Computing
https://doi.org/10.1007/s00607-019-00711-w

Homomorphically encrypted k-means on cloud-hosted servers with low client-side load

Georgios Sakellariou[1] · Anastasios Gounaris[1]

Received: 11 August 2018 / Accepted: 11 February 2019

Abstract
The significance of data analytics has been acknowledged in many scientific and business domains. However, the required processing power and memory capacity is a prohibiting factor for performing data analytics on proprietary platforms. An obvious solution is the outsourcing of data analytics to cloud storage and cloud computing providers but this entails that privacy and security issues are raised, given the fact that data can be valuable and/or personal. The aim of this paper is the development of a server-side k-means algorithm over encrypted data using homomorphic encryption in order to overcome both the lack of resources of the data owner and the security concerns. Current solutions that deal with homomorphic encryption impose a heavy load on the side of the data owner; this limitation is now addressed in this work. More specifically, in this paper, we present a framework for the implementation of an homomorphic version of k-means, we discuss the capabilities of the current state-of-the-art homomorphic encryption schemes, and we propose a novel approach to server-side computation of k-means assuming a new adversary model tailored to modern settings. We instantiate our framework in two different versions in terms of operation assignment each coming in three flavors of operation implementation. All alternatives are evaluated thoroughly using both real experiments and analytic cost models.

Keywords Homomorphic encryption · Encrypted analytics · Data analytics · k-means

1 Introduction

Modern data-driven services are a double-edged sword. On the one hand, they allow for personalized higher-quality offerings in a wide range of cases spanning from product

✉ Anastasios Gounaris
gounaria@csd.auth.gr

Georgios Sakellariou
gdsakell@csd.auth.gr

1 Department of Informatics, Aristotle University of Thessaloniki, Thessaloníki, Greece

recommendations to medicine prescriptions. Typically, such offerings are the result of advanced analytics building on top of the recent advances in the correlated fields of applied statistics, data mining and artificial intelligence. On the other hand, data-driven services may compromise privacy. Things become more complicated when cloud-based data analytics come into play to cope with issues, such as limited proprietary storage and processing capacity.

The scenarios that we target in this work are exemplified as follows. Imagine that a hospital gathers measurements from patients through any means (e.g., wearable devices, lab exam results, and so on) but lacks the IT infrastructure to permanently store and analyze them. Therefore, it decides to outsource storage and analysis to a cloud provider. The analysis consists of running the k-means clustering algorithm respecting the following two requirements: (i) no data is revealed to the cloud host; and (ii) the hospital (data owner) IT infrastructure should be allocated as low workload as possible.

Privacy issues in a cloud settings are addressed by a variety of approaches that include encryption, private information retrieval, secure hardware and intention-hiding [3]. The most relevant to our case is encryption, and more specifically, *homomorphic encryption (HE)*, according to which data are analyzed without being decrypted. In 2009, Gentry [10] provided a specific construction of a fully homomorphic scheme (FHS) that has been theoretically capable of performing arbitrary data analytics tasks. Nevertheless, such a scheme is impractical. Later improvements, such as those in [5,6,11] focused on increasing the practicality at the expense of not supporting complex operations, such as comparisons and divisions, which are both employed in k-means; other operations such as multiplications, additions and inner products do not pose any problem.

To date, there is no solution for running homomorphically k-means without putting a high burden on the client as well. The sketch of the algorithm is shown in Algorithm 1. The challenging points in HE-based implementations of k-means is in the assignment step, which involves comparison of distances, and in the update step, which involves division of sum aggregates over all point co-ordinates.

In this work, we adopt an implementation of the homomorphic scheme in [5]. This enforces us to decrypt data to perform comparison and divisions and then to re-encrypt the results. Decryption is an expensive procedure and thus cannot be performed on the data owner side. To tackle this problem, we introduce a new security model, which includes the data owner, an *honest-but-curious* server for main HE processing and an additional *honest-but-auditable* trusted server. The latter is employed for comparing the distances between the data points and the centroids. However, further issues arise.

Algorithm 1 High-level view of the k-means algorithm

```
INPUT: A set of points, $P$, the number of clusters $k$.
OUTPUT: A set of $k$ clusters,
  Choose randomly $k$ points as the centroids    ▷Initialization Step
  while not convergence AND/OR iteration limit not reached do
    Assign each point to its nearest centroid    ▷Assignment Step
    Recalculate the new centroids    ▷Update Centroids Step
  end while
```

If the trusted server learns about a single point, this suffices to compute the centers and then to extract information about the initial data. Simple solutions of perturbing the data (e.g., by adding a constant) or the the distances do not work. We address these further issues by allocating a small amount of work regarding the computation of the centroids to the data owner, which does not violate the limited processing power assumption, and by making the trusted server auditable, so that it is prohibited to use its decryption capabilities in a non-specified manner.

In summary, we make the following contributions:

1. We provide a solution for server-side k-means using HE without imposing a heavy load on the data owner. No similar solution exists to date.
2. We introduce a new security model that includes a trusted, auditable server.
3. We instantiate our framework in two different versions in terms of operation assignment each coming in three flavors of operation implementation.
4. We conduct a thorough comparison between alternatives using both real experiments and analytic cost models.

The remainder of this article is structured as follows. The next section provides background material and discusses the related work. Section 3 introduces our novel security model. The implementation details and alternatives are presented in Sect. 4. The evaluation is in Sect. 5 and we conclude in Sect. 6.

2 Background and related work

2.1 Homomorphic encryption basics

A fully homomorphic scheme (FHS) is defined as follows.

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

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

then

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

2. Secure. We assume that the scheme is semantically secure (with the usual notion in public key cryptosystems). Essentially, a scheme is semantically secure when it is infeasible for a "computationally bounded" adversary to derive significant information about a plaintext when given only its ciphertext and the corresponding public encryption key.
3. Compact. c^* is independent on the complexity of f (i.e. decrypting c^* is easier than computing f). The difference between the size of the ciphertext and the size of

the message is referred to as its ciphertext overhead. An encryption scheme is said to generate compact ciphertexts if the overhead is at most the size of one element in the underlying group.

Leveled homomorphic schemes (LHS), such as the one employed in this work, allow as many additions as but only a limited number of multiplications. For instance, we cannot make a multiplication of 20 integers, since, in that case, the noise added to the system will destroy the encrypted data.

2.2 Related work

We divide the discussion of related work into three parts: (i) cryptosystems, (ii) implementation of k-means using HE, and (iii) tools.

2.2.1 Cryptosystems

The homomorphic implementation of the k-means algorithm requires at least a leveled Full Homomorphic (FHE) encryption scheme, due to its complex nature, i.e., the fact that involves comparisons and divisions along with multiple additions and multiplications. The first FHE scheme was proposed by Gentry in his remarkable work [10]. Although innovative, Gentry's scheme has been proven inefficient for real applications. Currently, the most promising FHE schemes are based on the *Learning With Errors (LWE)* problem, and more specifically, on its special case called *Ring Learning With Errors (RLWE)*. There are two schemes that are the most relevant ones to our work, which are based on the LWE and RLWE problems, respectively: the *Brakerski-Gentry-Vaikuntanathan (BGV)* [5], implemented in HElib [23], and the RLWE variance of Brakerski's scheme [4], implemented as part of Wu's work [8,27]. In our work, we have chosen Brakerski's scheme and Wu's open source code [8] to implement our solution, due to its scale invariant property, which means that the modulo switching operation is not required whenever a homomorphic multiplication takes place; therefore a better performance of the k-means algorithm can be achieved. Nevertheless, the scheme in [8] lacks built-in support for comparisons. Homomorphic comparison on the binary level was recently added to HElib [23]. However, HElib's bit-level operations lead to added time and space complexity and performance degradation, and inevitably to a less efficient implementation of k-means.

2.2.2 Implementations of k-means using HE

The construction of a privacy-preserving or homomorphic k-means algorithm for distributed settings has already been studied in several works, but under different assumptions. Most of the prior proposals have dealt with a multi-party setting, where the data belongs to more than one party. Notable examples of such proposals include [7,20–22,26]. Moreover, these works are typically based on the Paillier's [18] cryptosystem (except [20]). As such, the Euclidean distance comparisons, a key factor of k-means, are not supported; thus these results cannot be transferred to our case.

There are also some proposals that refer to a pure client-server model, but none of them is built on top of a RLWE-based cryptosystem, as in this proposal. The work of [24] is based on the scheme in [6], and, apart from being significantly slower, involves a high number of operations on the data owner side. To provide some insights into the performance benefits from our proposal hereby, using a single machine, the approach in [24] requires more than 6 hours per iteration for a 5-dimensional dataset of 10 points for 90-bit security, whereas, in this work we can run each iteration in less than 10 min for 128-bit security for 10-dimensional datasets of 100 points. Finally, the proposals in [1,15] mitigate but not eliminate the workload of the data owner; nevertheless, they are based on a patented cryptosystem that has not been thoroughly analyzed yet [14], and consequently, no security guarantees are provided.

2.2.3 Other tools supporting encrypted data analysis

Another possible approach for a server-side homomorphic k-means is to proceed to an *SQL-based* implementation, such as the one proposed on [17], using, as the underlying database-management system (DBMS), either the MONOMI [25] or the CryptDB [19] that both execute SQL queries over encrypted data. At first sight, this approach seems promising, but an SQL implementation of k-means requires the writing of a User Defined Function (UDF). Currently, UDFs are not supported by MONOMI and are not officially supported by CryptDB (although someone can follow the same way as in CryptDB's core construction to write one). Moreover, even if the UDF's writing obstacle could be overcome in an efficient manner, none of them can fulfill the requirement for leveled FHE. Specifically, MONOMI supports only homomorphic addition based on the Paillier's [18] encryption scheme; CryptDB supports both homomorphic addition and multiplication, however these operations are provided by different encryption schemes [9,18]. In general, MONOMI and CryptDB have been designed to support only On-line Transaction Processing (OLTP) applications rather than On-line Analytical Processing (OLAP) and data mining operations as well.

3 Security model overview

A common assumption regarding privacy-preserving outsourced analytics is that the remote host is *honest-but-curious*. We partially depart from this assumption, and, in order to support cloud server-side data analytics, such as k-means, we advocate employing also another host that is *auditable*, so that it is deterred to look into the data. A naive approach is to employ only this type of cloud hosts for the complete server-side k-means computation. But (online) host auditing comes at extra cost; therefore we employ both *honest-but-curious* and *honest-and-auditable* hosts. We give the details below.

3.1 Security requirements

In order to define our security model, we first describe a use case scenario. In our scenario, Alice is the owner of a large amount of sensitive data. Alice wants to analyze her data using the k-means algorithm, but she is not able to run the analysis by her means due to storage and processing power limitations. Bob is a cloud storage and computing service provider, who offers data analytics services. Bob's storage and computational capabilities are sufficient enough to store and analyze Alice's data.

Alice desires to use the services provided by Bob to cluster her data; however, she needs an assurance that the whole process fulfills three security requirements: (i) there will be no data leakage with regards to the initial data; (ii) there will be no data leakage with regards to the k-means results; and (iii) the k-means algorithm is going to be executed without any function modifications.

3.2 Security in the proposed implementation

Let us suppose that Bob adheres to the *honest-but-curious* model, and Alice and Bob exchange encrypted messages only. Bob's honesty ensures that no functions are altered (according to the third requirement above). Assuming also SSL network connections, HE ensures in principle that the first two requirements are met as well. The problem is that, as we extensively analyze in Sect. 4.1, a homomorphic version of k-means should involve efficient computation of comparisons and divisions. But, as already mentioned, in the existing homomorphic encryption schemes, there is no efficient way to compute a comparison or a division without first decrypting the data. To deal with this issue, one approach is Bob to send the data to Alice, whenever a comparison or division operation is required during the k-means execution, subsequently Alice decrypts the data, computes the operation and she sends the results back to Bob afterwards. This approach is adopted in [24], but its main weakness is the burden of Alice with heavy processes like decryption, which contradicts to our hypothesis that Alice's system has processing limitations.

To overcome the problem above, we propose a new entity, the *trusted (or else, honest-and-auditable) server*, on which distance comparison operations can safely take place. The *trusted server* is able to decrypt ciphertexts or results coming from homomorphic operations on ciphertexts that have been produced by the private key of Alice. To this end, we capitalize on the properties of Brakerski's [4] homomorphic encryption scheme, the homomorphic operation *Switch Key* and the associated *Switch Matrix (SM)* structure, in line with the approach in [27] that is used to decrease noise. In our approach, Alice provides a private key, different of but equivalent to her own, and a *Switch Matrix* to the *trusted server*.[1] In addition, we assume that our proposed *honest-and-auditable* model applies to the *trusted server*. A distinctive feature of our proposal compared to [27] is that it leverages the switch key homomorphic operation for functionalities different from noise reduction but is also allows an entity other than Alice to decrypt messages.

For the division operations, we cannot follow the same approach, i.e., delegate their execution to the *trusted* server. If a server knows the distances from the centers (on which the comparison operations applied) and the centers themselves (for which the

[1] Someone could argue that it would be the same as providing the *trusted server* with Alice's private key. We prefer to provide the *trusted server* with an equivalent key in order to emphasize the fact the Alice and the *trusted server* are different entities.

division operation is needed), it can easily reconstruct the initial dataset that Alice wants to keep secret. However, the amount of division operators is far smaller in practice than comparisons. Therefore, we allocate the task of performing divisions to Alice, which both guarantees security and does not violate the limited processing capacity constraint on Alice.

3.3 A *honest-and-auditable* security model for remote servers

In our proposed approach, Alice is the owner of the data and the one who is willing to receive the results of k-means. Bob is the owner of the system that executes the heavy homomorphic operations according to the *honest-but-curious* model. The *trusted server* computes any comparison needed in the k-means algorithm and is *honest-and-auditable*, defined as follows.

Definition 2 An entity is considered as *honest-and-auditable*, when it belongs to the *honest-but-curious* category and any attempt to deviate from the protocol is revealed to the data owner.

In practice, this model implies that any attempt of the trusted server to decrypt data and perform actions other than comparison between specific pairs of encrypted numbers is detectable. Therefore, the server is deterred to exercise its curiosity. The model can be enforced in practice with the help of blockchain technology [16,28] for example, tailored to smart contracts for data manipulation operations, e.g., as in [29]. In such a scenario, a simple protocol could be followed so that, before the *trusted server* decrypts any ciphertext, a log entry is inserted so that any decryption action can be audited in the future. In any case, how the proposed model can be implemented in practice is out of the scope of this work.

Overall, our approach can be considered as a special case of the secure multiparty computation paradigm [13]. By definition, our approach is classified as semi-honest (passive) solution, regarding the total security of the system [13]. In bibliography, a similar model to *honest-and-auditable* is that of *covert adversary* [2]. *Covert adversary* is a threat model that is placed between *honest-but-curious (passive)* and *malicious (active)* models. A *covert adversary* is willing to cheat during the execution of multiparty computation protocol, but only in case that cheating remains undetectable. Protocols that protect against covert adversaries ensure that any cheating attempt will be immediately revealed and all the participants will be informed about. The main difference of *honest-and-auditable* and *covert adversary* models is that the *covert adversary* model assumes that the participants are *equally untrusted*, while the *honest-and-auditable* model assumes the existence of at least one honest party; this derives from the fact that the former assumes a setting, in which data is owned by multiple entities.

4 Our solutions

In this section, we provide implementation details and alternative techniques differing in the way the entities of our model interact with each other.

Table 1 Main notation in three groups

Symbol (data-related)	Explanation
$n \in \mathbb{Z}^+$	Number of points
$d \in \mathbb{Z}^+$	Point dimensionality
$P = \{p_1, p_2, \ldots, p_n\}$	Set of points
$p_i = (a_1^i, a_2^i, \ldots, a_d^i) \in \mathbb{Z}^d$	ith point

Symbol (k-means-related)	Explanation
$k \in \mathbb{Z}^+$	The number of clusters(centers) in k-means
A, A'	$n \times k$ binary matrices where $A(i, y) = 1$ if $p_i \in y$th cluster

Symbol (encryption-related)	Explanation
(P_k, S_k)	The pair of public and secret key of Alice
SM_N	The noise reduction switch matrix used after homomorphic multiplication sent to Bob
(S_k^M, SM)	The pair of the secret key and the switch matrix sent to the trusted server
$Q_1 = \{q_1(x), q_2(x), \ldots, q_n(x)\} \in \mathbb{Z}[x]/\langle r\rangle$ with r prime, $\langle r\rangle$ the corresponding prime ideal, and $r \gg a_j$	Set of polynomials for the 1st representation method
$Q_2 = \{q_1^1, \ldots, q_d^1, \ldots, q_1^n, \ldots, q_d^n\} \in \mathbb{Z}[x]/\langle r\rangle$ as above, and q_j^i, $j = 1, 2, \ldots, d$ integers	Set of polynomials for the 2nd representation method
$E_{P_k}(q_i) = c_i$ (resp. $D_{S_k}(c_i) = q_i$)	Encryption (resp. decryption) of $q_i \in Q_1$
$E_{P_k}(q_j^i) = c_i$ (resp. $D_{S_k}(c_j^i) = q_i$)	Encryption (resp. decryption) of $q_j^i \in Q_2$
$E_{P_k}(vector(q_j^i)) = vector(c_j^i)$ (resp. $D_{S_k}(vector(c_j^i)) = vector(q_j^i)$)	Encryption (resp. decryption of $vector(q_j^i) \in Q_2^d$)

4.1 Implementation issues and approach overview

Table 1 summarizes the notation that we shall use.[2] We assume familiarity with the basic concepts of the k-means algorithm and the concepts of polynomials with integer coefficients, rings and finite fields in mathematics.

4.1.1 Point representation and distance computation

The adopted homomorphic encryption scheme from [8] works with polynomials with integer coefficients. There are two methods to represent points as such polynomials. The first method is to directly represent each point, $p_i \in P$, to one polynomial, $q_i \in Q_1$. Namely, we define $f : P \rightarrow Q_1$ such that $f(p_i) = q_i(x) = a_1 + a_2x + \cdots + a_dx^{d-1}$. The second method is to represent each coefficient of a point with a

[2] The term i is an index and not a power. In this paper, we always use parenthesis, e.g. $(t)^2$, to show a power of a number.

zero degree polynomial, i.e., as an integer. Namely, we define $g : P \rightarrow Q_2$ such that $g(p_i) = vector(q^i_j) = (a^i_1, \ldots, a^i_d)$.

A main operation of k-means is the computation of the distance between a point and a centroid, which is another point. To this end, a variety of metrics can be used. In our work, we study and homomorphically implement the Euclidean and Manhattan distances. In the $\mathbb{Z}^d$, the Euclidean distance between points p_1 and p_2 is $d_E(p_1, p_2) = \sqrt{\sum_{j=1}^{d}(a^1_j - a^2_j)^2}$. One minor issue of this formula is the existence of the square root, which cannot be homomorphically implemented, but due to monotonic property of square root, the comparison result of two or more distances is not affected by the existence of the square root. Therefore, we can omit the square root or, in other words, to use the square of the Euclidean distance during k-means execution. So, the homomorphic form of the (squared) Euclidean distance is defined as:

$$E_{P_k}((d_E(p_1, p_2))^2) = d_E(vector(c^1_j), vector(c^2_j)) = \sum_{j=1}^{d}(c^1_j - c^2_j)(c^1_j - c^2_j)$$

It is obvious that the implementation of the Euclidean distance requires the input data to have a form that follows the second representation method from Table 1.

The Manhattan distance in $\mathbb{Z}^d$ is computed by the formula $d_M(p_1, p_2) = \sum_{j=1}^{d} \mid a^1_j - a^2_j \mid$. The homomorphic version of Manhattan distance can be implemented with the input data represented using either the first or the second representation method. In the former case, the homomorphic formula of Manhattan distance is

$$E_{P_k}(d_M(p_1, p_2)) = d_M(c_1, c_2) = \mid c_1 - c_2 \mid = \mid c_1 + (-1 \cdot c_2) \mid$$

The result of this formula is an encrypted polynomial $q_M \in \mathbb{Z}[x]/r$, $deg(q_M) = d - 1$, from which we extract the distance using the Algorithm 2. In the later case, the Manhattan formula takes the form

$$E_{P_k}(d_M(p_1, p_2)) = d_M(vector(c^1_j), vector(c^2_j)) = vector(c^1_j - c^2_j) = vector(c^M_j)$$

and the actual distance can be extracted from the encrypted data with a similar algorithm that iterates over the coefficients of $vector(a^M_j)$ instead of q_M.

The cost model of the distance computation alternatives is presented to Table 2, assuming that $\delta_1, \delta_2, \delta_3$ and δ_4 are the costs of homomorphic multiplication(M), scalar multiplication (ScM), addition (A) and decryption (D) respectively.

Algorithm 2 Manhattan distance extraction for the 1st point representation method

```
$d_M = 0$
for all coefficients $a$ of $q_M$ do
  if $a > r/2$ then
    $a = a - r$
  end if
  $d_M += a$
end for
```

Table 2 Homomorphic Euclidean and Manhattan distances cost model

Distance	M	ScM	A	D	Cost
d_M (1st method)	–	1	1	1	$\delta_2 + \delta_3 + \delta_4$
d_M (2nd method)	–	d	d	d	$d(\delta_2 + \delta_3 + \delta_4)$
d_E	d	$2d$	$3d - 1$	1	$d\delta_1 + 2d\delta_2 + (3d - 1)\delta_3 + \delta_4$

4.1.2 Comparisons on the trusted server

In k-means, a homomorphic version of the comparison operation is required for the discovery of the minimum distance while assigning points to clusters. As mentioned is Sect. 2.2.1, the selected encryption scheme does not support homomorphic comparison and therefore, the distance comparison is provided by a *trusted entity* (Sect. 3.2), using the *Switch Key* operation. Specifically, the *trusted server* is equipped with a pair (S_k^M, SM), where the S_k^M is randomly produced and is equivalent to Alice's secret key. $SM = SwitchKeyGen(S_k, S_k^M)$ is produced with the help of the $SwitchKeyGen$ functionality described in [27]. This allows the trusted server to decrypt any message encrypted by Alice. To decrypt a ciphertext c encrypted by P_k, it first produces a new ciphertext $c_M = SwitchKey(SM, c)$ and then computes $p = D_{S_k^M}(c_M)$. After the decryption of the distances, the *trusted server* compares the Euclidean distances as ordinary numbers, while in the case of Manhattan distances, it first applies the Algorithm 2 and then continues normally.

4.1.3 Putting everything together

The abstract k-means algorithm consists of (i) the Initialization step; (ii) the Assignment step; and (iii) the Centroid Update step. Steps (ii) and (iii) are executed a predefined number of iterations.[3] Algorithm 3 depicts the abstract idea of our homomorphic implementation of k-means. The key points of our implementation idea are described as follows:

– Alice sends to Bob her data encrypted with her public key—Step (i).
– Alice constructs a second private key and a switch matrix, that relates her private key with the second private key, and sends them to the *trusted server*—Step(i)
– Bob cooperates with the *trusted server* in order to execute the assignment step of the k-means algorithm—Step(ii).
– In each iteration, Bob sends the sum of the coordinates and the amount of points per cluster to Alice, who can then compute the new centroids—Step(iii).

[3] It is straightforward to consider flavors where k-means terminates when the centroids have converged as well.

Algorithm 3 Homomorphic K-means

```
INPUT: $P, k, iter$ \\$iter$ is the maximum number of iterations
OUTPUT: A set of clusters
  **Initialization Step**
  $C = E_{P_k}(P)$ \\$C$: the encrypted points
  Choose randomly $k$ encrypted points and set $centoid_j = c_i, i = 1, .., k$
  Initialize $A = A' = 0$
  $round = 0$
  while $round < iter$ do
    **Assignment Step - Bob and trusted server collaborate**
    for all $c_i \in C$ do
      for all $centroid_j$ do
        $A'(i, j') = \{1 | j' = \arg\min_{j=1...k}(D_{S_k^{SM}}(SwitchKey(SM, d_{(M|E)}(c_i, centrod_j))))\}$
      end for
    end for
    $round = round + 1$
    $A = A'$
    $A'(i, j) = 0 \; \forall i, j \; with \; j > 1$
    **Centroid Update Step - Alice and Bob collaborate**
    for $j = 1$ to $k$ do
      $centroid_j = \frac{D_{S_k}(\sum_{i=1}^{n}(c_i | A(i,j)=1)))}{\sum_{i=1}^{n}(1 | A(i,j)=1)}$
      $centroid_j = E_{P_k}(centroid_j)$
    end for
  end while
```

4.2 Technique I—Bob knows the cluster assignment of unknown points

In Technique I, before k-means starts execution, Alice generates the cryptographic elements P_k, S_k, SM_N, S_k^T, SM; then, she sends the P_k and a noise reduction SM_N to Bob and the pair (S_k^T, SM) to the *trusted server*. Finally, Alice encrypts the data points and sends them to Bob.

Afterwards, the k-means algorithm is initialized by Bob, who selects randomly k centroids from the existing points. During the execution of k-means, in the Assignment step, Bob computes k distances for each point - between the point and the centroids - which are sent to *trusted server*. Then, Bob receives back from the *trusted server* the cluster index where each point is classified in the form of the A matrix. Upon receiving the response, Bob proceeds on to the Update step, where he computes the sum of the coordinates and the amount of points for each cluster to Alice. Alice can then compute the new centroids and send them back to Bob in an encrypted form. If the termination condition is not met, i.e., if the predefined number of operations has not been reached, this process repeats. The interactions are shown in Fig. 1.

In this technique, the main responsibility of the *trusted server* is the computation of the cluster index to which a point belongs, having done a comparison of the k distances that has been sent by Bob.

This technique can be instantiated in three flavors, each corresponding to one of the three alternatives explained in Sect. 4.1.1. A final technical point is that, during homomorphic multiplications, the ciphertexts can grow very large. To mitigate this

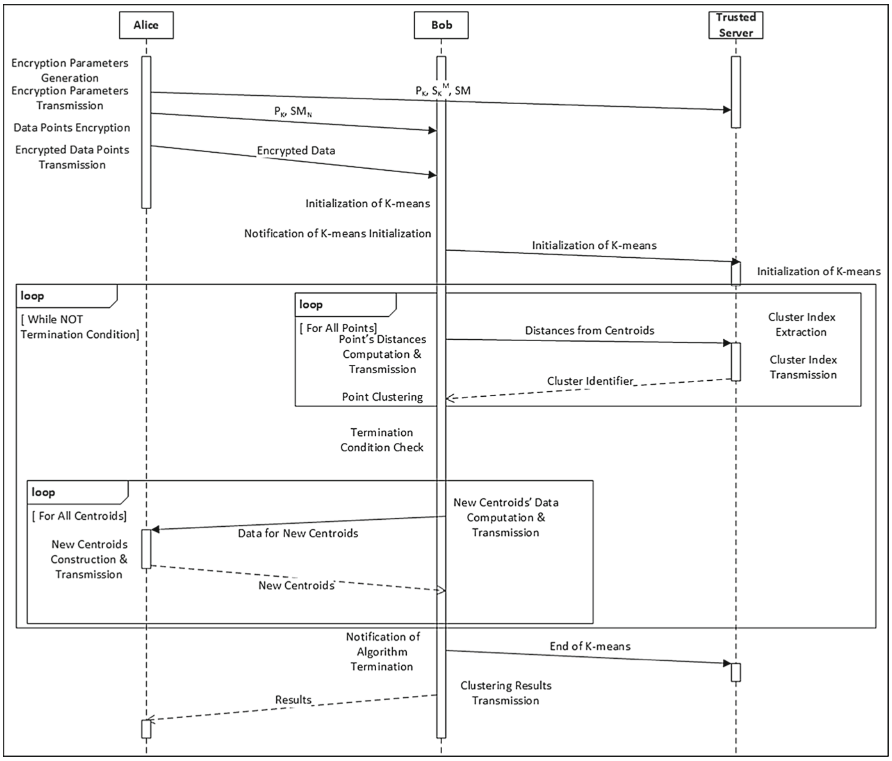

Alice
Bob
Trusted Server
Encryption Parameters Generation
Encryption Parameters Transmission
P_K, S_K^M, SM
P_K, SM_N
Data Points Encryption
Encrypted Data Points Transmission
Encrypted Data
Initialization of K-means
Notification of K-means Initialization
Initialization of K-means
Initialization of K-means
loop
[While NOT Termination Condition]
loop
[For All Points]
Point's Distances Computation & Transmission
Distances from Centroids
Cluster Index Extraction
Cluster Index Transmission
Cluster Identifier
Point Clustering
Termination Condition Check
loop
[For All Centroids]
New Centroids' Data Computation & Transmission
Data for New Centroids
New Centroids Construction & Transmission
New Centroids
Notification of Algorithm Termination
End of K-means
Clustering Results Transmission
Results

Fig. 1 The sequential diagram of Technique I

problem, Bob employs its noise reduction SM in order to bring the resulted ciphertexts to their initial length.

4.3 Technique II—Bob does not know anything

In the previous technique, the *trusted server* sends the matrix A to Bob in an non-encrypted format. This results into a limited information leakage, since Bob becomes aware of the cluster sizes and the assignment of point ids to these clusters. This technique addresses this limitation and does not allow Bob to learn anything about the algorithm's result. To fulfill this, the *trusted server* sends back A row-by-row in an encrypted manner. The implication is that Bob cannot perform a scalar multiplication to compute the sums of coordinates and simple additions to compute the size of each cluster; instead he has to perform inner products between vectors of size n and homomorphic additions, respectively (see Fig. 2).

4.4 Analysis

The analysis of the techniques has a twofold purpose: firstly, to extensively analyze the homomorphic operations that take place, and secondly, to theoretically compare

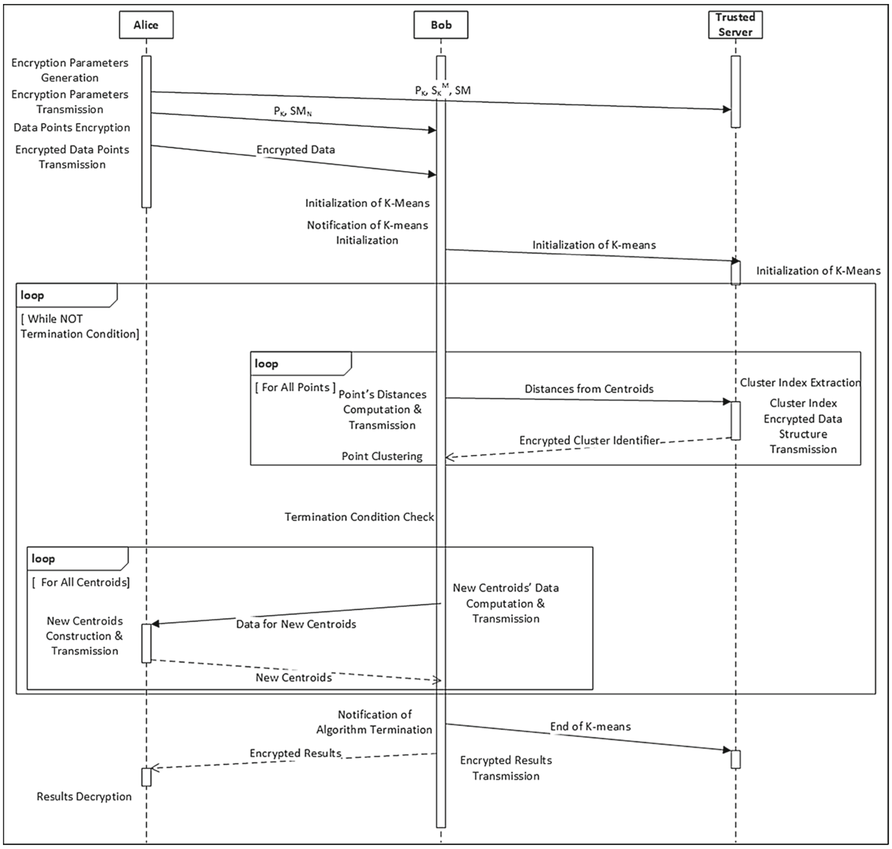

Alice
Bob
Trusted Server
Encryption Parameters Generation
Encryption Parameters Transmission
P_K, S_K^M, SM
P_K, SM_N
Data Points Encryption
Encrypted Data Points Transmission
Encrypted Data
Initialization of K-Means
Notification of K-means Initialization
Initialization of K-means
Initialization of K-Means
loop
[While NOT Termination Condition]
loop
[For All Points]
Point's Distances Computation & Transmission
Distances from Centroids
Cluster Index Extraction
Cluster Index Encrypted Data Structure Transmission
Encrypted Cluster Identifier
Point Clustering
Termination Condition Check
loop
[For All Centroids]
New Centroids' Data Computation & Transmission
New Centroids Construction & Transmission
Data for New Centroids
New Centroids
Notification of Algorithm Termination
End of K-means
Encrypted Results
Encrypted Results Transmission
Results Decryption

Fig. 2 The sequential diagram of Technique II

the performance and complexity of the two techniques. The analysis of techniques in this section is both orthogonal and an extension to the analysis of the distance metric and point representation summarized in Table 2.

Tehcniques I and II differ in the way (i) the sums of each centroid coordinate and (ii) the cluster sizes need to be computed. For example, if the second representation method is used, in each iteration of Technique I, nd homomorphic additions required for the computation of the sums of centroid coordinates and no homomorphic operations are needed for the cluster sizes. In Technique II, ndk homomorphic multiplications and $(n-1)dk$ additions are needed for the sums of the coordinates and nk additions for the cluster sizes.

The theoretical performance analysis of algorithms with homomorphic operations using a standard algorithmic time complexity approach needs to be performed with care. The reason behind that is the variance in complexity of the fundamental operations of homomorphic encryption schemes, which implies that we cannot directly link the algorithm's parts with the amount of input data and to simply add the intermediate results in a meaningful manner. In our case, the encryption and decryption operations have a crucial role, which, together with the involved data transmissions, increases the actual complexity.

Below, we present a detailed analysis assuming $iter$ execution rounds focusing on (a) the generation and transmission of encryption elements, (b) the encryption operations, (c) the decryption operations, (d) the transmissions of encrypted data, and (e) the homomorphic operations of homomorphic (scalar) multiplications, additions and SM ciphertext transformations[4].

We start the analysis with Technique I:

(a) Alice produces (5) cryptographic elements: the pairs (P_k, S_k), (S_k^M, SM) and a noise reduction SM_N. Bob receives the noise reduction SM_N along with P_k. The *trusted server* receives the (S_k^M, SM) pair along with P_k. This process is not affected by the different choices regarding the distance computation and the point representation.

(b) The number of encryption operations depends on the point representation choice. In the first method, which supports only the Manhattan distance, there is a single encryption operation per point, whereas, in the second method, there is a single encryption operation per point co-ordinate (dimension). Alice encrypts the data points only once at the beginning. She also encrypts the new centroids in each iteration apart from the first and the last ones; in the first iteration, the centroids are some random points and in the last iteration, the centroids need not be sent back to Bob. Overall, in the first representation method, $n + k(iter - 2)$ encryption operations are performed; and in the second representation method, the operations are d times more.

(c) The *trusted server* executes $nk(iter)$ (resp. $nkd(iter)$) decryption operations, which are involved in distance comparison, while Alice performs $k(iter)$ (resp. $kd(iter)$) decryption operations involved in the new centroids construction when the points are represented according to the 1st (resp. 2nd method).

(d) In the 1st representation method, where there is a single ciphertext per data point, Alice transmits n encrypted data items at the beginning, Bob transmits $nk(iter)$ encrypted data items related to the distances from the centroids and $k(iter)$ items related to new centroid computation. Alice sends back the centroids in an encrypted form when the iteration limit has not been reached; i.e., this amounts to additional $k(iter - 2)$ items. In the second representation method, in case of d_E all the above quantities except the distances are multiplied by d, in case of d_M there is no exception.

(e) Regarding the homomorphic operations, we have to distinguish between both the representation and the distance types. For the first representation type of d_M, no multiplications of ciphertexts are executed. Bob executes $nk(iter)$ scalar multiplications and $nk(iter)$ additions for the computation of distances, and $(n-k)(iter)$ additions for the new centroids construction. The *trusted server* executes $nk(iter)$ switch key operations during the comparison of distances. In case of d_M com-

[4] As we have already discussed, there are two kinds of SM transformations. The first one uses the SM_N, executed by Bob for noise reduction purposes after a homomorphic multiplication, while the second one executed by the *trusted server* using the SM for switch key purposes. However, there is no difference between them from the perspective of theoretical performance analysis, so in summarization tables we refer to them together.

puted on points represented with the second representation method, the above are multiplied by d. When d_E is employed, Bob executes $ndk(iter)$ multiplications, $2ndk(iter)$ scalar multiplications and $(3d-1)nk(iter)$ additions for the distance computations. Bob, also executes $(n-k)d(iter)$ additions for the sums of co-ordinates. Finally, $ndk(iter) + nk(iter)$ switch key operations are performed, from which $ndk(iter)$ are executed by Bob and $nk(iter)$ are executed by the *trusted server*.

Regarding Technique II, the following statements hold:

(a) The analysis of encryption elements generation and transmission is similar to that in Technique I.
(b) Alice produces the same ciphertexts like in Technique 1. The *trusted server* executes $nk(iter)$ encryption operations during the point's clustering regardless of the representation method.
(c) The decryption operations in Technique II are same as in Technique I. The only difference is that Alice performs one more decryption per centroid per iteration regarding the number of points in each cluster.
(d) The additional transmissions are: (i) $nk(iter)$ for the encrypted A matrix contents sent from the *trusted server* to Bob, (ii) an additional item per centroid per iteration sent by Bob to Alice for the cluster size and (iii) nk for the encrypted A result matrix, that Bob sends to Alice.
(e) For the first representation type of d_M, Bob executes $nk(iter)$ scalar multiplications and $nk(iter)$ additions during the computation of distances. To compute the sums of the coordinates, Bob performs $nk(iter)$ multiplications between ciphertexts and $(n-1)k(iter)$ additions, whereas additional $(n-1)k(iter)$ additions are required for the computation of the cluster sizes. The *trusted server* executes $nk(iter)$ switch key operations during the comparison of distances, and Bob executes also $nk(iter)$ SM operations. In case of d_M computed on points represented with the second representation method, all the above are multiplied by d except of additions related to cluster sizes computation. When d_E is employed, Bob executes $ndk(iter)$ multiplications, $2ndk(iter)$ scalar multiplications and $(3d-1)nk(iter)$ additions for the distance computations. Bob, also executes $ndk(iter)$ multiplications and $(n-1)dk(iter) + (n-1)k(iter)$ additions for the sums of co-ordinates and the size of clusters. Finally, $2ndk(iter) + nk(iter)$ switch key operations are performed, from which $2ndk(iter)$ are executed by Bob and $nk(iter)$ are executed by the *trusted server*.

Table 3 summarizes the detailed cost model for each alternative considered.

5 Experimental evaluation

We split the evaluation in three parts: (i) performance experiments; (ii) cost model validation; and (iii) workload analysis per entity.

Table 3 Detailed cost model of alternatives

Technique I			
Flavor	1	2	3
Distance Metric	d_M (1st method)	d_M (2nd method)	d_E
generate/transmit encr. elements	5/5	5/5	5/5
encryptions	$n + k(iter - 2)$	$nd + kd(iter - 2)$	$nd + kd(iter - 2)$
decryptions	$(n + 1)k(iter)$	$(n + 1)kd(iter)$	$(n + 1)kd(iter)$
transmission of encrypted data	$n + nk(iter) + 2k(iter - 1)$	$nd + ndk(iter) + 2dk(iter - 1)$	$nd + nk(iter) + 2kd(iter - 1)$
Homomorphic Multiplications	$-$	$-$	$ndk(iter)$
Scalar Hom/phic Multiplications	$nk(iter)$	$dnk(iter)$	$2ndk(iter)$
Homomorphic Additions	$(nk + n - k)(iter)$	$(nk + n - k)d(iter)$	$((3d - 1)nk + (n - k)d)(iter)$
SM operations	$nk(iter)$	$dnk(iter)$	$(d + 1)nk(iter)$
Technique II			
Flavor	1	2	3
Distance Metric	d_M (1st method)	d_M (2nd method)	d_E
generate/transmit encr. elements	5/5	5/5	5/5
encryptions	$n + k(iter - 2) + nk(iter)$	$nd + kd(iter - 2) + nk(iter)$	$nd + kd(iter - 2) + nk(iter)$
decryptions	$(n + 2)k(iter)$	$(nd + d + 1)k(iter)$	$(nd + d + 1)k(iter)$
transmission of encrypted data	$n + nk(2iter + 1) + k(2iter - 1)$	$nd + nk(d + 1)(iter) + 2dk(iter - 1) + k(n + iter)$	$nd + nk(2iter + 1) + 2kd(iter - 1) + k(iter)$
Homomorphic Multiplications	$nk(iter)$	$ndk(iter)$	$2ndk(iter)$
Scalar Hom/phic Multiplications	$nk(iter)$	$ndk(iter)$	$2ndk(iter)$
Homomorphic Additions	$(3n - 2)k(iter)$	$dnk(iter) + (2nd - d + n - 1)k(iter)$	$(3d - 1)nk(iter) + (n - 1)(d + 1)k(iter)$
SM operations	$2nk(iter)$	$2ndk(iter)$	$2ndk(iter) + nk(iter)$

5.1 Setting

For all the experiments, a machine with 4-core Intel i7-4800MQ processor, 6MB cache and 8 GB memory was used. The operating system was Linux Ubuntu 16.04 LTS with a `gcc` version 5.4.0 compiler. Moreover, the core library of our implementation is a modified version of the code developed on [27]. All the code is available on https://github.com/HomEncKmeans, in three different projects, where each one contains the code related to Alice, Bob and *trusted server*, respectively.

In cryptography, a *128-bit* security level is considered strong. Seeking to reach a *128-bit* security level without the noise of ciphertexts to affect the validity of the encryption scheme, we follow the approach of [27] and [12] to determine the encryption parameters of the scheme. See the document in the code repository for how the encryption parameters are set to provide guarantees for *128-bit* security level.

5.2 Experiments

During the experimental evaluation, six randomly created data sets were used. The corresponding ($n \times d$) dimensions are 20×2, 20×5, 20×10, 100×2, 100×5 and 100×10, respectively.

The times we present refer to a setting where $iter = 5$ and $k = 3$ and there is no communication or SSL connection cost. More specifically, all three entities, namely Alice, Bob and the trusted server are collocated on the same machine and we report end-to-end times.

5.2.1 Performance results

In Fig. 3, the experimental evaluation of each technique flavor is illustrated. The key observations are summarized below:

1. It takes more than 43 min to make a meaningful progress of 5 iterations in k-means when processing a 10-dimensional dataset of only 100 points on a modern processor. This provides strong insights into how compute-intensive homomorphically enctrypted clustering remains.
2. As we expect from the theoretical analysis, which is summarized in Table 3, the performance of the first flavor (referring to representing points as a single polynomial and using the Manhattan distance) is independent of the data dimensionality.
3. When using the second representation method, the difference in performance when using Manhattan or (squared) Euclidean distance is small (if not negligible).
4. The times of flavors 2 and 3 (i.e., the second representation method) grow linearly in the number of dimensions.
5. As shown more clearly in Fig. 4, all flavors grow linearly in the number of data points.

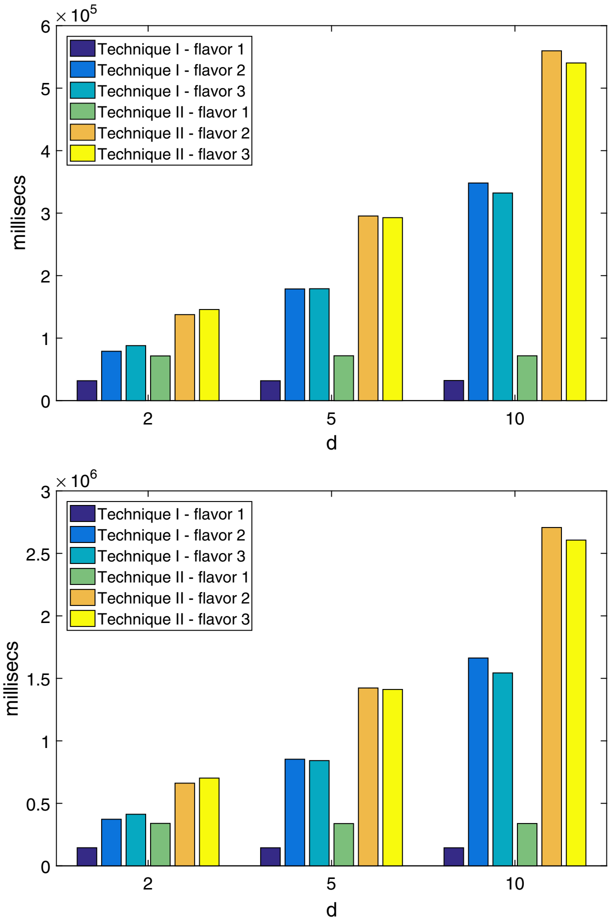


Fig. 3 Experimental evaluation of techniques ($n = 20$ (top), $n = 100$ (bottom))

5.2.2 Cost model validation

Next, seeking to compare the theoretical cost model of each technique against the real measurements, we evaluate the validity of the cost model using the Table 4 as the parameter cost matrix.

In Fig. 5, the comparison of the analytical cost model and the real measurements is illustrated. As, we can observe, in general, there is an underestimation of the cost regarding the real measurements, but the bar graphs of the cost models have the same shape with those of the real measurements when examined per technique flavor. When comparing across flavors, we see that the model does not capture well the extra processing needed in the second flavor due to absolute value operation and the increased amount of encrypted data transmissions. Thus we can infer that the differences are mostly attributed to the cost matrix parameters (i.e., due to the inherent difficulty to precisely measure the time cost of each operation presented in cost matrix),

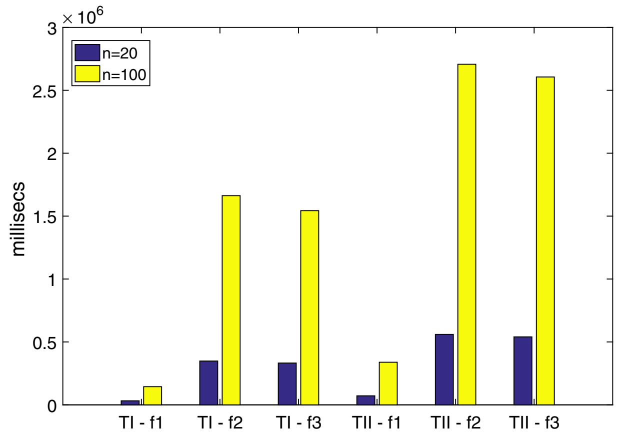


Fig. 4 Experimental evaluation of techniques ($d = 10$)

Table 4 Cost matrix of elementary operations

Operation	Time (s)
Generation and transmission of encryption elements	1.012
Encryption operations	0.023
Decryption operations	0.014
Transmissions of encrypted data	0.0006
Transmissions of encrypted data	0.00001
Homomorphic operations	
M	0.04
ScM	0.001
A	0.001
SM	0.04

the communication cost of control parameters during the algorithm's execution, and the extra processing in the second flavor, which has not been modelled.

Table 5 drills down on the most common case of using k-means, which employs the Euclidean distance. We can observe that the cost model error decreases as the number of points and/or the dimensions increases. For $n = 100,\ d = 10$ the relative error is 17.5% for Technique II, which is not particularly high. For Technique I it becomes negligible.

Overall, the most important lesson learnt is that our solution is particularly computation intensive on the one hand, but, on the other hand, it adheres to the $O(n \cdot d)$ complexity as its non-encrypted version, while the analysis in Table 3 offers a good estimate of the expected time cost for Technique II and is nearly accurate for Technique I.

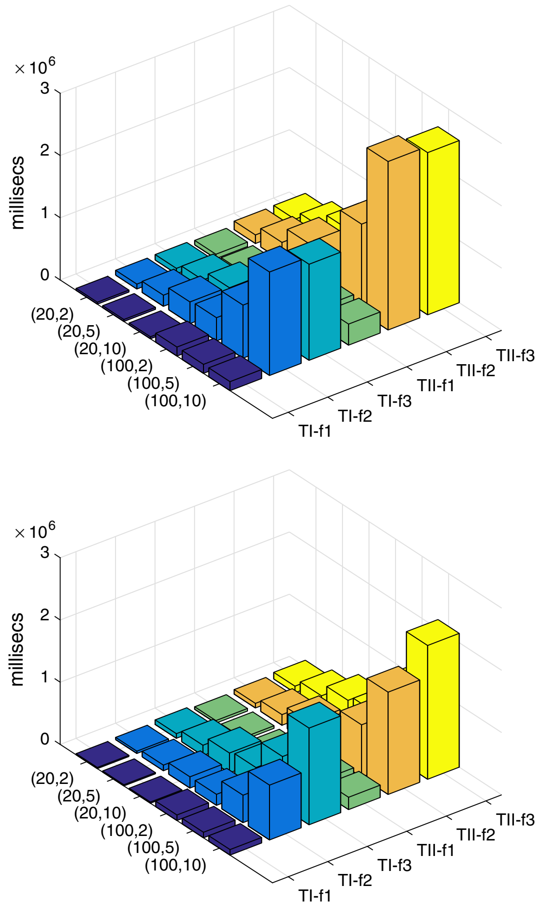


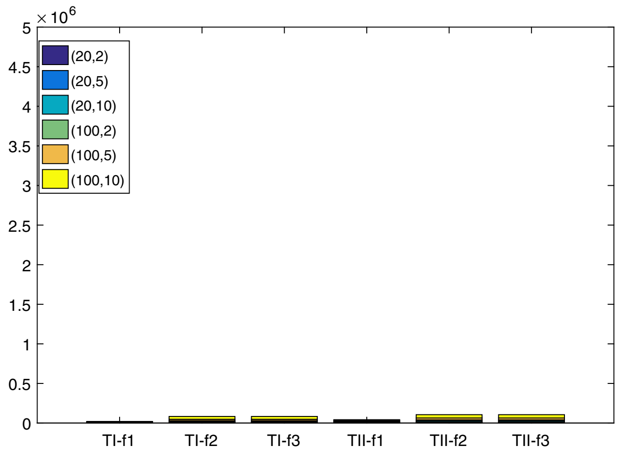
Fig. 5 Comparison of real (top) and theoretical model (bottom) cost

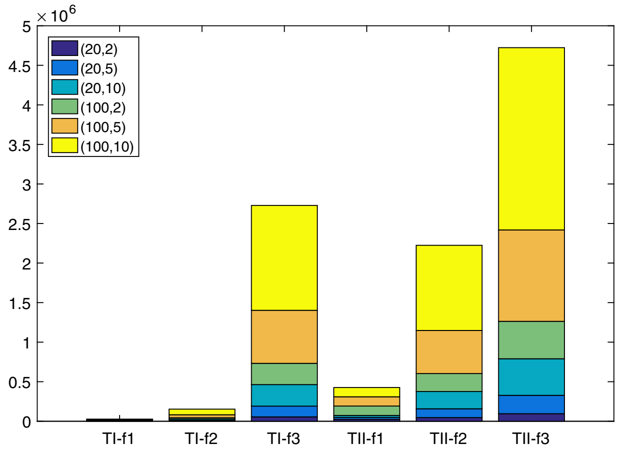
Table 5 The relative error when using the squared Euclidean distance

n	*d*	TI relative error (%)	TII relative error (%)
20	2	15.7	26.91
20	5	7.22	21.63
20	10	3.87	19.62
100	2	11.89	25.28
100	5	2.81	19.64
100	10	0.65	17.5

5.2.3 Data owner and server workload

In this final part of the evaluation, we aim to show the contribution of each entity workload to the total execution cost. Figure 6 presents the cumulative CPU time usage of Alice, Bob and *trusted server* for each technique flavor. Specifically, each column





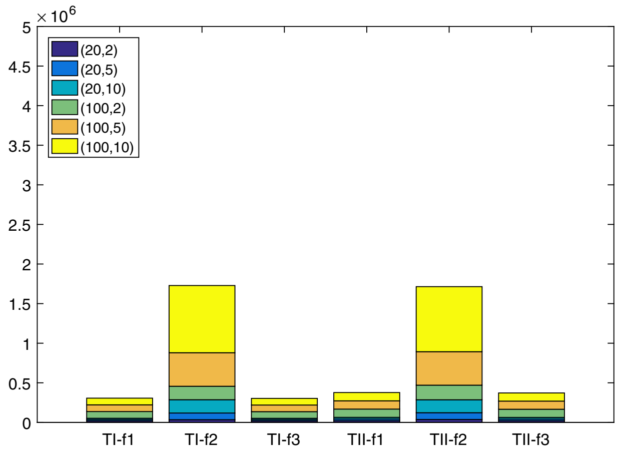


Fig. 6 The workload per entity; Alice (top), Bob (middle), trusted server (bottom)

Table 6 The contribution of Alice to the total cost when using the squared Euclidean distance

n	d	TI (%)	TII (%)
20	2	4.08	3.73
20	5	4.03	3.00
20	10	4.00	2.69
100	2	2.16	2.36
100	5	2.36	1.90
100	10	2.47	1.71

represents the cumulative CPU time usage in all conducted experiments. Based on this figure, we can infer that Alice has the minimum burden during the execution of k-means in comparison to Bob and the *trusted server*.

Table 6 provides exact numbers for the third flavor, which employs the squared Euclidean distance. In such cases, the contribution of Alice does not exceed 4.08%, and is less than 2% for the largest dataset for Technique II.

Alice's overhead provides also insights into the maximum speedup that can be achieved if we employ massive parallelism on the server side. The largest dataset requires more than 43 min using Technique II, as already stated. Even if we manage to make the server cost negligible, due to Amdahl's law, the maximum speed-up would be 58X and the time cost approximately 45 s due to Alice's overhead.

6 Conclusions

During the past decade, the advancement in the field of homomorphic encryption has provided data scientists with a rich arsenal of theoretical and practical tools. Despite this evolution, there are still many theoretical obstacles that prevent the development of homomorphic versions of most data analysis algorithms. However, the demand of privacy preserving solutions on data-driven services highlights the necessity of homomorphic algorithms in data analytics.

In this paper, we proposed and developed a solution of server-side k-means using the current technological state-of-the-art in HE. To cope with the theoretical issues that arise from the homomorphic implementation of complex algorithms, we introduce a new security model, which has, as cornerstone, a trusted, auditable server.

We implement the proposed framework in two versions with different security standards, in three flavors each, to thoroughly test and compare the performance. A twofold performance analysis has taken place. The theoretical part, based on a cost model, provides a detailed cost analysis of our framework, which clearly reflects the impact of the point representation method and the distance metric on the total performance of the framework. The experimental part provides a real comparison between the alternatives and proves the validity of the theoretical part's estimations. Moreover, it indicates that the proposed framework keeps the heavy load of homomorphic operations to a minimum level on the side of the data owner.

In conclusion, given the theoretical obstacles in HE, like the implementation of comparison and division operations, the proposed framework provides both an efficient solution to the problem of efficiently computing *k*-means on cloud hosts and an alternative base for the implementation of other complex algorithms of data analytics. The latter is due to the fact that the proposed framework can cover additional analytics by offloading complex operations to a cloud-hosted auditable server.

Acknowledgements We would like to thank Dr. K. Draziotis for all his help and advice on cryptographic issues.

References

1. Almutairi N, Coenen F, Dures K (2017) K-means clustering using homomorphic encryption and an updatable distance matrix: secure third party data clustering with limited data owner interaction. Springer, Cham, pp 274–285
2. Aumann Y, Lindell Y (2010) Security against covert adversaries: efficient protocols for realistic adversaries. J Cryptol 23(2):281–343
3. Barhamgi M, Bandara AK, Yijun Y, Belhajjame K, Nuseibeh B (2016) Protecting privacy in the cloud: current practices, future directions. IEEE Comput 49(2):68–72
4. Brakerski Z (2012) Fully homomorphic encryption without modulus switching from classical GapSVP. In: Lecture notes in computer science (including subseries Lecture notes in artificial intelligence and lecture notes in bioinformatics)
5. Brakerski Z, Gentry C, Vaikuntanathan V (2012) Fully homomorphic encryption without bootstrapping. In: Innovations in theoretical computer science
6. Brakerski Z, Vaikuntanathan V (2011) Efficient fully homomorphic encryption from (standard) LWE. In: 2011 IEEE 52nd annual symposium on foundations of computer science. IEEE, pp 97–106
7. Bunn P, Ostrovsky R (2007) Secure two-party k-means clustering. In: Proceedings of the 14th ACM conference on Computer and communications security—CCS '07, New York, New York, USA. ACM Press, p 486
8. David W (2017) FHE-SI: Implementation of Brakerski's leveled homomorphic encryption system. https://github.com/dwu4/fhe-si. Accessed 11 Aug 2018
9. Elgamal T (1985) A public key cryptosystem and a signature scheme based on discrete logarithms. IEEE Trans Inf Theory 31(4):469–472
10. Gentry C (2009) Fully homomorphic encryption using ideal lattices. In: Proceedings of the 41st annual ACM symposium on symposium on theory of computing—STOC '09
11. Gentry C, Halevi S (2011) Implementing gentry's fully-homomorphic encryption scheme. In: Lecture notes in computer science (including subseries Lecture notes in artificial intelligence and lecture notes in bioinformatics)
12. Gentry C, Halevi S, Smart NP (2012) Homomorphic evaluation of the AES circuit. In: Lecture notes in computer science (including subseries Lecture notes in artificial intelligence and lecture notes in bioinformatics)
13. Lindell Y, Pinkas B (2009) Secure multiparty computation for privacy-preserving data mining. J Priv Confid 1(1):59–98
14. Liu D (2013) International Patent Application No.: PCT/AU2013/000674. Accessible via http://patentscope.wipo.int/search/en/WO2013188929. Accessed 11 Aug 2018
15. Liu D, Bertino E, Yi X (2014) Privacy of outsourced k-means clustering. In: 9th ACM symposium on information, computer and communications security, ASIA CCS '14, Kyoto, Japan, 03–06 June 2014, pp 123–134
16. Nakamoto S (2008) Bitcoin: a peer-to-peer electronic cash system. www.Bitcoin.org. Accessed 11 Aug 2018
17. Ordonez C (2006) Integrating K-means clustering with a relational DBMS using SQL. IEEE Trans Knowl Data Eng 18(2):188–201

18. Paillier P (1999) Public-key cryptosystems based on composite degree residuosity classes. In: Lecture notes in computer science (including subseries Lecture notes in artificial intelligence and lecture notes in bioinformatics)
19. Popa RA, Redfield CMS, Zeldovich N, Balakrishnan H (2011) Cryptdb: protecting confidentiality with encrypted query processing. In: Proceedings of the 23rd ACM symposium on operating systems principles 2011, SOSP 2011, Cascais, Portugal, 23–26 October 2011, pp 85–100
20. Rao F-Y, Samanthula BK, Bertino E, Yi X, Liu D (2015) Privacy-preserving and outsourced multi-user k-means clustering. In: IEEE conference on collaboration and internet computing, CIC 2015, Hangzhou, China, 27–30 October 2015, pp 80–89
21. Samet S, Miri A (2008) Privacy preserving ID3 using Gini index over horizontally partitioned data. In: 2008 IEEE/ACS international conference on computer systems and applications. IEEE, pp 645–651
22. Samet S, Miri A, Orozco-Barbosa L (2007) Privacy preserving k-means clustering in multi-party environment. In: SECRYPT, pp 381–385
23. Shai H, Victor S (2018) HElib: An implementation of homomorphic encryption. https://github.com/shaih/HElib. Accessed 11 Aug 2018
24. Theodouli A, Draziotis KA, Gounaris A (2017) Implementing private k-means clustering using a LWE-based cryptosystem. In: 2017 IEEE symposium on computers and communications, ISCC 2017, Heraklion, Greece, 3–6 July 2017, pp 88–93
25. Tu S, Frans KM, Madden S, Zeldovich N (2013) Processing analytical queries over encrypted data. In: Proceedings of the VLDB Endowment
26. Vaidya J, Clifton C (2003) 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, pp 206–215
27. Wu D, Haven J (2012) Using homomorphic encryption for large scale statistical analysis. Technical Report from https://crypto.stanford.edu/~dwu4/papers/FHE-SI_Report.pdf. Accessed 11 Aug 2018
28. Zheng Z, Xie S, Dai H-N, Wang H (2018) Blockchain challenges and opportunities: a survey. Int J Web Grid Serv 14(4):352–375
29. Zyskind G, Nathan O, Pentland AS (2015) Decentralizing privacy: using blockchain to protect personal data. In: Proceedings, IEEE security and privacy workshops. SPW 2015:2015