d$
decryptions	$n(k+d)$	nk	nk
transmission	$n(k+d)$	nk	$n(k+d)$
additions	$nk(d-1)$	$nk(d-1)$ + $(n-k)d$	$nk(d-1)$ + $kd(n-1)$
multiplications	nd	nd	$nd(k+1)$
extra client memory	d	0	k

TABLE 2: Summary of amount of operations on encrypted integers per iteration of k-means.

distances back to the client for each Q_j without sending Q_j per se.

• The client decrypts all the k distances and computes their minimum. Let us assume that the identifier of the cluster with the minimum distance is i. The client creates a vector with k entries, where all entries are 0 apart from the i-th one, which is set to 1. It encrypts this vector and sends it to the server.

◦ The server stores the encrypted vector row-by-row. It forms a matrix $n \times k$, which we call it B. Table 1 shows an example of the total information stored on the server for 5 4-dimensional points and 3 centers.

2) *Re-calculate the positions of the k centers- Update step.*
◦ The server has already formed a matrix A having as columns the encrypted points (i.e. $n \times d$) and is homomorphically multiplied by B (i.e. dk inner products are computed). The resulting dk values correspond to the sums for each dimension for each center. These values are sent to the client.

• The client decrypts the information received and calculates the new centers, i.e., it divides each of the d numbers with the number of the points in each cluster and keeps the nearest integer. If at least one new center differs from its previous values by at least $d_{threshold}$, it encrypts the new centers, sends them to the server, and the assignment step is re-executed.

Table 2 summarizes the main features of each variant. Encryptions and decryptions take place on the client, whereas additions and multiplications on encrypted data are performed on the server. From the table, it is clear that the 1st (resp. 3rd) variant places more load on the client (resp. server), whereas the 2nd variant is less loaded, but at the expense of a compromise on the information revealed.

enc./dec.	add./dec.	mult./dec.
$0.002/0.52s$	$10^{-4}/0.67s$	$6.15/355s$

TABLE 3: The cpu times for encrypting/decrypting an integer with 20-bits (left), adding and decrypting two 20-bit integers (middle) and multiplying two 20-bit integers.

d	inner prod./dec.	square of distance/dec.
2	$13/377s$	$13/350s$
5	$61/406s$	$52/381s$
10	$136/427s$	$116/361s$

TABLE 4: The cpu times for homomorphic inner product and the square of their distance between two vectors in $\mathbb{Z}^d$ for 3 values of d; we considered vectors with entries integers of 20-bits.

4.4. Implementation Issues

As already mentioned, BV-11 is a SHS allowing as many additions and one multiplication. After an homomorphic multiplication the size of the multiplicative ciphertext grows up to $O(N^2)$ coefficients, where the input ciphertexts consist of N coefficients each. For d-multiplications (of two integers) we have $dO(N^2)$ coefficients. So for relatively small d we can control the noise, in order to get correct decryption. Much care is needed when we choose the value t of BV-11, so as to decrypt right. The parameters of LWE are N, a suitable prime q and the parameters for the Gaussian α. Furthermore, for BV-11, we considered three more parameters, an integer t which defines the message space $\{-\lfloor t/2 \rfloor, ..., \lfloor t/2 \rfloor\}$, an integer ℓ which is the maximum number of bits for the integers defining our points, and ρ, which expresses the depth of the multiplications we use. We choose the previous parameters as follows: $t = 2^{\lfloor 2 \cdot \ell + 4 \ln(\rho) \rfloor}$, $\rho = 4$, q is set to the prime nearest to the integer $t \cdot 2^{16.5\rho + 5.4} 8^{2(\rho-3)} N^{\rho}$ and $\alpha \cdot q = 3.2\sqrt{2\pi} = \sigma \cdot \sqrt{2\pi}$ (for the last choice see [10, section 7]).

We constructed a class which has the following methods: `encrypt`,`decrypt`,`add`,`mult`,`scalar_mult`, `inner_product` and `hom_distance`. We used a basic class of Sagemath `LWE`(N, q, σ) which has a method that generate LWE-samples. Having the previous methods, it is easy to implement the three suggested variants for k-means. The implementation of this class can be found in https://github.com/drazioti/python_scripts/tree/master/paper_lwe.

5. Experimental results

In our experiments, we used a Ubuntu machine with 8GB RAM and an i5 3.5GHz processor. We implemented the three variants in Sagemath. In Table 3, we provide the CPU times for some simple operations of the BV-11

	Variant I	Variant II	Variant III
$(d = 2, n = 20)$ client:	$810s$	$684s$	$803s$
server:	$209s$	$190s$	$606s$
$(d = 2, n = 100)$ client:	$3723s$	$3771s$	$3880s$
server:	$1030s$	$1095s$	$3127s$
$(d = 5, n = 20)$ client:	$689s$	$798s$	$1005s$
server:	$780s$	$804s$	$1865s$
$(d = 10, n = 20)$ client:	$687s$	$784s$	$1024s$
server:	$1931s$	$1923s$	$3808s$

TABLE 5: Times for two k-means iterations, with $N = 100$ providing ≈ 30-bit security, and the number of clusters $k = 3$.

	Variant I	Variant II	Variant III
$(d = 5, n = 10)$ client:	$12.04h$	$12.12h$	$13.2h$
server:	$1.66h$	$1.65h$	$1.79h$

TABLE 6: Times for two k-means iterations for 90-bit security, and the number of clusters $k = 3$.

cryptosystem. We chose the following parameters for LWE to get 90-bit security $N = 256$, $t = 17592186044416$, q a prime with 132 bits (as we suggested in the previous section), $\alpha \cdot q = 8.02$ and $\sigma = 3.2$. Table 4 shows the times for some more advanced applications for the same level of security, i.e., 90-bit. As can be observed, decryption dominates.

In Table 5, we provide the overall cpu-times for the three suggested variants for k-means for the client and server side if we lower the security of the cryptosystem to a 30-bit one. The times for 90-bit security are significantly higher, as shown in Table 6. The results concern only two iterations.

Of course we should never use the parameters given in Table 5, since BV-11 is not secure. We remark that the first two variants have almost the same performance, but differ in the load they impose on the client. The third variant has the advantage that it is more secure than the 2nd one, since it hides the identifiers of the clusters, but has the worst performance.

6. Conclusions and Future Work

In this work, we investigated a client-server implementation of k-means according to a homomorphic encryption scheme that relies on the simple LWE problem. More specifically, we proposed a framework using the BV-11 SHS. We implemented our framework in Sagemath and executed experiments with parameters giving security $\approx 90-$bits, i.e. the best known attack needs 2^{90} running time to break BV-11. Our source code is publicly available. Our framework has been instantiated in three variants, with different trade-offs regarding the load on the client and the security. In all cases, the client has constrained resources in that it cannot hold the complete dataset.

The value of our results is twofold. First, it provides a basis for practitioners to benefit from our experience in the efficiency of the implementation and build extensions. Second, it proves that a BV-11-based solution cannot scale for a medium or large dataset. Even if we manage to keep the server response times at tolerable levels, e.g., through parallelism, the decryption overhead on the client side remains a dominant cost. Our directions for future work are, first to investigate solutions based on the *ring-LWE* problem [17]; second, to explore approaches that require less collaboration and alleviate the amount of decryptions on the client side. For example, using a FHS, would allow us to implement a circuit for division and comparison, so that the server can become fully responsible for the evaluation of k-means. Then, using parallelism in the context of frameworks such as Spark, we could address server-side bottlenecks.

References

[1] V. V. Gligorov, "Real-time data analysis at the lhc: present and future," *JMLR: Workshop and Conference Proceedings*, vol. 42, pp. 1–18, 2015. [Online]. Available: https://arxiv.org/abs/1509.06173

[2] C. Gentry, "Fully homomorphic encryption using ideal lattices," *STOC '09*, 2009.

[3] R. Rivest, L. Adleman, and M. Dertouzos, "On data banks and privacy homomorphisms," in *Foundations of secure computation*, 1978.

[4] R. Bost, R. A. Popa, S. Tu, and S. Goldwasser, "Machine learning classification over encrypted data," in *22nd Annual Network and Distributed System Security Symposium, NDSS*, 2015.

[5] T. Graepel, K. Lauter, and M. Naehrig, "Ml confidential: Machine learning on encrypted data," pp. 1–21, 2012.

[6] Z. Brakerski and V. Vaikuntanathan, "Efficient fully homomorphic encryption from (standard) lwe," *SIAM Journal on Computing*, vol. 43, no. 2, 2014.

[7] O. Regev, "The learning with errors problem," *Invited survey in CCC 2010 (http://www.cs.tau.ac.il/ odedr/)*.

[8] The Sage Development Team, "Sage mathematics software," *http://www.sagemath.org*, 2015.

[9] Z. Brakerski, A. Langlois, C. Peikert, O. Regevn, and D. Stehle, "Classical hardness of learning with errors," *https://arxiv.org/pdf/1306.0281.pdf*, 2013.

[10] M. R. Albrecht, R. Player, and S. Scott, "On the concrete hardness of learning with errors," *https://eprint.iacr.org/2015/046.pdf*, 2015.

[11] 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*. ACM, 2003, pp. 206–215.

[12] P. Bunn and R. Ostrovsky, "Secure two-party k-means clustering," in *Proceedings of the 14th ACM conference on Computer and communications security*. ACM, 2007, pp. 486–497.

[13] B. Goethals, S. Laur, H. Lipmaa, and T. Mielikäinen, "On private scalar product computation for privacy-preserving data mining," in *International Conference on Information Security and Cryptology*. Springer, 2004, pp. 104–120.

[14] S. Samet, A. Miri, and L. Orozco-Barbosa, "Privacy preserving k-means clustering in multi-party environment." in *SECRYPT*, 2007, pp. 381–385.

[15] S. Samet and A. Miri, "Privacy preserving id3 using gini index over horizontally partitioned data," in *2008 IEEE/ACS International Conference on Computer Systems and Applications*. IEEE, 2008, pp. 645–651.

[16] A. S. Biswas, A. Bubna, D. Doss, and S. Scheffler, "Privacy preserving k-means clustering," 2016.

[17] V. Lyubashevsky, C. Peikert, and O. Regev, "On ideal lattices and learning with errors over rings," *Journal of the ACM (JACM)*, vol. 60, no. 6, p. 43, 2013.