Privacy-Preserving Clustering: A New Approach Based on Invariant Order Encryption

Mihail-Iulian PLEȘA and Cezar PLEȘCA

***Abstract*—Cloud computing is increasingly used. One main use of cloud computing is the running of a machine learning algorithm. Due to the large amount of data required for these algorithms, they can no longer be run on personal computers. Uploading personal data to the cloud automatically raises the issues of confidentiality of this data. In this paper, we show through a series of experiments that an order-preserving encryption algorithm can be applied to guarantee the confidentiality of the input of two well-known clustering algorithms: K-Means and DBSCAN. We show that K-Means can be modified to be applied over the encrypted data. We also proposed a slight improvement to an order-preserving encryption scheme to ensure that it is randomized, therefore increasing its security level. Finally, after studying the performance of clustering algorithms over encrypted data we show a practical application of this idea, namely the color reduction over an encrypted image.**

***Index Terms*—cloud computing, DBSCAN, order-preserving encryption, K-Means.**

I. Introduction

Nowadays, cloud computing has starting to be used more and more often. Software as a service (SaaS) is probably the best-known application of cloud computing. SaaS allows third-party providers to host applications, that are available to the users over the internet. In recent years, the number of SaaS applications has increased. One such application is machine learning. Given the large amount of data required for a machine learning algorithm, running it on personal computers is becoming increasingly difficult and costly. Companies allow this type of software to be run in the cloud for certain costs. Some examples are IBM Watson Machine Learning, Microsoft Azure, Google Cloud AI or Amazon AWS Machine Learning [1-4]. Machine learning as a service is certainly more efficient for both programmers and users but it raises a major problem: data confidentiality. Normally, a machine learning algorithm requires a training stage before it can be used by users. In this step, the algorithm receives as input a large amount of data. For the algorithm to be useful, users must upload their data to the cloud where the model is running, which causes a privacy problem.

Currently, there are two main solutions to this problem: homomorphic encryption and secure multi-party computation. Homomorphic encryption represents one type of encryption algorithms that allows calculations to be performed directly over the encrypted data. The programmer encrypts its data before uploading it to the cloud, trains the model over the encrypted data, and exposes this model to the user. To use the model, the user encrypts its data, uploads it to the cloud, runs the model, and obtains the encrypted result. The user then will decrypt the result returned by the cloud. One framework able to use homomorphic encryption in machine learning problems is CryptoNets [5].

In secure multi-party computation, a model allows several parties to calculate together the result of a function that receives inputs from each party, keeping the confidentiality of those inputs. One example of using secure multi-party computation in machine learning problem is tf-encrypted [6].

In this paper, we propose a third solution to the problem of data *confidentiality* in machine learning problems. We focus on two unsupervised machine learning algorithms used for clustering: K-Means and DBSCAN. Our solution is to apply order-preserving encryption to encrypt the input of the clustering algorithm. The result we propose can be summarized by the following steps:

1. The user locally encrypts the input of the algorithm and sends the encrypted data to a remote server.
2. The server runs the algorithm over the encrypted data and send back the results to the user.

Through a series of experiments, we show that the output of the algorithm running over the encrypted data is as good as the algorithm running over the plain data. Since no encryption oracle is involved, we are concerned only with ciphertext only attacks. As far as we know at the moment, this is the first paper proposing the idea of combining order-preserving encryption with the clustering algorithm to assure data confidentiality when the algorithm is running in the cloud. We also proposed improvement over a known order-preserving encryption scheme. The paper is structured as follows: in Section II we present the modified encryption scheme, in Section III we provide an introduction into clusterization algorithms, Section IV presents evaluation metrics for clustering algorithms, Section V is dedicated to experiments, Section VI presents an example of secure colour reduction, Section VII unfolds the conclusions drawn.

II. The Encryption Scheme

One of the best-known order-preserving encryption schemes and the one treated in this paper belongs to Boldyreva [7]. Let N be an integer and denote by $[N]$ the set $\{1,2,\ldots,N\}$. The scheme defines the space of plaintexts as the set $[N]$ and the space of ciphertexts as the set $[M]$, $M >> N$. The core idea behind the scheme is to consider a

M.-I. PLEȘA is with the Military Technical Academy "Ferdinand I", Bucharest, Romania (e-mail: mihai.plesa@mta.ro).

C. PLEȘCA is with the Military Technical Academy "Ferdinand I", Bucharest, Romania (e-mail cezar.plesca@mta.ro).

function $f:[N]\to[M]$ such that $\forall x,y\in[N]$, $f(x)>f(y)$ if $x>y$. The scheme is symmetric with a key size of 32 bytes. Every encryption key defines a unique function f. To encrypt a plaintext message $n\in[N]$ the function f is applied, obtaining the ciphertext $c=f(n)$. Let $g:Imf(f)\to[N]$ be the function that maps each element from the image of the function f to its corresponding element from the preimage, $\forall n\in[N]$, $g(f(n))=n$. Since $[N]$ is the sets of the plaintexts, the preimage of the function f is the set $[N]$ itself. To decrypt the ciphertext $c\in Imf(f)$, the function g is applied, obtaining the plaintext $n=g(c)$. One thing to note is that the decryption operation is not defined for all elements in the set $[M]$ but only for those belonging to the image of f.

In this paper, we use a modified version of the Boldyreva scheme. The original scheme has two major drawbacks: it is deterministic and therefore suffers from an attack through which half of the plaintext bits can be recovered only from the cyphertext [8]. We address these two problems by slightly modifying the original scheme.

The deterministic nature of the original scheme is a serious problem if the attacker knows the range of plaintexts. Suppose for example that the user encrypts a grayscale image. Each plaintext is an integer in the range $[0,255]$. Since the scheme is order-preserving the attacker can deduce that the smallest ciphertext corresponds to a plaintext value of 0, the next smallest ciphertext corresponds to a plaintext value of 1, etc. In the case of image encryption, given the deterministic nature of the scheme, the ciphertextof a certain pixel will appear several times in the encrypted image. The fact that the attacker decrypts a single ciphertext implies that he will know several plaintext values from the encrypted image. If the scheme would be randomized, multiple encryptions of the same plaintext would result in multiple independent ciphertexts.

In this section, we proposed a simple modification of the scheme to improve its security. Let $(GenB, EncB, DecB)$ be the key generation, encryption and decryption algorithms defined in the original version of the scheme.

The key generation algorithm, $GenB$, is modified to return two keys: k_1 and k_2. The key k_1 is the output of $GenB$, where the second key k_2 is a 32 bytes randomly generated integer. We denoted the modified version of $GenB$ as Gen.

The modified encryption algorithm denoted by Enc receives as input an integer m and the keys k_1 and k_2. To encrypt the plaintext message $m\in Z$, Enc generates a random integer, r, in the range $[0,k_2-1]$. The ciphertext outputted by Enc_{k_1,k_2} is the encryption of $m*k_2+r$ using $EncB$ and the key k_1 as in (1):

$$Enc_{k_1,k_2}=EncB_{k_1}(m*k_2+r) \tag{1}$$

The modified decryption algorithm denoted by Dec receives as input a ciphertext $c\in Z$ and the keys k_1 and k_2. The ciphertext c is decrypted using $DecB$ and the key k_1. The value obtained is scaled down by a factor of k_2 as in (2):

$$Dec_{k_1,k_2}=\left\lfloor\frac{DecB_{k_1}(c)}{k_2}\right\rfloor \tag{2}$$

Intuitively, each plaintext m is represented by a random integer chosen from the range $[k_2*m, 2*k_2*m-1]$. In this way, the scheme is randomized without losing the classic order relations (<,=,>) over the $\mathbb{R}$). Given a ciphertext c, an attacker will now recover half of the bits of $m*k_2+r$ but since r is randomly chosen at each encryption, it will be harder for the attacker to recover bits of m.

The idea behind our proposed modification is somewhat similar to that of LWE (Learning with Errors) encryption schemes: before encryption, we add a random error to the plaintext, large enough to increase the security of the original scheme but sufficiently small not to break the order relations between the ciphertexts [15].

To see how the security of the scheme is improved by our modification considered the following example. Suppose that the attacker knows that the plaintext space consists of two messages: m_0 and m_1, $m_0<m_1$. The attacker intercepts two ciphertexts: c_0 and c_1, $c_0<c_1$. Since the original scheme is deterministic and order-preserving the attacker can easily deduce that c_0 is the encryption of m_0 and c_1 is the encryption of m_1. In our modified version, instead of directly encryptingm_i, we encrypt a random integer in the range $[k_2*m_i, 2*k_2*m_i-1]$ thus even if $c_0<c_1$, they can represent the encryption of the same plaintext. The attacker can no longer deduce that c_0 is the encryption of m_0 and c_1 is the encryption of m_1.

III. Clustering Algorithms

Clustering algorithms are unsupervised machine learning algorithms that indicate the geometric structure of a set of points. In this paper, we consider two well-known clustering algorithms: K-Means and DBSCAN. Although each algorithm has its own specific input data, both of them receive as input a finite set of points $X=\{x_1,x_2,\ldots,x_N\}\subset\mathbb{R}^n$. The general purpose of a clustering algorithm is to group the points from the input set into clusters. Let k be the number of clusters (in the case of K-Means this number is given as an input to the algorithm and in the case of DBSCAN the number if determined by the algorithm). Both algorithms output a set of labels $L=\{l_1,l_2,\ldots,l_N\}\subset\mathbb{N}^n$, $1\le l_i\le k\ \forall 1\le i\le N$ indicating the cluster to which each point belongs, the point $x_i\in X$ belonging to the cluster $l_i\in L$. In *addition* to the set of labels L, K-Means also returns a set of centroids, $C=\{c_1,c_2,\ldots,c_k\}\subset\mathbb{R}^n$. Let $S_i=\{x_j\mid l_j=i\}$, $1\le i\le k_1\le j\le N$, be the set of all points that belong to cluster i. The centroid c_i represent the arithmetic

 Digital Object Identifier 10.32754/JMT.2020.2.10

mean of all points from the cluster *i* and is calculated in (3). The centroids can be further used in an algorithm like KNN

$$c_i = \frac{1}{|S_i|}\sum_{x_j \in S_i} x_j \tag{3}$$

In this paper, we propose a new strategy that allows running either of the two algorithms in the cloud without compromising the privacy of the input *X*. We are not concerned with the confidentiality of the other inputs parameters which the algorithms have. Our strategy involves encrypting every point of the input set and send that encrypted set into the cloud where the clustering algorithm will receive it as input. We prove by a series of experiments that the set of labels output by the algorithm run over the encrypted set are as good as those output by the algorithm run over the plain data.

In the case of *k*-means, we prove experimentally that decrypting the centroids resulted from encrypted data is geometrically close (i.e. using a well-known metric) to the centroids determined by the algorithm over the plain data. The encryption scheme used in all experiments is the one presented in Section II.

Let $(p_1, p_2, \ldots, p_n)$ the coordinates of a point $p \in X$. By definition, encrypting the point *p* involves encrypting each coordinate. Since the encryption function of a point is not defined in the scheme, we define the encryption of a point $p \in X$ of coordinates $(p_1, p_2, \ldots, p_n), p_i \in \mathbb{Z}$ under the keys k_1 and k_2 generated with *Gen* as in (4).

$$\begin{aligned} &EncP_{k_1,k_2}(p) = \\ &= \left(Enc_{k_1,k_2}(p_1), Enc_{k_1,k_2}(p_2), \ldots, Enc_{k_1,k_2}(p_n)\right) \end{aligned} \tag{4}$$

Similarly, the decryption of point is defined in (5).

$$\begin{aligned} &DecP_{k_1,k_2}(p) = \\ &= \left(Dec_{k_1,k_2}(p_1), Dec_{k_1,k_2}(p_2), \ldots, Dec_{k_1,k_2}(p_n)\right) \end{aligned} \tag{5}$$

K-Means is one of the best-known clustering algorithms. There are many variants of the algorithm but the one used in this paper is the naive K-Means [9-12]. The input of the algorithm consists of a finite set $X = \{x_1, x_{2,} \ldots, x_N\} \subset \mathbb{R}^n$ of *N* points and a natural number $k > 0$ representing the number of clusters to be formed. The algorithm outputs the coordinates of *k* centroids and assigns a label to each point in the set *X* that represents the cluster to which the point belongs. The procedure is iterative and consists of two main steps. Let $C^{(t)} = \{c_1^{(t)}, c_2^{(t)}, \ldots, c_k^{(t)}\}$, $L^{(t)} = \{l_1^{(t)}, l_2^{(t)}, \ldots, l_N^{(t)}\}$, $1 \le l_i^{(t)} \le k \forall 1 \le i \le N$ be the sets of centroids and labels at iteration t.

The first step of the algorithm is the assignment. In this step, for each point $x_i \in X$, $1 \le i \le N$, the distance from all centroids is computed and the label is updated to include the point in the cluster of the nearest centroid. Let d_{ij}, $1 \le i \le N 1 \le j \le k$, be the distance between the point x_i and centroid $c_j^{(t-1)}$. The label of x_i at iteration *t*, $l_i^{(t)}$ is computed such that $d_{il_i^{(t)}} \le d_{ij} \forall 1 \le j \le k$.

The second step of the algorithm is the update. In this step, the set $C^{(t)}$ is updated according to the set of labels $L^{(t)}$ calculated in the first step. Let $S^{(t)}{}_i = \left\{x_j \mid l_j^{(t)} = i\right\}$, $1 \le i \le k 1 \le j \le N$, the set of all points that belong to cluster *i*. Each centroid $c_i^{(t-1)}, 1 \le i \le k$ is updated according to (6)

$$c_i^{(t)} = \frac{1}{\left|S^{(t)}{}_i\right|}\sum_{x_j \in S^{(t)}{}_i} x_j \tag{6}$$

For the first step, the set of centroids, $C^{(0)}$, is chosen randomly from the set $\mathbb{R}^n$. The algorithm converges and stops when $C^{(t)} = C^{(t-1)}$.

Let X_{enc} be the set of encrypted points in *X*. Since we apply the algorithm over the encrypted data the resulting centroids will be calculated over the ciphertexts space. For the decryption operation to be defined, the encrypted centroids must belong to the set of valid ciphertexts. To solve this problem, we propose a slight modification on the update step of the K-Means algorithm.

In the original version of the algorithm, the centroid $c_i^{(t)}$ is updated as the arithmetic means of all points that belong to cluster i. After this computation over cyphertexts, the updated centroid $c_i^{(t)}$ could be an invalid ciphertext (i.e. a ciphertext for which the decryption is not defined).

Our solution to this problem is to approximate the updated centroid $c_i^{(t)}$ to the nearest point in the set X_{enc}. Since the entire set of points, X_{enc}, is composed of valid ciphertexts, it is certain that the updated centroid $c_i^{(t)}$ will also be a valid ciphertext. So, to update the $c_i^{(t-1)}$ centroid, the arithmetic mean value of all points that belong to cluster *i* is calculated and then it is approximated to the nearest point in the set X_{enc}. If *p* is the arithmetic mean obtained by (6) and d_j, $1 \le j \le N$, the distance between the point *p* and $enc(x_j)$, then the updated the centroid $c_i^{(t)}$ will be calculated as $enc(x_u)$ such that $d_u \le d_j \forall 1 \le j \le N$, $1 \le u \le N$.

DBSCAN is another well-known clustering algorithm [13]. Unlike K-Means, DBSCAN does not require the number of clusters to be known and does not return a set of centroids. It is also able to find noise points i.e. points that do not belong to any other cluster. There are several differences between K-Means and DBSCAN:

1. DBSCAN works on non-globular data.
2. DBSCAN produces clusters according to the density of points.
3. K-Means is optimized according to least-squares while DBSCAN search density-connected regions of points.

There are three inputs to the algorithm. The first input is a finite set $X = \{x_1, x_{2,} \ldots, x_N\} \subset \mathbb{R}^n$ of points that will be

clustered. The second input is a real number $\varepsilon \in \mathbb{R}$ that specifies the radius of a neighbourhood of a point from the set *X*. The last parameter is a natural number $M \in \mathbb{N}$ that defined the minimum number of points needed in a neighbourhood such that it can be considered as a cluster. The output of the algorithm is the set of labels $L = \{l_1, l_2, \ldots, l_N\}$ assigned to each point.

For each point $p \in X$ we define the neighbourhood of the point *p* as the set $V_\varepsilon(p) = \{q \in X \mid d_{pq} \le \varepsilon\}$ where d_{pq} represents the Euclidean distance between the points *p* and *q*. A point $p \in X$ is called a core point if the set $V_\varepsilon(p)$ contains at least *M* points i.e. $|V_\varepsilon(p)| \le M$. A point $q \in X$ is a directly reachable point from a core point *p* if $d_{pq} \le \varepsilon$. The definition of a directly reachable point is similar to the definition of a neighbourhood. The difference between the two is that the set $V_\varepsilon(p)$ is defined for every point $p \in X$ while a directly reachable point is defined only from a core point.

DBSCAN maintains an internal variable *C* which initially is 0 and is incremented each time a new cluster is found. At each iteration, the algorithm identifies the first point, $x_i \in X$ that does not have a label assigned to it. If the set $V_\varepsilon(x_i)$ has more than *M* elements (i.e. $|V_\varepsilon(x_i)| \ge M$) then a new cluster has been found and the variable *C* is incremented, the point x_i receiving the label *C*, $l_i = C$. If the neighborhood of the point x_i does not contain more than *M* elements, then the point is considered a noise point. For each other point $x_j \in X$, $j \ne i$, if the point x_j is noise or does not have a label, if $|V_\varepsilon(x_j)| \ge M$ then the point x_j also receive the label *C*.

IV. Evaluation Metrics of Clustering Algorithms

Let $X = \{x_1, x_2, \ldots, x_N\} \subset \mathbb{R}^n$ be the input data in plain form and $X_{enc} = \{encP(x_1), encP(x_2), \ldots, encP(x_N)\} \subseteq \mathbb{C}$ be the encrypted input data. Both algorithms analyzed in this paper return a set of labels. Let $L = \{l_1, l_2, \ldots, l_N\} \subset \mathbb{N}^n$ be the set of labels returned by the algorithm over the plain data, *X*. Similarly, we define $L_{enc} = \{l_{enc\_1}, l_{enc\_2}, \ldots, l_{enc\_N}\} \subset \mathbb{N}^n$ as the set of labels returned by the algorithm over the encrypted input X_{enc}. To evaluate the performance of the clustering algorithm over the encrypted data we will use the Adjusted Rand index or *ARI* [16].

ARI is an evaluation metric used for clustering algorithms when the real clustering is known. Let $C = \{c_1, c_2, \ldots, c_N\} \subset \mathbb{N}^n$, $1 \le c_i \le n_1 \forall 1 \le i \le N$ be the set of labels known to be truth. Let $K = \{k_1, k_2, \ldots, k_N\} \subset \mathbb{N}^n$, $1 \le k_i \le n_2 \forall 1 \le i \le N$ be the set of labels returned by the clustering algorithm. Here n_1 and n_2 are the ground truth number of clusters respectively the number of clusters found by the algorithm. Let *x* be the number of pairs of points $(i, j), 1 \le i, j \le N, i \ne j$ that have the same label both in the set *C* and in the set *K* i.e. $c_i = c_j$ and $k_i = k_j$. Let *y* be the number of pairs of points $(i, j), 1 \le i, j \le N, i \ne j$ that have different labels in the set *C* and different labels in the set *K* i.e. $c_i \ne c_j$ and $k_i \ne k_j$. The *ARI* index is calculated in (7):

$$ARI = \frac{RI - E[RI]}{\max(RI) - E[RI]}, \tag{7}$$

where $E[X]$ is the expected value of the random variable *X* and *RI* is calculated in (8):

$$RI = \frac{x + y}{C_2^N} \tag{8}$$

ARI values are bounded in the interval $[-1,1]$. A uniform random assignment of labels by the algorithm will result in an *ARI* value of 0. An *ARI* value of –1 indicates independent clustering i.e. the set *C* and *K* are not correlated with each other. An *ARI* value of 1 indicates the two clustering are identical i.e. any pair of points that are part of the same cluster according to the real clustering will be part of the same cluster according to the clustering given by the algorithm.

Given the fact that K-Means returns a set of centroids in addition to the set of labels, we have to define an evaluation metric to compare the centroids given by K-Means over the plain data with those given by the algorithm over the encrypted data.

Let $C_t = \{C_1, C_2 \ldots C_{n_1}\} \subset \mathbb{R}^n$ be the set of centroids known to be true (i.e. obtained on plain data clustering) and $C_k = \{C_1, C_2 \ldots C_{n_2}\} \subset \mathbb{R}^n$ be the set of centroids returned by K-Means. Intuitively, to analyze the performance of the algorithm we must compute some kind of distance between the sets C_t and C_k. Since we are working in a Euclidean space which is a metric space, we can use the Hausdorff distance.

Let $d(x, y)$ be the Euclidean distance between the points $x, y \in \mathbb{R}^n$. The Hausdorff distance between two finite subsets *X* and *Y* of $\mathbb{R}^n$, $d_H(X, Y)$ is defined in (9)

$$d_H(X, Y) = \max\left(\max_{x \in X} \min_{y \in Y} d(x, y), \max_{y \in Y} \min_{x \in X} d(x, y)\right) \tag{9}$$

If the two set of centroids, C_t and C_k are identical then $d_H(C_t, C_k) = 0$.

V. The Experiments

In the first experiment we study how the proposed modification of the original K-Means algorithm impacts its performance, using the two metrics presented in the previous section. We consider different sets of points and for each set we calculate the labels and the centroids given by the original algorithm and those returned by the modified version. We then use the *ARI* metric to compare the two sets of labels and Hausdorff distance to compare the two sets of centroids.

 Digital Object Identifier 10.32754/JMT.2020.2.10

In this experiment we generate 100 sets of points denoted as $X_1, X_2, \ldots, X_{100}$. The set $X_i, 1 \le i \le 100$ is partition into $i*10$ clusters. Each set has a total of 5000 points. The coordinates of each centroid are randomly generated in the interval $\left[-2^8, 2^8\right]$. The standard deviation of each cluster is set to 5. To generate the clusters, we used the function *make_blobs* from the scikit-learn library [14].

In Figure 1 we plot how the Hausdorff distance changes according to the number of clusters. Similarly, in Fig. 2 we show how *ARI* changes as the number of clusters increases. From Figure 1 it can be seen that the Hausdorff distance increases as the number of clusters increases. This means that the centroids resulted from the modified version of k-means tend to distance from the centroids determined by the original version of k-means. Since all the points have coordinates in the range $\left[-2^8, 2^8\right]$, the biggest distance between two points is $\sqrt{524288} \approx 724$. In our experiment, the Hausdorff distance is a maximum 20 which is about 2.7% of the biggest distance.

The *ARI*, plotted in Fig. 2, decreases as the number of clusters increases. This means that the labels assigned by the modified version of k-means tends to behave more like uniform randomly assigned labels.

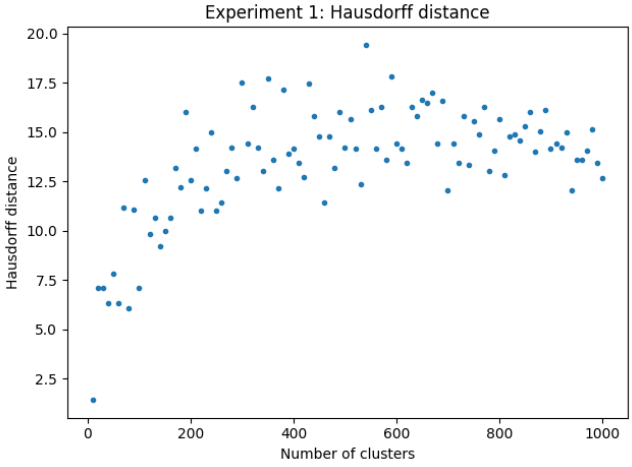

Figure 1. *Hausdorff* distance between the two groups of points

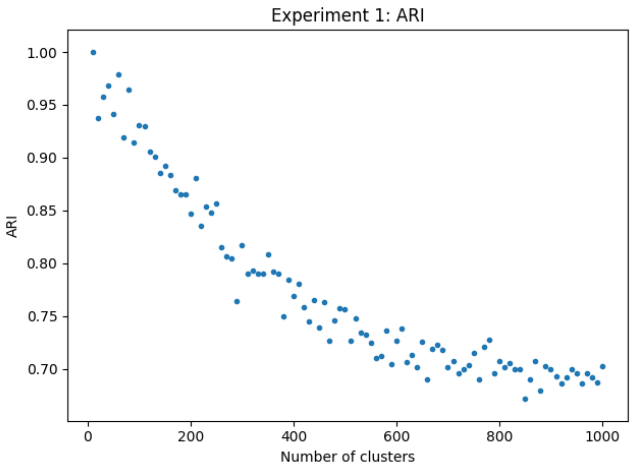

Figure 2. *ARI* between the two sets of labels

Overall, it can be concluded that the efficiency of the modified k-means decreases compared to that of the original version. However, in the previous experiment, the standard deviation of a cluster is a constant value of 5. Intuitively, the results of the first experiment can be explained by the fact that as the number of clusters increases, because the standard deviation is constant, the distance between the clusters becomes small thus the clusters can no longer be differentiated. A visual example of this is given in Fig. 3 where the whole set of 5000 points was grouped into 500 clusters each with a standard deviation of 5 was plotted.

To compensate for this situation, we choose to adapt the standard deviation according to the number of clusters. If *N* is the number of clusters then the standard deviation of each cluster, σ_N is calculated in (10):

$$\sigma_N = \frac{1}{0.02 * N} \tag{10}$$

The motivation behind (10) is to decrease the standard deviation of each cluster as the number of clusters increases. In this way, the space between clusters will increase with their number. Fig. 4 shows the clustering of 5000 points into 500 clusters but his time the standard deviation was adjusted according to (10).

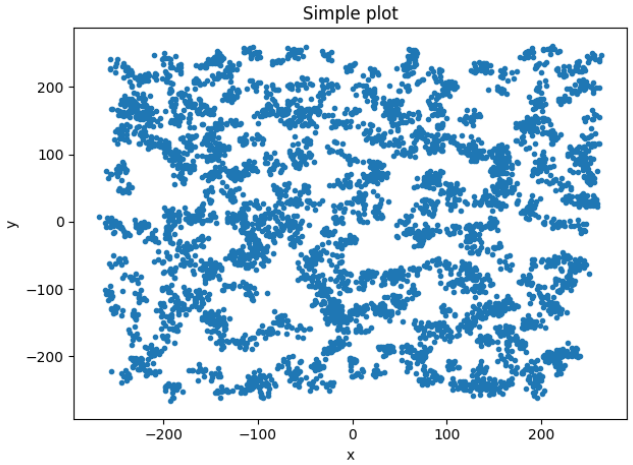

Figure 3. 500 clusters with a standard deviation of 5

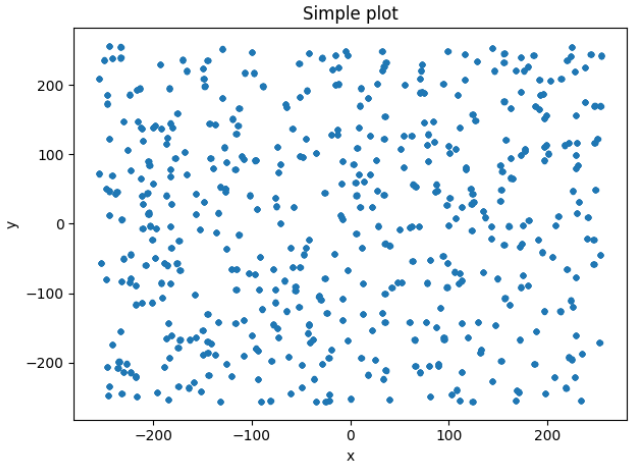

Figure 4. 500 clusters with a standard deviation of 0.1

To study the impact that standard deviation adjustment has on Hausdorff distance and *ARI*, the first experiment was repeated under the same conditions, only this time the standard deviation was updated according to (10). Fig. 5 and Fig. 6 show the new values for the Hausdorff distance and *ARI*. From the plots displayed in the Fig. 5 and Fig. 6, it can

be observed that both the Hausdorff distance and *ARI* keep their values almost constant. This time, the Hausdorff distance is 0.2% from the biggest distance between two points. This means that the centroids resulted from the modified k-means are very closed to those returned by the original version. The value of *ARI* is very close to 1 which shows that the clustering made by the modified algorithm is almost identical to that made by the original algorithm.

We can conclude that as long as the clusters are separable, our modification does not impact the performance of the algorithm. When the clusters are not separable it is known the k-means is not a suitable choice for clustering.

We can conclude that our modification does not impact the performance of the algorithm provided that the input set can be clustered using k-means (i.e. the input data is geometrically separable).

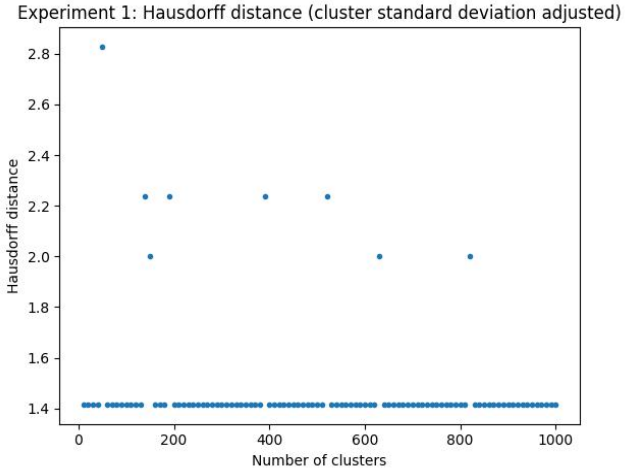

Figure 5. Hausdorff distance after cluster standard deviation was updated

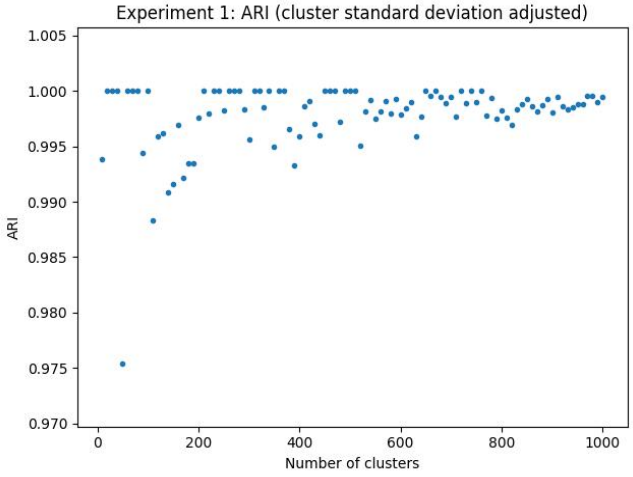

Figure 6. *ARI* after cluster standard deviation was updated

Our second experiment studies the performance of k-means over the encrypted data. As we have specified in Section III, to be able to run k-mean over the encrypted data we must use our modified version of the algorithm. In the second experiment we will study the impact of data encryption with respect to clustering performance. For simplicity, we called the modified version of k-means mk-means. We will study the *ARI* between the labels returned by the mk-means over the plain set of points and labels returned by mk-means over the encrypted set of points. The centroids returned by the mk-means over the encrypted set of points are decrypted and then the Hausdorff distance between the decrypted centroids and the centroids returned by mk-means over the plain set is calculated.

As in the previous experiment, 100 sets of points were generated: $X_1, X_2, \ldots, X_{100}$. Each set, X_i, $1 \le i \le 100$ contains 1000 points partitioned into *i* cluster. The standard deviation of each set was calculated according to (10). Fig. 7 and Fig. 8 show the Hausdorff distance and *ARI*, respectively.

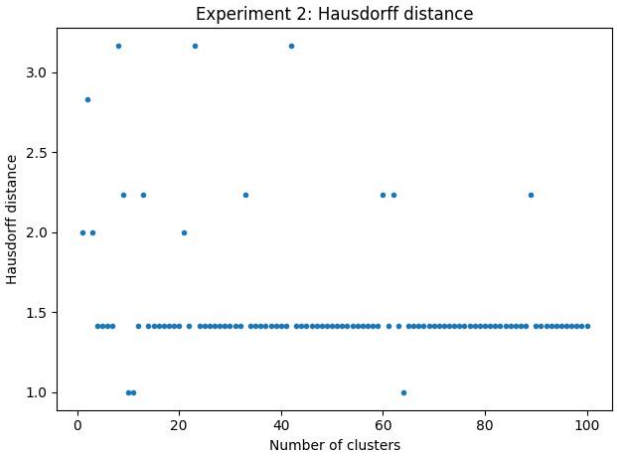

Figure 7. Hausdorff distance

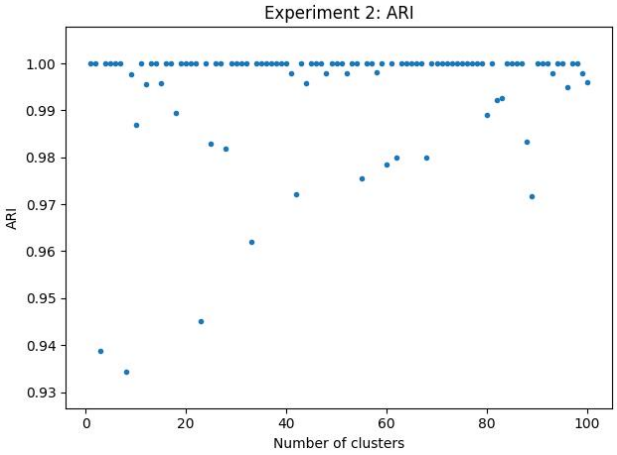

Figure 8. *ARI* index

The Hausdorff distance between the decrypted centroids calculated using mk-means over the encrypted data and the centroids resulted from the same algorithm over the plain data is about 0.2% from the biggest distance that can be defined in this experiment.

The *ARI* between the labels of the encrypted data and the labels of the plain data is almost 1 which means that the clusters generated by mk-means over the encrypted data are almost identical to those generated over the plain data.

Although k-means is an efficient algorithm, there are cases in which the configuration of the points is not globular thus k-means will not produce the expected clustering. In those cases, DBSCAN is a much more appropriate algorithm. Fig. 9 shows an example of clustering using k-means and DBSCAN. There are 1000 grouped into 4 clusters. To generate the points, a linear transformation with randomly generated values from a normal distribution was applied to the output of *make_blobs* function. In this case, the clustering produced by DBSCAN is more relevant than the one produced by k-means.

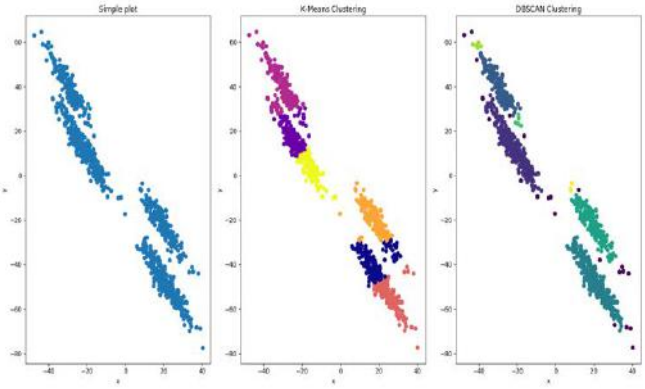

Figure 9. K-Means and DBSCAN vs on non-globular clusters

Another example when DBSCAN produces better results than k-means is shown in Fig. 10. This time the points were *generated* using *make_circle* function from sklearn library.

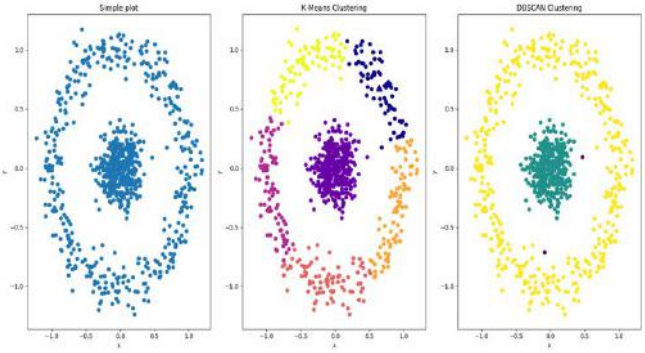

Figure 10. K-Means and DBSCAN vs on circular clusters

In our third experiment we study the performance of DBSCAN over the encrypted data. In this experiment, we generate 100 sets of points: $X_1, X_2, \ldots, X_{100}$ each set X_i containing 1000 points. To create non-globular clusters, a 2×2 linear transformation with random values from a normal distribution was applied to each point. Each set was encrypted and then DBSCAN was applied over the plain set and the encrypted set. The *ARI* between the two sets of labels was calculated and plotted in Fig. 11.

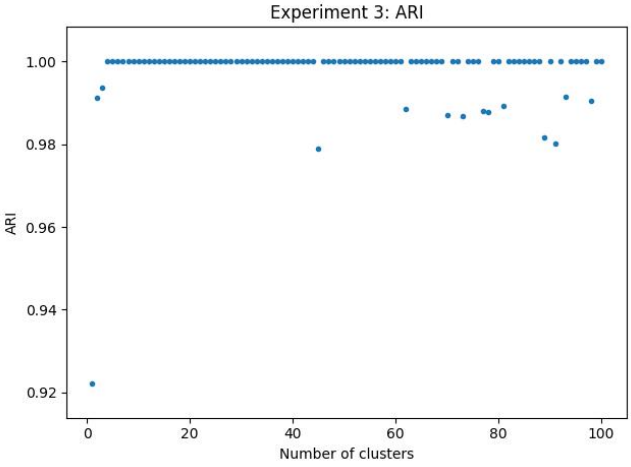

Figure 11. *ARI* index

The *ARI* is almost 1 regardless of the number of clusters which means that the algorithm can be applied over the encrypted data and will produce the same clustering as if applied over the plain data. In addition to the clustering itself, this experiment also studied the number of clusters produced by the algorithm. Although each set X_i was generated to contain *i* clusters, DBSCAN can produce a different number. Fig. 12 shows the relation between the number of clusters initially generated and the number of clusters produced by DBSCAN.

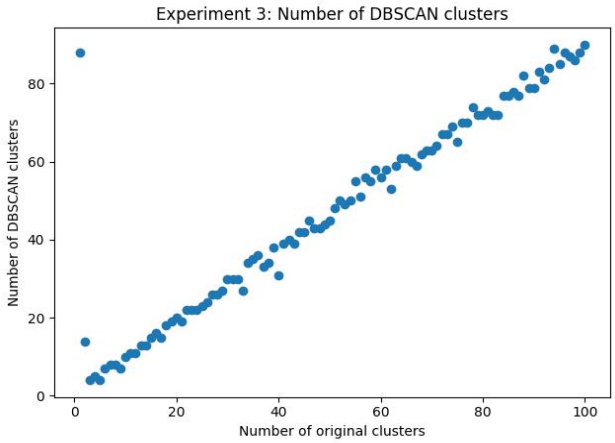

Figure 12. The number of clusters produced by DBSCAN

This experiment shows that the relation between the two number of clusters is almost linear with a few exceptions when DBSCAN produced a very different number of clusters than the original one.

VI. Colour Reduction over Encrypted Images

The colour reduction is a technique that reduces the number of colours in an image while maintaining much of the information of it. In this section we present an experiment that studies colour reduction on encrypted images as a direct application of our modified version of k-means. Let *I* be an *nxm* RGB image. If each pixel has a unique colour, then we need *nxmx*3 bytes of memory to store the image *I*. Colour reduction allows the representation of the same image *I* with a small number of colours $k \ll nxmx3$ without affecting its semantic meaning.

Let *X* be the set of all colours that appear in the image *I*. Since colour is a triplet of natural numbers, we can view $X = \{X_1, X_2, \ldots, X_n\}$ as a set of points from $\mathbb{N}^3$, $X \subset \mathbb{N}^3$.

Colour reduction starts by applying the k-means algorithm over the set *X*. Let $C = \{C_1, C_2, \ldots, C_k\}$, $k \ll n$ and $L = \{L_1, L_2, \ldots, L_n\}$ be the sets of centroids respectively labels returned by k-means. The next step of the algorithm consists of replacing each point from the set *X* with its centroids: the point X_i is replaced by the point $C_{L_i}, \forall i,\ 1 \le i \le n$. The number of colours is reduced from *n*, the original number of colours of the image to *k*, the number of centroids.

To maintain the confidentiality of the image, it will be encrypted using the scheme presented in Section II. As shown in Section V, our modified version of k-means performed well over encrypted data both from the perspective of *ARI* and from the perspective of Hausdorff distance. Since colour reduction is simply a call to k-means, we can reduce the number of colours directly from the encrypted image by applying our modified version of the algorithm, mk-means.

In the fourth experiment we evaluate the performance of colour reduction over encrypted images by applying the algorithm to both the original and the encrypted image. We compare the decrypted image with the original using PSNR, a metric that calculates how different two images are.

For this experiment we use the TensorFlow Flower Dataset. The dataset contains a total 3669 colour images from which 100 were randomly selected. Each selected image was resized to 32×32 pixels and the number of colours was reduced to 64.

The experiment shows that the average of the 100 PSNR values is 34.27 with a standard deviation of 3.66.

To better illustrate the performance of our algorithm, Table I shows the results for five concrete images from TensorFlow Flower Dataset. The images were resized to 128×128. For each image the number of colours was reduced to 64.

TABLE I. THE PSNR OBTAINED FOR FIVE IMAGES

Name of the image	PSNR
4914793782_d0ea760791.jpg	34.05
5840476802_dfa40deb1f_m.jpg	36.79
3697780051_83e50a6dd1_m.jpg	40.78
15381511376_fd743b7330_n.jpg	39.28
14335561523_f847f2f4f1.jpg	34.69

VII. CONCLUSIONS AND FURTHER DIRECTION OF RESEARCH

In this paper, we showed by a series of experiments that order-preserving encryption can be applied to solve the problem of data confidentially when they represent input for a clustering algorithm. In the first part, we show how the Boldyreva encryption scheme can be randomized to obtain a better security level. We then propose a slight modification of the k-means algorithm to be able to run over the encrypted data generated by our scheme.

The first experiment showed that modification made to the original K-Means algorithm very slightly affects its performance. The second experiment showed that the results of our K-Means proposal over the encrypted data are as good as those obtained applying the same clustering method over the plain data.

The third experiment proved that the DBSCAN performance over the encrypted data is as good as that obtained by the same algorithm over the plain data. Our fourth experiment presented a practical application of k-means over the encrypted data, namely colour reduction of an encrypted image.

Overall, from an experimental point of view, we can conclude that order-preserving encryption is a suitable solution to the problem of input privacy of the clustering algorithm. A further direction of research is to perform more experiments using other encryption schemes of this type. Another direction of research is the formal study of the applicability of this type of encryption scheme together with unsupervised machine learning algorithms.

REFERENCES

[1] "Watson Machine Learning - Overview," *IBM*. [Online]. Available: https://www.ibm.com/ro-en/cloud/machine-learning. [Accessed: 02-Sep-2020].

[2] "Cloud AI | Google Cloud," *Google*. [Online]. Available: https://cloud.google.com/products/ai. [Accessed: 02-Sep-2020].

[3] "Train and Deploy Machine Learning Models: Microsoft Azure," *Train and Deploy Machine Learning Models | Microsoft Azure*. [Online]. Available: https://azure.microsoft.com/en-us/free/machine-learning/. [Accessed: 02-Sep-2020].

[4] T. M. Mitchell, "Machine learning," *Amazon*, 2017. [Online]. Available: https://aws.amazon.com/machine-learning/. [Accessed: 02-Sep-2020].

[5] R. Gilad-Bachrach, N. Dowlin, K. Laine, K. Lauter, M. Naehrig, and J. Wernsing, "Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy" presented at the 2016 International Conference on Machine Learning, New York, June 19-24 2016. doi: 10.5555/3045390.3045413

[6] M. Dahl, J. Mancuso, Y. Dupis, B. Decoste, M. Giraud, I. Livingstone, J. Patriquin and G. Uhma, *Private machine learning in tensorflow using secure computation*. 2018, *arXiv preprint arXiv:1810.08130*.

[7] A. Boldyreva, N. Chenette, Y. Lee, and A. O'Neill, "Order-preserving symmetric encryption", presented at the 2009 *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, Berlin, April 26-30, 2009. doi:10.1007/978-3-642-01001-9_35

[8] A. Boldyreva, N. Chenette, and A. O'Neill, "Order-preserving encryption revisited: Improved security analysis and alternative solutions", presented at the 2011 *Annual Cryptology Conference*, Berlin, August14-18, 2011. doi:10.1007/978-3-642-22792-9_33

[9] K. Krishna and M. N. Murty, Genetic K-means algorithm. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics), 29*(3), pp. 433-439, 1999. doi: 10.1109/3477.764879

[10] K. Wagstaff, C. Cardie, S. Rogers, and S. Schrödl, "Constrained k-means clustering with background knowledge", in *Icml*, vol. 1, pp. 577-584, June 2001. doi:10.5555/645530.655669

[11] K. Alsabti, S. Ranka, and V. Singh, "*An efficient k-means clustering algorithm*", IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 27 997. doi:10.1109/TPAMI.2002.1017616

[12] H. H. Bock, "Origins and extensions of the k-means algorithm in cluster analysis", *Electronic Journal for History of Probability and Statistics*, vol. 4, no. 2, pp. 1-18, 2008.

[13] E. Schubert, J. Sander, M. Ester, H. P. Kriegeland, and X. Xu, "DBSCAN revisited, revisited: why and how you should (still) use DBSCAN". *ACM Transactions on Database Systems (TODS)*, vol. 42, no. 3, pp. 1-2, 2017. doi:10.1145/3068335

[14] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourgand, J. Vanderplas, "Scikit-learn: Machine learning in Python", *The Journal of machine Learning research*, 12, 2011, pp. 2825-2830. doi:10.5555/1953048.2078195.

[15] D. Micciancio and O. Regev, "Lattice-based Cryptography" *Post-Quantum Cryptography*, Post-Quantum Cryptography, pp. 147–191, 2009. doi:10.1007/978-3-540-88702-7_5.

[16] L. Hubert and P. Arabie, "Comparing partitions", *Journal of Classification*, vol. 2, no. 1, pp. 193–218, 1985. doi:10.1007/BF01908075