Secure k-Anonymization over Encrypted Databases

Manish Kesarwani, Akshar Kaul
IBM Research, India
{*manishkesarwani,akshar.kaul*}*@in.ibm.com*

Stefano Braghin, Naoise Holohan, Spiros Antonatos
IBM Research Europe — Ireland
{*stefanob@ie.,naoise.holohan@,santonat@ie.*}*ibm.com*

2021 IEEE 14th International Conference on Cloud Computing (CLOUD) | 978-1-6654-0060-2/21/$31.00 ©2021 IEEE | DOI: 10.1109/CLOUD53861.2021.00015

***Abstract*—Data protection algorithms are becoming increasingly important to support modern business needs for facilitating data sharing and data monetization. Anonymization is an important step before data sharing. Several organizations leverage on third parties for storing and managing data. However, third parties are often not trusted to store plaintext personal and sensitive data; data encryption is widely adopted to protect against intentional and unintentional attempts to read personal/sensitive data. Traditional encryption schemes do not support operations over the ciphertexts and thus anonymizing encrypted datasets is not feasible with current approaches. This paper explores the feasibility and depth of implementing a privacy-preserving data publishing workflow over encrypted datasets leveraging on homomorphic encryption. We demonstrate how we can achieve uniqueness discovery, data masking, differential privacy and k-anonymity over encrypted data requiring zero knowledge about the original values. We prove that the security protocols followed by our approach provide strong guarantees against inference attacks. Finally, we experimentally demonstrate the performance of our data publishing workflow components.**

I. Introduction

Nowadays, applications interact with a plethora of potentially sensitive information from multiple sources. As an example, modern applications regularly combine data from different domains such as healthcare and IoT. While such rich sources of data are extremely valuable for analysts, researchers, marketers and other professionals, data privacy technologies and practices face several key challenges to keep pace.

There are two major obstacles when it comes to hosting and sharing sensitive data. The first is that the public cloud solutions are not trusted with sensitive data (e.g. health, financial, or critical infrastructure data) and thus organisations have to invest in private or hybrid clouds as the hosting and processing environment. This adds complexity for the design and implementation and often comes with additional cost due to security and customisation. Homomorphic encryption provides an answer to these kinds of obstacles, by encrypting the data at their source while allowing operations on them and thus lifting the trust barrier from the hosting solution.

The second is data privacy. Data privacy technologies are applied in two major use cases. The first use case concerns data sharing, where data need to be sufficiently anonymized before being shared with researchers and analysts. The second use case concerns security. By anonymizing data at rest, the risk of breaches is minimised since sensitive information is protected.

Latest advances in regulation, like EU General Data Protection Regulation (GDPR), also propose anonymization for safely processing data when consent is not an option or organisations want to use them for purposes beyond those for which it was originally obtained for an indefinite period of time.

In this paper, we present how to apply different data privacy approaches to homomorphically encrypted data. Specifically, we present how we can achieve uniqueness discovery, data masking, differential privacy and k-anonymity over encrypted data, requiring zero knowledge about the original values. Uniqueness discovery allows the user to find which attributes or combinations of attributes (quasi-identifiers) appear with a lower frequency than a predefined threshold. This leads to the selection of attributes that need to be protected via a combination of data masking, differential privacy and k-anonymity approaches. We explore how we can securely apply all these techniques without leaking information about the original data, such as the domain cardinality or diameter.

The rest of the paper is organised as follows. Section II describes the motivating scenarios and use cases behind this work. Section III provides background information about the basic principles of data privacy and homomorphic computations and further outlines the data publishing workflow and describes the building blocks to achieving data privacy. In Section IV we present the related work. In Section V we present the secure protocols while in Section VI and Section VII we discuss their security guarantees and performance respectively. Finally, we conclude in Section VIII.

II. Motivation

The first major question that arises is why data encryption alone is not enough. In general, to protect the privacy of sensitive data, only encrypted data is outsourced to the third-party cloud providers and there exist well-established systems which allow secure query processing directly over encrypted data. On top of that, a more important question is, why someone should perform masking and anonymization over encrypted data in the cloud environment. In a typical scenario, the data is anonymized at the source and then uploaded to the cloud or shared with a third party. However, there are many reasons to perform the anonymization part on the cloud after data encryption.

The answer to these two questions relies on two major observations from the GDPR compliance standard. First, data encryption does not meet the high compliance standards,

2159-6190/21/$31.00 ©2021 IEEE
DOI 10.1109/CLOUD53861.2021.00015

since all the data encryption schemes are reversible. Second, anonymized data is not considered personal information and benefit from relaxed standards under GDPR [1]. Thus the need to apply non-reversible masking and anonymization over encrypted data is required. Furthermore, the GDPR mandates that the data controller needs to demonstrate that the state-of-the-art strategy is applied when it comes to pseudonymization/anonymization approaches. By relying on the cloud to deploy the state-of-the-art approaches, the operational and compliance model for the data owners becomes significantly easier.

Apart from the compliance regulations, there are several other factors that motivate anonymization over encrypted data. First, the objective of the data use may change over the course of time. Initially, it may not be desirable to share the data but after some time it might be required. This is common with enterprise data where confidentiality must be kept for several years before the data can be exchanged. Second, the users might want to selectively share data and thus apply anonymization to only the selected portion. In both cases, by pre-anonymizing the data we will not be able to reach the desired outcome. Furthermore, in the distributed IoT scenario, individual sensors may not have sufficient storage and processing capability and the ever-increasing volume of data makes it hard to anonymize at the source. As an operating model, it is far easier to homomorphically encrypt the data at source before outsourcing to the cloud and then anonymize on-demand at the cloud when it comes to sharing and collaborating.

Specific masking operations, k-anonymity and differential privacy fall into the non-reversible anonymization category and thus are the focus of this paper.

Our approach guarantees two major properties: a) the data owner does not send the data as-is and so the data host cannot see the original data and b) facilitate the application of on-demand non-reversible anonymization approaches to the data in order to meet compliance standards or selective data sharing.

III. Background

A. k-Anonymity

Based on the data privacy terminology, attributes in a dataset are classified as direct or quasi identifiers. Direct identifiers are uniquely identifying and are always removed or masked before the data release. Quasi-identifiers are sets of attributes that can uniquely identify one or very few individuals. As an example, the combination of ZIP code plus gender plus birth date can be uniquely identifying (in case of the US this combination can uniquely identify 87% of the population). Based on the k-anonymity approach [2], quasi-identifiers are generalized and clustered in such a degree that an individual is indistinguishable from at least $k-1$ other individuals.

B. System Entities

A Database-as-a-Service (DBaaS) architecture consists of below entities:

1) **Data Owner (DO)**: A company or an individual who is having a proprietary right to a sensitive database, such as a Bank. The Data Owner wants to securely outsource its data storage and future computations over data to a Cloud Service Provider.
2) **Cloud Service Provider (CSP)**: A third party, that provides the storage and computation capability as a service to its clients. For our scenario, the Cloud Service Provider is a system where any present-day state of the art database engine is running. For example, Amazon Redshift and Microsoft Azure SQL Database.
   In particular, we work in the two-party federated cloud setting, with two non-colluding public cloud servers. This model was introduced in Twin Clouds [3] and was subsequently used in related problems [4], [5]. Federated clouds are an example of Interclouds [6], a collection of global stand-alone clouds. Intercloud allows better load balancing and allocation of resources. A detailed survey of the taxonomy of intercloud architectures is presented in [6].

C. Trust Assumption

We assume that the CSP is honest-but-curious (or semi-honest) i.e. it is honest and executes the protocol correctly, but is also interested in the plaintext of the encrypted data stored at its site, either because it is curious or it has been compromised. In this paper, we will show that the honest-but-curious adversary will not be able to learn anything about the plaintext of the encrypted database, even though it can observe the computations and can take memory dumps. Further, we also prevent the leakage of any data clustering information available in the intermediate steps of secure k-anonymization, masking or differential privacy algorithms.

D. Homomorphic Computation

Homomorphic encryption schemes support direct computation of functions over encrypted data without needing to decrypt it first. To this end the seminal work of Gentry [7] presents a fully homomorphic encryption (FHE) scheme, which is capable of evaluating any arbitrary dynamically chosen function over an encrypted database without needing the secret key. But since computation over fully homomorphic encrypted data is still many orders of magnitude slower than the plaintext execution, this limits the practical deployment of these scheme for real workloads.

Another line of research built partial and somewhat homomorphic encryption schemes. Specifically partial homomorphic encryption (PHE) schemes support evaluation of a chosen atomic function over encrypted data (like Addition or Multiplication). For example, Paillier cryptosystem [8] supports addition over encrypted data without needing a secret key and ensures strong security guarantees. On the other hand, the somewhat homomorphic encryption (SHE) scheme supports the computation of low degree polynomials over encrypted data. For example, BGN cryptosystem [9] supports evaluation of any polynomial of degree *two* over encrypted

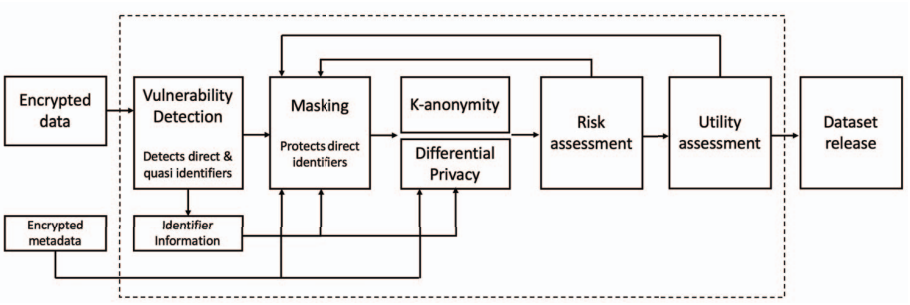


Fig. 1: The workflow for encrypted data de-identification process

data, while in LFHE cryptosystem [10], degree *d* polynomials can be evaluated, but it bases security on weaker assumptions of learning with error (LWE) or ring-LWE (RLWE) problems.

In general, a SHE encryption scheme consists of five basic algorithms: a) key generation $SHE.KeyGen(1^{\lambda})$ that takes as input the security parameter λ and output the secret key sk, the public key pk and public parameters $params$ b) encryption $SHE.Enc(params, pk, m)$ that takes the message m and evaluates the corresponding ciphertext c using $params$ and the public key pk c) decryption $SHE.Dec(params, sk, c)$ that takes the ciphertext c and decrypts it using $params$ and the secret key sk and outputs the corresponding plaintext message m d) addition $SHE.Add(pk, c_1, c_2)$ which takes two ciphertexts c_1 and c_2 and adds them homomorphically such that the output $c_3 \leftarrow SHE.Enc(params, pk, m_1 + m_2)$, where m_1 and m_2 are the plaintext corresponding to the input ciphertexts c_1 and c_2 respectively and finally e) multiplication $SHE.Mult(pk, c_1, c_2)$ which multiplies homomorphically two ciphertexts c_1 and c_2 such that the output $c_3 \leftarrow SHE.Enc(params, pk, m_1 * m_2)$.

In this paper, we use the LFHE encryption scheme to compute squared Euclidean distance over encrypted databases, which is a key building block of our secure anonymization protocol. The details of LFHE cryptosystem can be found at [10].

E. Differential Privacy

Differential privacy is a more recent development in the field of privacy-preserving data publishing and data mining. Achieved by adding randomness to the data, differential privacy renders individuals' data and data mining outputs statistically indistinguishable, thereby protecting individuals' privacy [11]. Differential privacy has been shown to provide strong guarantees against auxiliary information attacks [12], [13], and in recent years has been adopted by large corporations when collecting/publishing sensitive data [14], [15], [16].

A mechanism M is said to be ϵ-differentially private if adding or removing a single data item in a database only affects the probability of any outcome within a small multiplicative factor. Formally, a randomized mechanism M is ϵ-differentially private if for all data sets D_1 and D_2 differing on at most one element, and all $S \subseteq Range(M)$ then

$$Pr[M(D_1) \in S] \leq exp(\epsilon) \cdot Pr[M(D_2) \in S]$$

F. Our Contributions

In this work we build a secure privacy-preserving data publishing workflow over encrypted datasets. The workflow consists of five major components. Figure 1 illustrates the steps of the workflow. The workflow expects as input the encrypted data as well as encrypted meta-data, such as the dictionaries to be used for masking or encrypted parameters for differential privacy. In Section V we present the details of our secure protocols.

1) **Secure Privacy Vulnerability Identification**. This step detects direct identifiers and combinations of attributes-values (quasi-identifiers) [2] that lead to high re-identification risk. The detection is based on the attribute values. In Section V-A we illustrate how direct and quasi-identifiers can be obtained from an encrypted database.
2) **Secure Data Masking**. This component protects the direct identifiers detected by the privacy vulnerability identification component. The values of direct identifiers can be replaced with fictionalized values or redacted; the action taken is based on a pre-defined configuration.
3) **Secure k-anonymity and differential privacy**. This component protects the quasi-identifiers, by applying algorithms with strong security guarantees, such as differential privacy and k-anonymity. Here data are generalized and/or suppressed and/or perturbed so that the re-identification risk becomes smaller than a pre-specified threshold.
4) **Risk assessment**. This component assesses the risk associated with the dataset. It is an additional step during exploratory phases in which expert assessors and policymakers are still evaluating what additional privacy constraints to apply, in addition to what is required by the current legislation.
5) **Utility assessment**. This component allows the estimation of the loss in utility caused by the de-identification/ anonymization process.

IV. Related work

L. Sweeney et al. [2] introduced the concept of k-anonymity and how it can protect against re-identification attacks via creating indistinguishable records. Khaled El Emam et al. [17] proposed a way to achieve globally optimal k-anonymity. LeFevre et al. [18] proposed Mondrian as an approach to achieve good balance between generalization and information loss for multidimensional datasets. These works, along with numerous others that present optimal solution to achieve k-anonymity, try to prevent re-identification attacks through generalization. All these approaches work on unmodified data and they do not include the notion of anonymity over encrypted datasets.

Achieving k-anonymity using clustering is not a new concept. Bertino et al. [19] proposed an efficient k-anonymization algorithm called k-member, which is useful in identifying required generalization to apply k-anonymity to a given dataset. Loukides and Shao [20] propose novel clustering criteria that treat utility and privacy on equal terms and propose sampling-based techniques to optimally set up its parameters. Aggarwal et al. [21] use a personalized clustering algorithm in order to provide a level of anonymity to the individuals recorded in the dataset. All of the proposed algorithms require direct access to the data and do not operate over encrypted data.

Jiang and Clifton [22] propose a secure distributed framework for achieving k-anonymity. Their paper describes a method to locally anonymize dataset so that the joined dataset will be k-anonymous. A two-party secure distributed framework is developed which can be adopted to design a secure protocol to compute k-anonymous data from two vertically partitioned sources. This framework does not apply to encrypted data shared in an hybrid cloud infrastructure. Jiang and Atzori [23] propose a privacy-preserving strategy to mine k-anonymous frequent item sets between two, or more, parties. The proposed algorithm operate on encrypted data to extract insights. The original data are not modified and they are still not compliant with any privacy model after the application of the proposed algorithm.

Differential privacy and homomorphic encryption has been considered previously. In [24], *differentially private encryption schemes* were considered as a way to prevent leakage of information. The authors proposed the Encrypt+DP concept, that imposes differential privacy on the decryption process, rendering it a stochastic process that not always be correct. They also propose DP-then-Encrypt, whereby noise satisfying differential privacy is first added to the data before being encrypted. Both of these schemes are different from the one presented in this paper, as we achieve differential privacy on encrypted data, without having to see the plaintext and without having to decrypt the ciphertext.

The work that is closely related to our paper is the approach proposed by Liu et al. [25]. In this paper a method for performing k-means over homomorphic encrypted data is presented. The paper uses a specific encryption scheme. The main difference is that their approach does not extend to k-anonymity and that the execution scenario described in their approach assumes that the clustering algorithm is performed in a single VM. Furthermore, our work extends to vulnerability identification, masking and differential privacy.

PRIvacy Masking and Anonymization (PRIMA) [26] provides several features for the strategy design and enforcement of data privacy in production grade systems. PRIMA aims to guide decision makers through the data de-identification process while minimizing required input. PRIMA operates on a different trust model, where the data are anonymized before reaching or in the cloud environment, and has no ability to work on encrypted data.

V. Secure Protocols

As described in Section III-B, all the protocols proposed in the paper are considered in the two-party Honest-but-Curious cloud setting. The Data Owner (*DO*) has a plaintext database table $\mathcal{T}$ consisting of *N* data points $\{t_1, t_2, \cdots, t_N\}$. Each data point is a d dimensional value, i.e. $t_i = \{t_i^1, t_i^2, \cdots, t_i^d\}$. Furthermore, let the domain of plaintext space be $\mathcal{P}$ and the domain of ciphertext space be $\mathcal{R}$. The *DO* calls the *KeyGen* function of the SHE algorithm to get the public key and secret key pair (*pk, sk*). Next, the *DO* encrypts the plaintext database $\mathcal{T}$ using *pk* to generate an encrypted database $\mathcal{T}^*$ such that $t_i^* = \{Enc_{pk}(t_i^1), Enc_{pk}(t_i^2), \cdots, Enc_{pk}(t_i^d)\}$. Then, the *DO* shares *pk*, $\mathcal{T}^*$ and the identification threshold k with Party $P1$ and *sk* with Party $P2$.

A. Privacy Vulnerability Identification

The privacy vulnerability identification process explores the combinatorial space of data attributes and aims to identify direct and quasi-identifiers – value sets that appear fewer times than a pre-defined identification threshold k. The process starts by inspecting single attributes and tries to find values that appear fewer than k times. All attributes detected to have values appearing fewer than k times are reported as direct identifiers. In our example, each name value appears only once, thus the name attribute is a direct identifier. The process then starts to inspect pairs of attributes that are not direct identifiers, then the algorithm proceeds to inspect combinations of three identifiers and so on.

Naïve exploration of the entire combinatorial space is infeasible for a large number of attributes since for d attributes 2^d combinations need to be checked. Pruning techniques are employed to avoid the exploration of the full space. Pruning can be applied in the following two scenarios. First, if an attribute, or a set of attributes, T is a quasi-identifier then all the combinations of attributes including this attribute, or set of attributes, are also quasi-identifiers. Second, if T is not a quasi-identifier then all subset combinations of T are not quasi-identifiers. This leads to a dramatic reduction of the number of combinations of attributes that need to be checked, thus resulting in a significant improvement in execution time of the protocol. Refer to [27] for further discussion of the impact of pruning in the identification of privacy vulnerabilities.

We use Algorithm 1 to identify direct identifiers. Since our encryption function is non-deterministic, a direct comparison of encrypted attribute values will not be helpful. So for each encrypted value in the attribute, $P1$ computes its difference from the remaining $N-1$ values, where N is the number of tuples in the dataset (the difference will be zero if there is a value match within the attribute). Then $P1$ multiples these differences with random values in the matrix $\mathcal{R}^*$ and send the computed matrix $\mathcal{M}^*$ to party $P2$. This is shown in Steps $1-7$ of Algorithm 1. Next, for each attribute, $P2$ counts the number of zeros for every encrypted value, if there is an encrypted value for which the count of zeros is less than k, then this attribute is a direct identifier. $P2$ tracks the direct identifiers by setting to 1 the corresponding index in vector $\mathcal{V}$. This is shown in Steps $8-18$ in Algorithm 1. Then, $P2$ returns the vector $\mathcal{V}$ to Party $P1$.

Algorithm 1 ***Direct Identifier (DI)***

Require: $P1$ has $\mathcal{T}^*$ and pk; $P2$ has sk
Ensure: $P1$ learns which attributes of $\mathcal{T}^*$ are direct identifiers

$P1$:
1) **for** $i = 1$ to d **do**
2) Compute $\mathcal{D}^*_{uv} = t^{i*}_u - t^{i*}_v \ \forall u, v \in N$
3) Sample a matrix $\mathcal{R}$ of size $N \times N; R_{uv} \in_R \mathcal{P}$
4) Compute $\mathcal{R}^* \ni \mathcal{R}^*_{uv} \leftarrow Enc_{pk}(\mathcal{R}_{uv})$
5) Evaluate Hadamard product $\mathcal{M}^*_i \leftarrow \mathcal{D}^* \circ \mathcal{R}^*$
6) **end for**
7) **Send** $\mathcal{M}^*$ to $P2$

$P2$:
8) Create a vector $\mathcal{V}$ of length d
9) Set $\mathcal{V}_i = 0 \ \forall i \in d$
10) **for** $i = 1$ to d **do**
11) **for** $j = 1$ to N **do**
12) count = 0
13) count += $\mathcal{I}(Dec_{sk}(\mathcal{M}^*_{ijl}) == 0) \ \forall l \in N$
14) **if** count $< k$ **then**
15) $\mathcal{V}_i = 1$; break;
16) **end if**
17) **end for**
18) **end for**
19) **Return** $\mathcal{V}$ to $P1$

Similarly, we utilize the matrix $\mathcal{M}^*$ computed in Algorithm 1 along with the pruning mechanism described earlier to find quasi-identifiers.

B. Data Masking

Data masking is applied when there is need to replace the original values with fictionalized ones. If we operate on non-encrypted data, then multiple options are available: format-preserving and semantic-preserving masking, compound masking as well as some generic masking providers, like nullification, hashing, randomization, truncation and numeric value shifting. Format-preserving masking dictates that the masked value will have the same format as the original one. Semantic-preserving masking ensures that parts of the original value that contain auxiliary information need to be maintained.

Since we operate on encrypted data, not all options are available. Semantic-preserving and format-preserving masking cannot be applied since they require access to the original value unless the data owner encrypts only the unique parts of the value. This requires additional metadata so the cloud environment knows how to handle each value (e.g. offsets and lengths of encrypted portions of the value). However, in this paper, we apply Masking of dictionary-based entities. Entities like names, organization, cities, countries and many more rely on dictionaries to perform format-preserving masking. For example, if we want to replace a name with another one, then we pick a random name from its dictionary. We can apply the same operation over encrypted data. The user uploads a fully encrypted dictionary for the attribute. Then we select a random value from the encrypted dictionary and replace the value. However, the encrypted version of the dictionary needs to be immune to inference attacks. For specific attributes, an attacker can infer the attribute type and values based on cardinality attacks. As an example, a dictionary of two entries could potentially be a gender dictionary. To alleviate this problem, we can append copies of its values to the dictionary. Since the encryption is non-deterministic, we can increase the cardinality of the values infinitely.

C. Differential Privacy

In this section we demonstrate achieving differential privacy on encrypted data with select mechanisms. To implement the differential privacy mechanisms on numerical data given in Section III-E, some information on the data is required, such as the diameter $diam$ for the Laplace mechanism, and the binary values for the binary mechanism. This information must somehow be provided to the CSP for the mechanisms to be implemented. As we will show later, it is sufficient for this information to be available in encrypted form. Making such information about the data publicly available may reveal unwanted information and lead to inference attacks (e.g. attribute type, extreme values, etc.), and is therefore not desirable.

Before the data is encrypted, the DO selects lower and upper bounds $l \leq u \in \mathbb{R}$ that are independent of the data. This may be performed by examination of the attribute in question (e.g. a person's age), or by other means, but must not be a function of the data (i.e. the range of the data). In the case of binary-valued data, l and u will simply be the two binary values. Non-informative bounding, as discussed in [28], ensures no additional privacy leakage, allowing the entire privacy budget ϵ to be spent on the differential privacy mechanism itself. These bounds must then be encrypted and stored securely alongside the dataset in question. For the remainder of this subsection, we will refer to the encrypted values $l^* = Enc_{pk}(l)$ and $u^* = Enc_{pk}(u)$.

Below, we detail how we can use $SHE.Add$ and $SHE.Mult$ (Section III-D) to render the encrypted values differentially private, without having to decrypt the original values. This process is then applied independently to each value of interest.

- **Laplace mechanism**: To achieve differential privacy, the required scale factor is $b = \frac{diam}{\epsilon}$. In determining the noise to add to the data, we sample $L \sim \text{Lap}(0, b)$, and add this to the encrypted value. In generating L, we

draw a value r at random from a uniform distribution on $\left[-\frac{1}{2}, \frac{1}{2}\right]$, $r \sim \text{Unif}\left(-\frac{1}{2}, \frac{1}{2}\right)$, and use the inverse of the cumulative probability distribution of $\text{Lap}(0, b)$ to find

$$L = -b\,\text{sgn}(r)\log(1 - 2|r|),$$

where $\text{sgn}(\cdot)$ is the signum function, defined by

$$\text{sgn}(r) = \begin{cases} 1, & r > 0, \\ 0, & r = 0, \\ -1, & r < 0. \end{cases}$$

We cannot calculate the plaintext L, since $diam$ can only be calculated in encrypted form. We can, however, calculate its ciphertext $L^* = Enc_{pk}(L)$ as :

$$L^* = Mult_{pk}\left(diam^*, Enc_{pk}\left(-\frac{1}{\epsilon}\,\text{sgn}(r)\log(1 - 2|r|)\right)\right)$$

where $diam^* = Enc_{pk}(diam)$ is given by:

$$diam^* = Add_{pk}(u^*, Mult_{pk}(Enc_{pk}(-1), l^*)).$$

The resultant value that is stored is therefore

$$Add_{pk}(d_i^*, L^*).$$

- **Binary mechanism**: If the original data d is binary, the binary mechanism can be used. This time we draw r at random from the unit interval $[0, 1]$. If $r \leq \frac{e^\epsilon}{1+e^\epsilon}$, then the value d_i^* remains unchanged. However, if $r > \frac{e^\epsilon}{1+e^\epsilon}$, then we flip d_i^* by setting $d_i' = u + l - d_i^*$. Again, this can be done without knowing the value of d_i^*, and by only knowing the ciphertexts l^* and u^*. We can implement this using $SHE.Mult$ to get $-d_i^*$, and then using $SHE.Add$ as before.
  In the case of the value being flipped, the value that is stored is

$$Add_{pk}(Mult_{pk}(d_i^*, Enc_{pk}(-1)), Add_{pk}(l^*, u^*)).$$

D. k-Anonymization

In this paper, we implement anonymization algorithms that support the *k-anonymity* privacy guarantee as formally defined in [2]. Given the identification threshold k, achieving *k-anonymity* over encrypted data is a three-step process. First, we securely partition the data into clusters. In this paper, we specifically apply k-means clustering algorithm over encrypted data and use *Squared Euclidean Distance* (SED) metric to calculate the proximity of values in their respective feature space. Second, to ensure that each cluster has at least k members we apply data suppression and re-assignment techniques as presented in Sections V-D2 and V-D3, respectively. And finally, we securely anonymize the original data values to a representative one. For the numerical attributes, we replace them with the cluster centroid. For each categorical attribute, we replace them with the common ancestor of the attribute value based on the respective generalization hierarchy.

Algorithm 2 outlines the procedure to compute k-anonymized data for a database table $\mathcal{T}$ having d attributes.

Algorithm 2 ***Secure k-Anonymization***

Require: $P1$ has T^*, k, *rounds* and *th*
Ensure: $P1$ computes the k-anonymized dataset $T^{*'}$
1: $k' \leftarrow N/k$
2: $\mathcal{C}^* \leftarrow \{c_1^*, \cdots, c_{k'}^*\}$, $c_i^* \in_R T^*$
3: *loop* $\leftarrow 0$
4: **while** *loop* $<$ *rounds* **do**
5: **for** $i = 1$ to N **do**
6: **for** $j = 1$ to k' **do**
7: $\mathcal{D}_{ij}^* \leftarrow \sum_{l=1}^{d} (\mathcal{T}_{il}^* - \mathcal{C}_{jl}^*)^2$
8: **end for**
9: **end for**
10: **for** $i = 1$ to N **do**
11: $\mathcal{I}_i^* \leftarrow$ ***ComputeMinIndex***$(\mathcal{D}_i^*)$
12: **end for**
13: $\mathcal{C}^* \leftarrow$ ***RecomputeClusterCentres***$(\mathcal{I}^*)$
14: *loop* $=$ *loop* $+1$
15: **end while**
16: $\mathcal{J} \leftarrow$ ***Non-kClusters*** $(\mathcal{I}^*)$
17: $\mathcal{I}^*, \mathcal{J} \leftarrow$ ***SupressClusters***$(\mathcal{I}^*, \mathcal{J}, th)$
18: $\mathcal{I}^* \leftarrow$ ***ReAssignClusters***$(\mathcal{I}^*, \mathcal{J})$
19: $T^{*'} \leftarrow$ ***AnonymizeClusters***$(T^*, \mathcal{I}^*)$

The algorithm takes as input the encrypted table $T^* = Enc_{pk}(T)$, the identification threshold k, the number of iterations of clustering algorithm *rounds* and the suppression threshold *th* at $P1$. Further $P2$ has the secret key sk. In the end, the algorithm outputs the corresponding k-anonymized database table $T^{*'}$. In the following sections, we will describe different steps of Algorithm 2.

1) Data Clustering: To produce a k-anonymized database, $P1$ can at most find $k' = N/k$ clusters, each having at-least k members, where N is the total number of tuples in the table $\mathcal{T}^*$. In Step 2 we randomly select k' tuples as the initial cluster centres and in Steps 4 – 9, squared Euclidean distance is computed for all the tuples from all the cluster centres using the homomorphic properties of the SHE encryption scheme and the results are stored in matrix $\mathcal{D}^*$.

Next, in Step 11 new cluster assignment for all the tuples are identified by calling the function ***ComputeMinIndex*** described in Algorithm 3. This algorithm takes as input a vector $\mathcal{D}_i^*$ of size k' and returns an encrypted vector of size k' having the encryption of value 1 at the index of nearest cluster centre and encryption of 0 at all other positions. In Steps 1 – 2, $P1$ selects a monotonic increasing polynomial $poly(x)$, such that $poly(x_1) \geq poly(x_2)$ iff $x_1 \geq x_2$ and homomorphically evaluate the polynomial $poly(x)$ over all the values in vector $\mathcal{D}_i^*$ and computes $\mathcal{D}_i^{*'}$. Next, in Step 3 $P1$ selects a pseudo-random permutation (PRP) π and permutes the vector $\mathcal{D}_i^{*'}$. Finally, it sends the vector $\mathcal{D}_i^{*''}$ to $P2$. Then, in Step 5, $P2$ decrypts the vector $\mathcal{D}_i^{*''}$ and in Steps 6 – 12 identifies the index of the minimum value element in vector $\mathcal{D}_i^{''}$. In Steps 13 – 14, $P2$ initializes a vector $\mathcal{I}_i^{'}$ of size k' with 0 and then sets the value at the index identified above to 1. Then, in Step 15 $P2$ encrypts the vector $\mathcal{I}_i^{*'}$ and sends it to $P1$. Note that the vector $\mathcal{I}_i^{*'}$ contains the encryption of 1 at exactly one position corresponding to the nearest cluster center, but since the vector $\mathcal{D}_i^{*''}$ was initially permuted by $P1$, hence $P2$ does not learns

the correct cluster assignment. Next, $P1$ applies the inverse permutation π^{-1} to $\mathcal{I}_i^{*'}$ in Step 17. Note, $P1$ has received the cluster assignment for tuple T_i^* but it does not learn the cluster to which this tuple is assigned, since all the entries in the vector $\mathcal{I}_i^*$ are encrypted using non-deterministic encryption. Similarly, $P1$ receives the encrypted cluster assignment for every tuple in the encrypted table $\mathcal{T}^*$

Algorithm 3 ***ComputeMinIndex***

Require: $P1$ has $\mathcal{D}_i^*$
Ensure: $P1$ gets an encrypted vector $\mathcal{I}_i^*$, having value 1 at the index of nearest cluster center and 0 otherwise

$P1$:
1) Choose a polynomial $poly(x) \leftarrow a_0 + a_1 \cdot x + \cdots + a_q \cdot x^q$, $q, a_l \in_R \mathbb{N}\ \forall l \in \{0, q\}$
2) $\mathcal{D}_{ij}^{*'} \leftarrow poly(\mathcal{D}_{ij}^*)\ \forall j \in \{1, k'\}$
3) $\mathcal{D}_i^{*''} \leftarrow \pi_s(\mathcal{D}_i^{*'})$, $\pi : \{0,1\}^{k'} \times \{0,1\}^s \rightarrow \{0,1\}^{k'}$
4) **Send** $\mathcal{D}_i^{*''}$ to $P2$

$P2$:
5) $\mathcal{D}_{ij}^{''} \leftarrow Dec_{sk}(\mathcal{D}_{ij}^{*''})\ \forall j \in \{1, k'\}$
6) $min\_ind \leftarrow 1$, $min\_val \leftarrow \mathcal{D}_{i1}^{''}$
7) **for** $j = 2$ to k' **do**
8) **if** $\mathcal{D}_{ij}^{''} < min\_val$ **then**
9) $min\_val \leftarrow \mathcal{D}_{ij}^{''}$
10) $min\_ind \leftarrow j$
11) **end if**
12) **end for**
13) $\mathcal{I}_i^{'} \leftarrow \vec{0}_{k'}$
14) $\mathcal{I}_i[min\_ind] \leftarrow 1$
15) $\mathcal{I}_i^{j*'} \leftarrow Enc_{pk}(\mathcal{I}_i^{j'})\ \forall j \in \{1, k'\}$
16) **Return** $\mathcal{I}_i^{*'}$ to $P1$

$P1$:
17) $\mathcal{I}_i^* \leftarrow \pi_s^{-1}(\mathcal{I}_i^{*'})$

Next, in Step 13 of Algorithm 2 $P1$ calls the function ***RecomputeClusterCentres*** to recompute the cluster center representatives. Algorithm 4 provides the details of this function. It takes as input the encrypted cluster assignment $\mathcal{I}^*$ and returns new cluster centres $\mathcal{C}^*$. In Steps 3 – 6, Algorithm 4 first computes the encrypted cluster count and sum for every cluster. Note in Step 5, tuple i is added to sum_j^* if and only if it belongs to a cluster j, since $\mathcal{I}^*{}_{ij} = Enc_{pk}(1)$ if tuple i belongs to a cluster j, else $\mathcal{I}^*{}_{ij} = Enc_{pk}(0)$. Next, in Step 7, $P1$ selects a random value u_j and multiplies it with the cluster count for cluster j. This step produces a one time pad blinding of the cluster count. Similarly in Step 8 the corresponding cluster sum is blinded with a random value v_j. All the above operations are performed using the homomorphic properties of the SHE encryption scheme. Now, $P1$ sends the blinded cluster count and sum to $P2$. In Steps 13 – 15 $P2$ decrypts and divides the corresponding cluster sum and count and re-encrypts the results. $P2$ then sends the encrypted divisions to $P1$. In Steps 19, $P1$ multiplies the cluster divisions with division of the random values selected in Steps 7 and 8. This step removes the randomness and $P1$ gets the updated cluster centres encrypted under the public key pk. The clustering process is repeated for *rounds* number of iterations in order to converge the cluster centres.

2) Cluster Suppression: The above clustering process does not provide guarantee on the number of members in each cluster. In order to achieve k-*anonymity*, we make sure that

Algorithm 4 ***RecomputeClusterCentres***

Require: Cluster assignment $\mathcal{I}^*$
Ensure: Returns new cluster centres $\mathcal{C}^*$

$P1$:
1) **for** $j = 1$ to k' **do**
2) $count_j^* \leftarrow Enc_{pk}(0)$; $sum_j^* \leftarrow Enc_{pk}(0)$
3) **for** $i = 1$ to N **do**
4) $count_j^*$ += $\mathcal{I}_{ij}^*$
5) sum_j^* += $\mathcal{I}_{ij}^* * \mathcal{T}_i^*$
6) **end for**
7) $count_j^{*'} \leftarrow count_j^* * Enc_{pk}(u_j)$; $u_j \in_R \mathbb{N}$
8) $sum_j^{*'} \leftarrow sum_j^* * Enc_{pk}(v_j)$; $v_j \in_R \mathbb{N}$
9) **end for**
10) Choose a PRP $\pi_s : \{0,1\}^{k'} \times \{0,1\}^s \rightarrow \{0,1\}^{k'}$
11) $sum^{*''} \leftarrow \pi_s(sum^{*'})$; $count^{*''} \leftarrow \pi_s(count^{*'})$
12) **Send** $sum^{*''}$ and $count^{*''}$ to $P2$

$P2$:
13) **for** $j = 1$ to k' **do**
14) $div_j^{*''} \leftarrow Enc_{pk}(Dec_{sk}(sum_j^{*''})/Dec_{sk}(count_j^{*''}))$
15) **end for**
16) **Return** $div^{*''}$ to $P1$

$P1$:
17) $div^{*'} \leftarrow \pi_s^{-1}(div^{*''})$
18) **for** $j = 1$ to k' **do**
19) $\mathcal{C}_j^* \leftarrow Enc_{pk}(u_j/v_j) * div_j^{*'}$
20) **end for**

each cluster has at-least k members. To achieve this, we apply a post-processing phase on the output clusters. In Step 16, Algorithm 2 calls function ***Non-kClusters*** to identify the clusters having fewer than k members. The details of this function is described in Algorithm 5. It takes as input the encrypted cluster assignment ($\mathcal{I}^*$) and encrypted cluster count ($count^*$) and returns a vector $\mathcal{J}$ of size k' indicating the clusters which need further processing. In Step 1 $P1$ selects a monotonically increasing polynomial $poly(x)$ and homomorphically evaluate the polynomial over the encrypted cluster counts and computes $count^{*'}$. Next, in Step 3 a PRP π_s is selected and used to permute the order of $count^{*'}$. Then in Step 4, the identification factor k is encrypted and masked with the polynomial $poly(x)$. Next both $count^{*'}$ and $mark^{*'}$ are sent to $P2$. Party $P2$ initializes a vector $\mathcal{J}^{'}$ of size k' with value 0. Then in Steps 7 – 13, it sets the entry of vector $\mathcal{J}^{'}$ to 1 if the count is less than k and finally returns $\mathcal{J}^{'}$. Party $P1$ then applies the inverse permutation π_s^{-1} and retrieves the vector $\mathcal{J}$.

Now, if the number of members is more than k (i.e. if $\mathcal{J} == 0$), then the cluster is left unmodified. If, however, the cluster contains fewer than k members we follow two strategies. First, we check if we can suppress the cluster and remove its points from the final anonymized output. For suppression, a threshold is required to specify the maximum percentage of the total points we are allowed to remove. If suppression is not allowed or we have reached the suppression threshold, then we apply cluster re-assignment techniques, in which nearest clusters are identified to merge with the non-k clusters. The cluster re-assignment strategies are described in Section V-D3.

3) Cluster Re-assignment: Once the suppression threshold is reached, the remaining non-k clusters are re-assigned to nearest clusters. The merging of two clusters is easily done

Algorithm 5 *Non-kClusters*

Require: Cluster assignment $\mathcal{I}^*$, Cluster counts $count^*$
Ensure: $P1$ learns Non-k Clusters $\mathcal{J}$

1: $P1$:
1) Choose a polynomial $poly(x) \leftarrow a_0 + a_1 \cdot x + \cdots + a_q \cdot x^q$, $q, a_l \in_R \mathbb{N}\ \forall l \in \{0, q\}$
2) $count_j^{*'} \leftarrow poly(count_j^*)$, $\forall j \in [1, k']$
3) $count^{*''} \leftarrow \pi_s(count^{*'})$ {π_s is a PRP}
4) $mark^* \leftarrow Enc_{pk}(k)$; $mark^{*'} \leftarrow poly(mark^*)$
5) **Send** $mark^{*'}$ and $count^{*''}$ to $P2$

2: $P2$:
6) Initialize vector $\mathcal{J}^{'} \leftarrow \vec{0}_{k'}$
7) $mark^{'} \leftarrow Dec_{sk}(mark^{*'})$
8) **for** $j = 1$ to k' **do**
9) $count_j^{''} \leftarrow Dec_{sk}(count_j^{*''})$
10) **if** $count_j^{''} < mark^{'}$ **then**
11) $\mathcal{J}_j^{'} \leftarrow 1$
12) **end if**
13) **end for**
14) **Return** $\mathcal{J}^{'}$ to $P1$

3: $P1$:
15) $\mathcal{J} \leftarrow \pi_s^{-1}(\mathcal{J}^{'})$

using the encrypted cluster assignment vector. For example, say we want to merge cluster j in cluster i, we can achieve this by adding the jth component of every encrypted cluster assignment vector $\mathcal{I}^*$ to its ith component. Further, merging the cluster with its nearest one does not guarantee k-anonymity. For example, let's assume we want to achieve 3-anonymity and a cluster has one member. Its nearest cluster also has a single member so merging them will not result in a cluster with a minimum of 3 elements. Thus, we need to apply the process iteratively until all created clusters have more than or equal to k data points. After cluster re-assignment step all the identified clusters have a minimum of k data points.

4) Data Anonymization: For numerical attributes, we replace them with the cluster centroid. For the categorical attributes, we replace them with the common ancestor of the attribute value based on the respective generalization hierarchy. In order to avoid inference attacks based on the hierarchy structure and cardinality of number of nodes per level, we can employ similar approaches like with data masking dictionaries; we can randomly insert dummy nodes at each level. One approach to calculate the common ancestor for all values is the following. We first calculate the common ancestor between the first value and the second one, let's call it $A_{1,2}$. Then we calculate the common ancestor between $A_{1,2}$ and the third value, $A_{1,2,3}$ and so forth. If at some point one of the common ancestors calculated is the root of the hierarchy, then the calculations stop. This approach requires at maximum O(N) checks and each check requires $O(M)$ operations, where N is the total number of values and M is the height of the generalization hierarchy.

E. Risk and utility assessment

In this Section, we sketch out how various risk and utility assessment algorithms can be implemented on top of encrypted data. Inference-based risk metrics, such as the ones described in [29], [30], [31] rely solely on the size of the equivalence classes and additional external information, such as population size (required) and bias estimation (optional). Thus, it is only required to group the data based on their equivalence class and count the size of each group.

Simple information loss metrics, such as Average Equivalence Class Size (AECS) [18] and discernibility [32], also rely on the equivalence class size to provide a result. Categorical precision [33] relies on the level of generalization applied for each value. This information is computed when we perform the data anonymization step (see Section V-D4). Similarly, generalized loss metric [34] requires the number of leaves for each anonymized value. However, metrics like non-uniform entropy [35] and global certainty penalty [36] require either frequency calculations or knowledge of the data diameter, which can be only aquired with access to the original values.

VI. Security Guarantees

As described earlier both the Parties $P1$ and $P2$ are considered in the Honest-but-Curious security model where both the parties correctly execute the protocol but may try to learn the plaintext value from their view of the encrypted data processing. We also assume that Party $P1$ and Party $P2$ do not collude. Further, Party $P2$ is additionally trusted with the secret key of the SHE encryption scheme. We want to emphasize that this cloud model is not new and has been used in related problem domain [4], [5].

Given above assumptions, informally we will prove that, *the views of Party $P1$ and Party $P2$ does not reveal any useful information about the plaintext database during the execution of secure k-Anonymization protocol*. We will formally prove this statement using Leakage Profile Analysis.

A. Leakage profile at Party $P1$

1) **Direct Identifier:** In Algorithm 1, for each encrypted value in the attribute, Party $P1$ computes its difference from the remaining $N - 1$ values and then multiplies them with a different random value. Both these operations are performed using the homomorphic properties of the SHE scheme. Hence any leakage in this step will break the security guarantee of the underlying SHE encryption scheme.
2) **ComputeMinIndex:** In Algorithm 3, for each entry in vector D_i^*, Party $P1$ evaluates a randomly chosen polynomial using the homomorphic properties of the SHE scheme. Now, since the polynomial evaluation is done over encrypted data, hence the security guarantee of the SHE scheme ensures that there is no leakage to Party $P1$. Next $P1$ chooses a pseudo-random permutation to hide the physical order of elements in vector $D_i^{*'}$. This step further breaks any physical order co-relation between the entries in different D_i^* vectors.
3) **RecomputeClusterCentres:** In Algorithm 4, Party $P1$ computes the number of data points in every cluster and the corresponding cluster sum. To compute the count, $P1$ applies addition operation over the encrypted cluster assignment vector $\mathcal{I}^*$ and

to compute the cluster sum, $P1$ first multiplies the encrypted vector $\mathcal{I}^*$ and encrypted data points $\mathcal{T}_i^*$ and then adds the encrypted values. All the operations in this algorithm are performed over encrypted data using the properties of the SHE scheme, hence the security guarantee of the SHE scheme ensures that there is no possible leakage to Party $P1$.

4) **Non-kClusters:** In Algorithm 5, Party $P1$ evaluates a randomly chosen polynomial over the encrypted cluster count values and the encrypted identification factor k, using the homomorphic properties of the SHE scheme. Hence, any leakage in these steps will break the security guarantee of the SHE scheme.
5) **Suppress and Reassign Clusters:** In this step, Party $P1$ only replaces some encrypted cluster count values with random values or adds two encrypted vectors , hence there is no extra leakage.

Theorem VI.1. Security Guarantee for Party $P1$ **:** The secure k-Anonymization protocol leaks no information to Party $P1$ except that it learns if an attribute is a direct identifier and number points in the non-k clusters. In particular, Party $P1$ does not gain any knowledge about the encrypted data points, the difference between two data points, the cluster to which a data point is assigned and the cluster centre representatives.

B. Leakage profile at Party $P2$

1) **Direct Identifier:** In Algorithm 1, Party $P2$ decrypts the encrypted matrix $\mathcal{M}$ using the secret key sk. But since Party $P1$ has multiplied each entry of the matrix $\mathcal{M}$ with a different random value before sending it to Party $P2$, the decrypted matrix $\mathcal{M}$ effectively contains random values. Hence the only leakage in this step is that Party $P2$ learns if two values in the attribute are equal since the corresponding decrypted difference will be 0 but nothing is revealed about the original data points or the difference between two unequal values.
2) **ComputeMinIndex:** In Algorithm 3, for every data point, Party $P2$ receives a vector $D_i^{*''}$ of size k'. The entries in this vector are the output of encrypted polynomial evaluation *poly(x)* over the distance of the data point i from the cluster centres. Further, the order of elements in the vector is permuted using a secure pseudo-random permutation, hence the exact identity of cluster centres associated with any given difference value is hidden from Party $P2$.

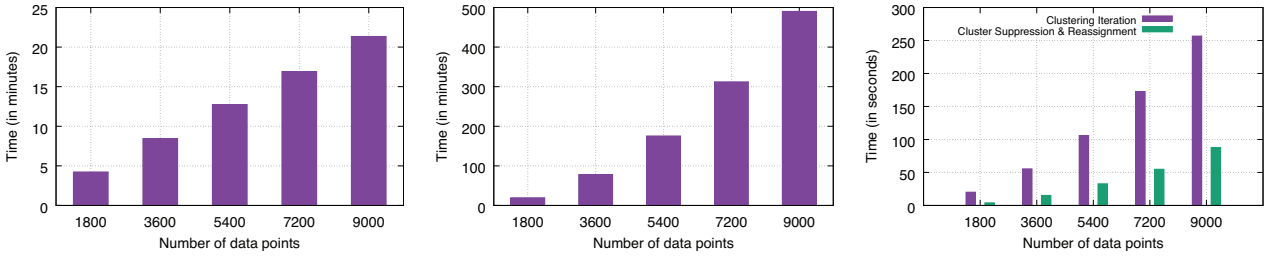
   Party $P2$ decrypts the entries in the vector $D_i^{*''}$ and since the polynomial *poly(x)* is order preserving, hence Party $P2$ can sort the decrypted values and identify the index of the nearest cluster centre. In the technical report [37], we prove that recovering the plaintext distances form $D_i^{''}$ is computationally infeasible for Party $P2$. The only possible leakage to Party $P2$ in this round is the presence of such points in the database that are equidistant from two or more cluster centres. This is leaked from the presence of identical values in the set $\{poly(d_{i,1}^{''}), poly(d_{i,2}^{''}), \cdots, poly(d_{i,k'}^{''})\}$. However, since the order of the values is randomly permuted by Party $P1$, Party $P2$ cannot map these values back to the original index of either the data point or the corresponding cluster centres in the database.
3) **RecomputeClusterCentres:** In this phase, Party $P2$ gains access (by virtue of decryption) to the following plaintext (but randomized) quantities:
   - Sum of data points nearest in each cluster centre
   - Number of data points in each cluster centre

   We note that since these quantities are multiplicatively randomized by Party $P1$, their actual values are effectively hidden from Party $P2$. It is also worth noting that the randomization used is different for each cluster, implying that Party $P2$ cannot hope to leverage any sharing/re-use of randomization across different cluster centres to gain additional information about the sum or number of data points for any given cluster centre.
4) **Non-kClusters:** In Algorithm 5, Party $P2$ gets access to the plaintext (but masked) of the anonymization factor k and the number of data points in each cluster center. But since Party $P1$ evaluates a random polynomial *poly(x)* over their encrypted values before sending them, hence Party $P2$ does not learn the actual anonymization factor k and the number of data points in each cluster centre. A similar proof as shown in Step 2 above can be presented.
5) **Suppress and Reassign Clusters:** In this step, Party $P2$ receives a permuted vector of size k' having the encrypted counts of the number of elements in non-k clusters padded with some fake values. Hence after decryption of this vector Party $P2$ cannot identify the number of elements in the non-k clusters, since we have picked a secure pseudo-random permutation, which is computationally difficult to invert, implying that the exact identity of cluster centres associated with any cluster count is hidden from Party $P2$.

Theorem VI.2. Security Guarantee for Party $P2$ **:** The secure k-Anonymization protocol leaks no information to Party $P2$ except that it only learns if an attribute is a direct identifier but does not gain any knowledge about the encrypted data points or the cluster to which a data point is assigned and the cluster centre representatives.

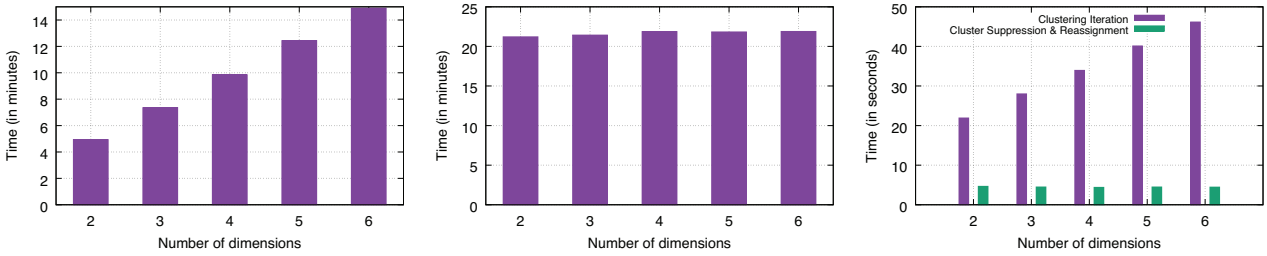
VII. Performance

In this section, we empirically evaluate the performance of our protocols. The experimental setup consists of three machines, representing the Data Owner, Party $P1$ and Party $P2$. The configuration of machines representing Party $P1$ and Party $P2$ is: 4 core 2.8 GHz processors, 64 GB RAM running Ubuntu 16.04 LTS; the configuration of the machine representing Data Owner is: 4 core 2.8 GHz processors, 8 GB RAM running Ubuntu 16.04 LTS. We use the HELib [38] library to encrypt the data using LFHE. Specifically, for HELib we set (i) $p = 1099511627689$, a large prime between 2^{40}



(a) Data Encryption Time (b) Time to check an identifier (c) Time taken per iteration of clustering

Fig. 2: Execution time for varying number of data points (each having 2 dimensions)



(a) Data Encryption Time (b) Time to check an identifier (c) Time taken per iteration of clustering

Fig. 3: Execution time for varying number of dimensions with 1800 data points

and 2^{87}, (ii) the maximum depth to 10 and (iii) the security parameter to 128.

The two parameters affecting the performance of our protocols are the number of data points and the number of dimensions in the data. To study the independent effect of each of these parameters on our protocols, we use simulated data. We generated two datasets, one with a varying number of data points (results shown in Figure 2) and one with a varying number of dimensions (results shown in Figure 3). The data were generated using a uniform distribution. We repeated each experiment multiple times with a newly generated dataset. The average time across these experiments is reported here.

LFHE allows SIMD operations by packing multiple plaintext values into a single structure and then encrypting them together into a single ciphertext. We encrypt each dimension of the data point independently. For each dimension, we pack data from multiple data points into a single structure and then encrypt this structure to get a single ciphertext. This ciphertext is then outsourced to Party $P1$. The time taken to encrypt the plaintext is shown in Figure 2a and Figure 3a. These figures clearly show that the data encryption time scales linearly with the number of data points and the number of dimensions.

The second major step in our protocols is to check if a particular combination of dimensions is a privacy vulnerability identifier or not. The results are shown in Figure 2b and Figure 3b. From the figures, it is clear that the time taken to identify a quasi-identifier scales linearly with the number of data points and is independent of the number of dimensions (since, the most computationally heavy step is decryption of distance at Party $P2$, which is independent of the number of dimensions).

Once a quasi-identifier is identified, the next step is to cluster the data in the quasi-identifiers. Furthermore, after clustering, we use the "Cluster to Cluster" re-assignment strategy to eliminate non-k clusters. Figure 2c and Figure 3c show that both the above operations scale linearly with the number of data points. The number of dimensions has a negligible effect on the cluster reassignment (again, the most expensive step being decryption of inter-cluster distance at Party $P2$, which is independent of the number of dimensions).

Note that number of dimension combinations to check, clustering as well as cluster re-assignment is data dependent. To remove this data dependence from the performance evaluation we report the average time taken for them. The above performance evaluation shows that our protocols scale linearly with the number of data point as well as the number of dimensions in the dataset.

VIII. Conclusions

This paper presents a set of secure algorithms on how to apply anonymization over homomorphically encrypted databases. It does not focus on a single anonymization approach but touches various components that are required for end-to-end privacy. It demonstrated how to achieve uniqueness discovery, data masking, differential privacy and k-anonymity over encrypted data without leaking information about original values. Feasibility of this solution is shown by empirical evaluation. This work is the first to perform several techniques, like vulnerability assessment, differential privacy and k-anonymity, over encrypted datasets which means there is room for improvement and future work, especially on the performance and optimization side.

REFERENCES

[1] "Comply with gdpr," https://ibm.biz/Bd2yUK, April 2017.

[2] L. Sweeney, "K-anonymity: A model for protecting privacy," *Int. J. Uncertain. Fuzziness Knowl.-Based Syst.*, vol. 10, no. 5, Oct. 2002.

[3] S. Bugiel, S. Nurnberger, A. Sadeghi, and T. Schneider, "Twin clouds: An architecture for secure cloud computing," in *Workshop on Cryptography and Security in Clouds (WCSC 2011)*, vol. 1217889, 2011.

[4] Y. Elmehdwi, B. K. Samanthula, and W. Jiang, "Secure k-nearest neighbor query over encrypted data in outsourced environments," in *2014 IEEE 30th International Conference on Data Engineering*, March 2014, pp. 664–675.

[5] M. Kesarwani, A. Kaul, P. Naldurg, S. Patranabis, G. Singh, S. Mehta, and D. Mukhopadhyay, "Efficient secure k-nearest neighbours over encrypted data," in *Proceedings of the 21th International Conference on Extending Database Technology, EDBT 2018*, 2018, pp. 564–575.

[6] N. Grozev and R. Buyya, "Inter-cloud architectures and application brokering: taxonomy and survey," *Software: Practice and Experience*, vol. 44, no. 3, pp. 369–390, 2014.

[7] C. Gentry *et al.*, "Fully homomorphic encryption using ideal lattices." in *STOC*, vol. 9, no. 2009, 2009, pp. 169–178.

[8] P. Paillier *et al.*, "Public-key cryptosystems based on composite degree residuosity classes," in *Eurocrypt*, vol. 99. Springer, 1999, pp. 223–238.

[9] D. Boneh, E.-J. Goh, and K. Nissim, "Evaluating 2-dnf formulas on ciphertexts," in *Theory of Cryptography Conference*. Springer, 2005, pp. 325–341.

[10] Z. Brakerski, C. Gentry, and V. Vaikuntanathan, "(leveled) fully homomorphic encryption without bootstrapping," *ACM Transactions on Computation Theory (TOCT)*, vol. 6, no. 3, p. 13, 2014.

[11] C. Dwork, "Differential privacy," in *Automata, Languages and Programming: 33rd International Colloquium, ICALP 2006, Venice, Italy, July 10-14, 2006, Proceedings, Part II*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 1–12.

[12] S. P. Kasiviswanathan and A. Smith, "A note on differential privacy: Defining resistance to arbitrary side information," *CoRR abs/0803.3946*, 2008.

[13] S. R. Ganta, S. P. Kasiviswanathan, and A. Smith, "Composition attacks and auxiliary information in data privacy," in *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2008, pp. 265–273.

[14] A. Chin and A. Klinefelter, "Differential privacy as a response to the reidentification threat: The Facebook advertiser case study," *North Carolina Law Review*, vol. 90, no. 5, 2012.

[15] Apple Inc., "Apple previews iOS 10, the biggest iOS release ever," http://www.apple.com/newsroom/2016/06/apple-previews-ios-10-biggest-ios-release-ever.html [Accessed: 2016-07-26], June 2016.

[16] U. Erlingsson, V. Pihur, and A. Korolova, "RAPPOR: Randomized Aggregatable Privacy-Preserving Ordinal Response," in *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '14. New York, NY, USA: ACM, 2014, pp. 1054–1067.

[17] K. E. Emam, F. K. Dankar, R. Issa, E. Jonker, D. Amyot, E. Cogo, J.-P. Corriveau, M. Walker, S. Chowdhury, R. Vaillancourt, T. Roffey, and J. Bottomley, "A globally optimal k-anonymity method for the de-identification of health data." *JAMIA*, vol. 16, no. 5, pp. 670–682, 2009.

[18] K. LeFevre, D. J. DeWitt, and R. Ramakrishnan, "Mondrian multidimensional k-anonymity," in *ICDE*, 2006.

[19] J.-W. Byun, A. Kamra, E. Bertino, and N. Li, "Efficient k-anonymization using clustering techniques," in *Proceedings of the 12th International Conference on Database Systems for Advanced Applications*, ser. DASFAA'07. Berlin, Heidelberg: Springer-Verlag, 2007, pp. 188–200.

[20] G. Loukides and J.-H. Shao, "An efficient clustering algorithm for k-anonymisation," *J. Comput. Sci. Technol.*, vol. 23, no. 2, pp. 188–202, Mar. 2008.

[21] G. Aggarwal, R. Panigrahy, T. Feder, D. Thomas, K. Kenthapadi, S. Khuller, and A. Zhu, "Achieving anonymity via clustering," *ACM Trans. Algorithms*, vol. 6, no. 3, pp. 49:1–49:19, Jul. 2010.

[22] W. Jiang and C. Clifton, "A secure distributed framework for achieving k-anonymity," *The VLDB Journal*, vol. 15, no. 4, pp. 316–333, Nov. 2006.

[23] W. Jiang and M. Atzori, "Secure distributed k-anonymous pattern mining," in *Sixth International Conference on Data Mining (ICDM'06)*, Dec 2006, pp. 319–329.

[24] C. Brunetta, C. Dimitrakakis, B. Liang, and A. Mitrokotsa, "A differentially private encryption scheme," in *Information Security*, P. Q. Nguyen and J. Zhou, Eds. Cham: Springer International Publishing, 2017, pp. 309–326.

[25] D. Liu, E. Bertino, and X. Yi, "Privacy of outsourced k-means clustering," in *Proceedings of the 9th ACM Symposium on Information, Computer and Communications Security*, ser. ASIA CCS '14. New York, NY, USA: ACM, 2014, pp. 123–134.

[26] S. Antonatos, S. Braghin, N. Holohan, Y. Gkoufas, and P. Mac Aonghusa, "Prima: an end-to-end framework for privacy at scale," in *ICDE*, 2018.

[27] A. Gkoulalas-Divanis, S. Braghin, and S. Antonatos, "FPVI: A Scalable Method for Discovering Privacy Vulnerabilities in Microdata," in *Proceedings of the Second IEEE ISC2*, 2016.

[28] F. Liu, "Statistical properties of sanitized results from differentially private laplace mechanisms with noninformative bounding," *ArXiv e-prints*, vol. 1607.08554 [stat.ME], Jul. 2016.

[29] G. Chen and S. Keller-McNulty, "Estimation of identification disclosure risk in microdata," *Journal of Official Statistics*, vol. 14, no. 1, p. 79, 1998.

[30] N. Hoshino, "Applying pitman's sampling formula to microdata disclosure risk assessment," *Journal of Official Statistics*, vol. 17, no. 4, p. 499, 2001.

[31] L. V. Zayatz, "Estimation of the percent of unique population elements on a microdata file using the sample," in *Statistical Research Division Report Number: Census/SRD/RR-91/08*. Citeseer, 1991.

[32] R. J. Bayardo and R. Agrawal, "Data privacy through optimal k-anonymization," in *ICDE*, 2005.

[33] L. Sweeney, "Achieving k-anonymity privacy protection using generalization and suppression," *Int. J. Uncertain. Fuzziness Knowl.-Based Syst.*, vol. 10, no. 5, Oct. 2002.

[34] V. S. Iyengar, "Transforming data to satisfy privacy constraints," in *KDD*. ACM, 2002.

[35] A. Gionis and T. Tassa, "k-anonymization with minimal loss of information," *IEEE TKDE*, vol. 21, no. 2, 2009.

[36] G. Ghinita, P. Karras, P. Kalnis, and N. Mamoulis, "Fast data anonymization with low information loss," in *VLDB*, 2007.

[37] M. Kesarwani, A. Kaul, S. Braghin, N. Holohan, and S. Antonatos, "Secure k-anonymization over encrypted databases," in *arXiv*.

[38] "Helib," https://github.com/shaih/HElib, September 2018.