Towards Meaningful Distance-Preserving Encryption

Christine Tex
Karlsruhe Institute of Technology
christine.tex@kit.edu

Martin Schäler
Karlsruhe Institute of Technology
martin.schaeler@kit.edu

Klemens Böhm
Karlsruhe Institute of Technology
klemens.boehm@kit.edu

ABSTRACT

Mining complex data is an essential and at the same time challenging task. Therefore, organizations pass on their *encrypted* data to service providers carrying out such analyses. Thus, encryption must preserve the mining results. Many mining algorithms are distance-based. Thus, we investigate how to preserve the results for such algorithms upon encryption. To this end, we propose the notion of distance-preserving encryption (DPE). This notion has just the right strictness – we show that we cannot relax it, using formal arguments as well as experiments. Designing a DPE scheme is challenging, as it depends both on the data set and the specific distance measure in use. We propose a procedure to engineer DPE-schemes, dubbed DisPE. In a case study, we instantiate DisPE for SQL query logs, a type of data containing valuable information about user interests. In this study, we design DPE schemes for all SQL query distance measures from the scientific literature. We formally show that one can use a combination of existing secure property-preserving encryption schemes to this end. Finally, we discuss on the generalizability of our findings using two other data sets as examples.

CCS CONCEPTS

• **Information systems** → **Data encryption**; **Data mining**; *Relational database query languages*; Relational database model; XML query languages;

ACM Reference Format:
Christine Tex, Martin Schäler, and Klemens Böhm. 2018. Towards Meaningful Distance-Preserving Encryption. In *SSDBM '18: 30th International Conference on Scientific and Statistical Database Management, July 9–11, 2018, Bozen-Bolzano, Italy.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3221269.3223029

1 INTRODUCTION

Today it is common practice for organizations to pass on their data to service providers, with the data being encrypted. An important service which organizations would like to benefit from is data mining. However, data mining on encrypted data generally does not yield meaningful results. To overcome this, encryption must preserve service-specific properties of the data [25]. Examples are equality-preserving, i.e., deterministic, or order-preserving encryption [2]. An important property of a data set are the pairwise distances between the data items. A wide spectrum of data-analysis algorithms, so called distance-based algorithms, relies on these distances only. For instance, think of distance-based clustering [14, 16, 26] or outlier detection [20], which are frequently used. If encryption preserves the pairwise distances between the data items, the mining results on the encrypted and the plain-text data are the same. However, to our knowledge, distance-preserving encryption has not been investigated systematically in the literature before. For instance, CryptDB [27] addresses only one specific case which we are about to discuss.

This paper examines how to design distance-preserving encryption (DPE) schemes for data of arbitrary types. For data sets consisting of items with a complex inner structure, such as graphs or query logs, this is challenging, for the following reasons: (1) unclear subject of encryption and (2) distance-measure variety. We now illustrate these points using SQL query logs as a use case; this will also be our running example in the paper. Query logs contain valuable information about user interests [6, 23] and therefore are an important resource for data mining. Since SQL query log mining is far from trivial, it is reasonable to outsource it to a service provider or an external researchers. We now illustrate the challenges. First, SQL queries have a complex inner structure. For data items with a complex structure, it tends to be unclear what "encryption of the data items" actually means, cf. Example 1.1.

Example 1.1 (Unclear Subject of Encryption)**.** *Consider a set of SQL queries, i.e., an SQL query log. Which parts of the query should be encrypted? For example, one may encrypt the query string as a whole or only the tokens within the query string.*

Thus, when designing a DPE scheme, one must specify a security model tailored to the type of data considered and an encryption scheme in line with this model. Second, for any type of data, there exist different distance measures. Example 1.2 illustrates that distance-preserving encryption depends on the measure in use.

Example 1.2 (Distance-Measure Variety)**.** *For distance-based data mining over an SQL query log, one needs a distance measure. Different such measures exist. For instance, we can use a string distance measure like the Levenshtein distance or a measure that depends on the overlap of the tuples in the results of the queries. These measures are conceptually different: For example, two queries may lead to the same result even if the query string is different. Therefore, different distance-preserving encryption schemes are needed: For string distance, it might be reasonable to encrypt every character in the query string. In contrast, for distance measures depending on the result tuples of the queries, a fundamental requirement is the executability of the encrypted query. This is because the result tuples and their overlap cannot be computed otherwise.*

Thus, when designing a DPE scheme, one must differentiate between the different distance measures.

SSDBM '18, July 9–11, 2018, Bozen-Bolzano, Italy

Contributions

In this paper, we examine how to design distance-preserving encryption schemes for complex-structured data. To this end, we split our paper in two parts: (1) An abstract part, in which we introduce the general concepts, and (2) a case-study part, in which we study how to instantiate the concepts using the example of SQL logs. First, in the abstract part, we define distance-preserving encryption for arbitrary data sets and distance measures. Using formal arguments and experiments, we show that it would not make sense to work with a notion that is less strict. In addition, we specify a general procedure for designing distance-preserving encryption named DisPE. This procedure describes the steps necessary to arrive at a distance-preserving encryption scheme for a certain data set. We review well-known property-preserving encryption schemes from literature to illustrate (a) that one can apply them to instantiate our DisPE procedure (b) to assess the security of the resulting distance-preserving encryption scheme. In our study, that is the second part of this paper, we examine which property-preserving encryption schemes one can apply when implementing distance-preserving encryption schemes for SQL logs. As result, we find DPE schemes for four well-known SQL query distance measures. The schemes have a higher security level than the ones CryptDB [27] would generate, i.e., shield against more attacks. Finally, by means of two further use cases, namely XQuery and relational data, we say to which extent our results can be leveraged and what remains to be done.

This paper is an extended version of an earlier publication [32], with the following extensions: We prove that one cannot meaningfully relax our notion of distance-preserving encryption, using formal arguments (Section 3.2) and experiments (Section 7). We also describe generalizations of the results of our case study (Section 6).

2 BACKGROUND AND RELATED WORK

In this section, we introduce background knowledge on encryption schemes together with notation and relate it to distance-preserving encryption. In addition, we review related work on property-preserving encryption schemes, since we leverage them to implement distance-preserving encryption. We also leverage other results from literature [27, 29] for substeps of our procedure. We explain them in the corresponding section in the body of this paper as far as needed.

2.1 Encryption Schemes and Their Attributes

An encryption scheme consists of three algorithms, cf. Definition 2.1. We refer to unencrypted data as plain-text data and to encrypted data as cipher-text data. Now, we discuss specifications and security of encryption schemes and relate them to distance-preserving encryption.

Definition 2.1 (Encryption Scheme)**.** *An encryption scheme is a Tuple (*Gen, Enc, Dec*) where*

- Gen *is a key-generation algorithm that outputs a tuple consisting of two Keys* (K_E, K_D)*,*
- $\mathsf{Enc}_{K_E}(P)$ *is an encryption algorithm that encrypts a Plain-Text Value* P *using Key* K_E*,*
- $\mathsf{Dec}_{K_D}(C)$ *is a decryption algorithm that decrypts Cipher-Text Value* C *using Key* K_D*.*

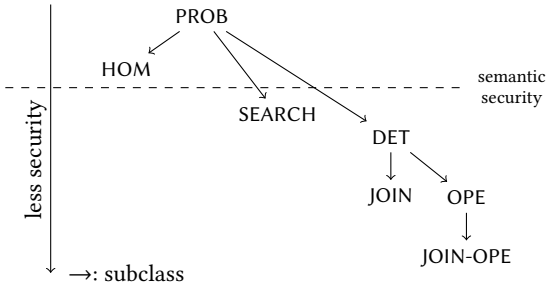


Figure 1: Taxonomy of Property-Preserving Encryption.

2.1.1 Specifications of Encryption Schemes. Encryption schemes form groups with the same specification. An example for a specification is being asymmetric or symmetric. It is application-dependent which encryption schemes with which specifications one should use. In the following, we review the relevant specifications from literature and relate them to distance-preserving encryption.

Asymmetric vs. Symmetric. In symmetric key encryption schemes, there is one key for encryption and decryption (i.e., $K_E = K_D$), while in asymmetric (or *public key*) schemes, $K_E \neq K_D$. Symmetric key encryption schemes are more efficient than asymmetric ones [1], but have the disadvantage that a key exchange is needed. In our scenario, symmetric schemes are suitable, since encryption and decryption is done by the same party (the organization), i.e., no key exchange is needed. However, using a more powerful asymmetric scheme is possible as well. Our considerations in the remainder hold for both, and we do not differentiate between them.

Stateful vs. Stateless. In stateful encryption schemes, the encryption algorithm has, next to the plain-text value and the encryption key, an additional input and output parameter – the state of the scheme [7]. For instance, when considering stateful encryption in combination with property-preserving encryption, the state often is the set of all values that have been encrypted so far [2]. In most scenarios, for instance in database encryption [27], the values to be encrypted are not known in advance. Stateful schemes are unpractical there. In our scenario however, the data set to be encrypted is fixed, as the organization shares an already existing data set with the service provider. Therefore, we can rely on stateful schemes.

2.1.2 Security of Encryption Schemes. Different encryption schemes provide different levels of security, i.e., shield against different attacks. In cryptography, we differentiate between *active* and *passive* attacks. In an active attack, the attacker has access to a decryption oracle, i.e., can decrypt certain cipher-texts to some extent. A passive attacker does not have this ability. In our context, we see the service provider as a passive attacker. Namely, it is not intended that he asks the organization for the decryption of cipher-text values. There typically are three types of passive attacks [29]:

- Cipher-Text-Only Attack: The attacker has access to several cipher-texts someone else has selected. The attacker is successful if he can decrypt (any) randomly selected cipher-text.

- Known-Plain-Text Attack: The attacker has access to several cipher-texts someone else has selected and the corresponding plain-texts. He is successful if, given one cipher- and two possible plain-texts, he can determine the correct plain-text.
- Chosen-Plain-Text Attack: The only difference between known-plain-text attacks and chosen-plain-text attacks is that the attacker can select which pairs of cipher text and corresponding plain-text he has access to.

We come back to these attacks in Section 5.1.1, when turning them to attacks on SQL query logs, our running example. The maximum security level for passive attacks is semantic security [15], which is security against the chosen-plain-text attack described above.

2.2 Property-Preserving Encryption

Preserving properties of a data set (e.g., the order of the data items) upon encryption is a common requirement. Property-preserving encryption schemes are classified by the plain-text properties they preserve. For such a scheme, "perfect security" against passive attacks, i.e., semantic security, cannot be guaranteed for most properties. This is because the scheme intentionally leaks (at least) the property of the plain-text preserved. Since different classes preserve and leak different properties, different classes provide different security. In the following, we briefly explain different classes of property-preserving encryption schemes, to apply them to distance-preserving encryption. Figure 1 depicts a taxonomy showing the relationships and the security levels of the various classes, inspired by [21, 27]. The rows stand for the security levels, higher is better. For classes in the same row, because of incomparable, class-specific security notions, a security ranking is not possible.

Probabilistic Encryption (PROB). Encryption schemes are *probabilistic* if, in general, two equal values are mapped to different cipher-texts. PROB schemes without any additional assumption do not preserve any property of the data. But they feature semantic security and even security against active attacks.

Homomorphic Encryption (HOM) [17]. Homomorphic schemes are probabilistic schemes allowing for arithmetic aggregate functions, e.g., sums, over encrypted data. As this property opens a way for active attacks, the security level is lower than for PROB schemes. However, beside probabilistic encryption, HOM is the only class that provides semantic security. The reason is that the leaked property is the homomorphism (e.g., regarding addition), but no information on the semantics of the actual values.

Searchable Encryption (SEARCH) [10, 31]. Searchable encryption allows for keyword search on encrypted data (e.g., LIKE '%unicorn%'). Thus, the property leaked is the occurrence of keywords in the plain-text.

Deterministic Encryption (DET). A scheme is *deterministic* if two equal values are mapped to the same cipher-text. Therefore, DET schemes allow for equality checks over encrypted data.

Order-Preserving Encryption (OPE) [2]. Order-preserving encryption schemes are deterministic and preserve the order of the data. Thus, one is able to perform range queries.

Join-Preserving Encryption (JOIN/JOIN-OPE) [27]. JOIN is a special usage mode of a DET scheme (JOIN-OPE of OPE, respectively), to allow for joins over encrypted databases. Particularly, the same encryption scheme and key encrypt the primary and foreign key.

Table 1: Common Encryption Schemes [21, 27].

Class	Stateless Schemes	Stateful Schemes
PROB	randomized AES [13]	-
HOM	Paillier [24]	-
SEARCH	Song et. al [31]	-
DET	block cipher, e.g., AES [13]	-
OPE	Boldyreva et. al [8]	OPES [2]

Encryption Schemes for Encryption Classes. For every encryption class, different encryption schemes exist. See Table 1 for the commonly used schemes. For every class, the associated schemes preserve the properties the class preserves. As mentioned in Section 2.1, we can use stateful schemes in this paper. To our knowledge, except for the OPE-class, no meaningful stateful scheme exists.

Implications. This section states which property-preserving encryption classes are known. However, so far, it is unknown how to realize distance-preserving encryption for complex data using these classes. On the one hand, this is challenging as one usually has to consider different distance measures. On the other hand, using known classes and schemes has the advantage that their security is well-studied. Thus, in the remainder of this paper, we focus on how to realize distance-preserving encryption using these classes.

3 DISTANCE-PRESERVING ENCRYPTION

In this section, we first define distance-preserving encryption for arbitrary data sets and distance measures. Then we discuss why our definition is reasonable.

3.1 Definition

With distance-preserving encryption (DPE), the pairwise distances for the plain-text and the cipher-text data items must be the same.

Definition 3.1 (Distance-Preserving Encryption (DPE))**.** *Let $\mathcal{D}$ be a data set,* d *be a distance measure and* Enc *an encryption algorithm for data items in $\mathcal{D}$. Then,* Enc *is* d*-distance preserving if*

$$\forall x, y \in \mathcal{D} : d(\mathsf{Enc}(x), \mathsf{Enc}(y)) = d(x, y).$$

Distance-preserving encryption enables distance-based data mining on encrypted data sets. This means that the mining results on cipher-text and plain-text data are the same. For instance, the same data items are assigned to clusters. To justify our definition, we explain its usefulness with the help of the k-medoids clustering algorithm [26], a very common distance-based clustering algorithm. Algorithm 1 is the k-medoids clustering algorithm. It works as the k-means algorithm [18], with the difference that the cluster centers always are objects occurring in the data set.

Lemma 3.1. *If* Enc *is* d*-distance-preserving, given the same initialization in Line 2, it holds that*

$$\mathsf{Enc}(k\text{-}medoids(d, k, \mathcal{D})) = k\text{-}medoids(d, k, \mathsf{Enc}(\mathcal{D})).$$

Proof. In the k-medoids algorithm in Algorithm 1, the only information extracted from the data set are the pairwise distances of the data items (Line 1). □

As the argumentation of Lemma 3.1 holds for all distance-based data mining algorithms, Lemma 3.1 holds for any of them.

Algorithm 1 k-medoids(d, k, $\mathcal{D}$)

```
D ← ∀x, y ∈ 𝒟: pairwise distance d(x, y)
select k data items as medoids
for x ∈ 𝒟 do
    associate x with the closest medoid
end for
cost = Σ_{Medoid M} Σ_{all Items x associated to M} d(M, x)
while cost decreases do
    for Medoid M and Associated Item x do
        swap the role of M and x
        recompute cost
        if cost increased then
            redo swap
        end if
    end for
end while
```

3.2 Inadequacy of Relaxed Notions

The equality condition regarding all pairwise distances in Definition 3.1 is a strict requirement. One may ask whether an ϵ-approximation of the distance value such as

$$|d(\mathsf{Enc}(x), \mathsf{Enc}(y)) - d(x, y)| \leq \epsilon \cdot d(x, y),$$

or preserving the relative ordering such as

$$d(x, y) < d(x, z) \iff d(\mathsf{Enc}(x), \mathsf{Enc}(y)) < d(\mathsf{Enc}(x), \mathsf{Enc}(z)),$$

is sufficient. This is not the case. We illustrate this with two examples in this section, as well as with experiments in Section 7. First, to understand that using an approximate value is not sufficient, consider Example 3.1.

Example 3.1 (Approximative Distance-Values)**.** *Consider a Data Set $\mathcal{D} = \{w, x, y, z\}$ where the pairwise distance between the data items is given by the following distance matrix:*

$$D = \begin{array}{c} \\ w \\ x \\ y \\ z \end{array} \begin{array}{c} \begin{array}{cccc} w & x & y & z \end{array} \\ \begin{pmatrix} 0 & 0.1 & 0.11 & 0.11 \\ 0.1 & 0 & 0.11 & 0.11 \\ 0.11 & 0.11 & 0 & 0.1 \\ 0.11 & 0.11 & 0.1 & 0 \end{pmatrix} \end{array}$$

A k-medoids clustering with $k = 2$ will result in the following clusters: $\{w, x\}$ and $\{y, z\}$. If we have an approximation of the distances with $\epsilon = 0.1$, this may result in a distance matrix where the distance between all data items is given by:

$$D = \begin{array}{c} \\ w \\ x \\ y \\ z \end{array} \begin{array}{c} \begin{array}{cccc} w & x & y & z \end{array} \\ \begin{pmatrix} 0 & 0.11 & 0.1 & 0.1 \\ 0.11 & 0 & 0.1 & 0.1 \\ 0.1 & 0.1 & 0 & 0.11 \\ 0.1 & 0.1 & 0.11 & 0 \end{pmatrix} \end{array}$$

The Items w and x will not be in the same cluster, as well as y and z, because their pairwise distances are higher than the other pairwise distances. Thus, the result of the clustering on encrypted data is different.

To understand why preserving the order of the pairwise distance is not sufficient, see Example 3.2.

Table 2: Example of Preserving the Order of the Pairwise Distances upon Encryption; Distances in Ascending Order.

Item-Pair	Distance on Plain-Text Data	Distance on Cipher-Text Data
(w, x)	0.1	0.1
(y, z)	0.11	0.95
(y, w)	0.5	0.96
(y, x)	0.51	0.97
(w, z)	0.52	0.98
(x, z)	0.53	0.99

Example 3.2 (Preserving Relative Order)**.** *Consider the example distances in Table 2. Again, we perform a k-medoids clustering with $k = 2$ on the plain-text data. This results in the following clusters: $\{w, x\}$ and $\{y, z\}$. Now we encrypt the data set. As we see in Table 2, the order of the distances is the same for the plain-text and the encrypted items. Now consider a variant of the k-medoids algorithm having a threshold for increasing the outlier robustness: If a Data Item x has, say, a distance ≥ 0.9 to all medoids, x is deemed an outlier and not assigned to a cluster. As we see, this is the case for Item y in the cipher-text data, but not in the plain-text data, as $d(y, z)$ differs. Thus, the result of the k-medoids algorithm differs, because it outputs an outlier when performed on cipher-text data, but not on the plain-text data.*

To conclude, the pathological examples in this section indicate that ensuring the equality of the distances upon encryption is necessary to guarantee that mining results on plain-text and cipher-text data are the same. In Section 7, we also show this for real data with experiments.

4 DISTANCE-PRESERVING ENCRYPTION WITH DISPE

In this paper, we aim at finding secure distance-preserving encryption schemes for complex data. To this end, we introduce the general procedure for designing distance-preserving encryption DisPE for complex data sets. We will instantiate it for SQL queries in Section 5. Now, in turn, we remain on an abstract level. The four steps of DisPE are as follows:

(1) Definition of the Security Model: In the first step, one has to specify the security goals one wants to achieve with encryption, e.g., "the SQL log should not reveal information on the content of the database". To this end, one has to (1) specify the threat model, i.e., the attacks to shield against, and (2) define a high-level encryption scheme for the type of data considered, e.g., "encrypt all constants in the query".
(2) Finding a Suitable Equivalence Notion: Our aim is to implement the high-level encryption scheme defined in the first step so that it is distance-preserving. The distance between data items is defined for pairs of items, but encryption is done item-wise. Therefore, we introduce an intermediate notion defined for single date items: For a given distance measure, an *equivalence notion* captures the characteristics of a single data item that should be preserved upon encryption, cf. Definition 4.1. For instance, when encrypting graph data, i.e, $\mathcal{S}$ is the set of all

valid graphs, a Characteristic c to be preserved could be the number of incoming edges of vertices in the graph.

Definition 4.1 (c-Equivalence). *Let $\mathcal{D} \subseteq \mathcal{S}$ be a data set and $c : \mathcal{S} \to \mathcal{S}$ a function (*characteristic*). In addition, let* Enc *be an encryption algorithm for data items in $\mathcal{S}$. Then* Enc *ensures c-equivalence if*

$$\forall x \in \mathcal{D} : \mathsf{Enc}(c(x)) = c(\mathsf{Enc}(x)).$$

(3) Ensuring the Equivalence Notions: In this step, one has to implement the high-level encryption scheme defined in Step 1 so that it ensures the equivalence notion defined in the second step. To this end, we deploy property-preserving encryption classes introduced in Section 2.2. Instead of specifying concrete schemes, we specify the corresponding encryption class, as the property to preserve is defined at class level and holds for all schemes belonging to the class. A user then selects a scheme in the class (cf. Table 1). In general, there are several encryption classes which can ensure an equivalence notion. We always select the appropriate encryption class according to Definition 4.2. As encryption-class taxonomy, we use the one from Figure 1.

Definition 4.2 (Appropriate Encryption Class). *For a given equivalence notion and encryption algorithm in {*$\mathsf{Enc}^{A.Const}$, Enc^{Attr}, Enc^{Rel}*}, an encryption class is appropriate according to an encryption-class taxonomy if*
(a) it ensures the equivalence notion and
(b) provides the highest security possible.

(4) Security Assessment: Finally, we have to assess the security of the encryption scheme implemented. If one uses only schemes whose security is known from the literature, the security assessment is given; this is the desired case. Otherwise, a security analysis as in [9] is needed.

5 CASE STUDY: DISTANCE-PRESERVING ENCRYPTION OF SQL QUERY LOGS

In this section, we instantiate our DisPE procedure with SQL query logs as our use case. After this, we explain which parts of our instantiation of DisPE one can reuse when dealing with types of data other than SQL logs.

5.1 Security Model

In this section, we introduce the threat model for attacks on SQL query logs considered in this paper and a high-level encryption scheme for SQL queries.

5.1.1 Threat Model. In this section, we (1) explain a general threat model and (2) instantiate it for SQL query logs. For the instantiation, we leverage the solution in [29]. In the scenario addressed here, an organization and a service provider share an encrypted SQL log. The service provider has the opportunity to perform certain attacks on the encrypted log. Our goal is to ensure security in the form of confidentiality for the underlying database. This means that we want to limit the information one can infer from the log about (1) names of relations occurring in the database, (2) names of attributes of the relations and (3) the content of the database. As stated in Section 2.1.2, one has to shield against passive attacks only. Hence, it is necessary to transform the abstract passive attacks explained in Section 2.1.2 to query logs. Literature already features this [29], and the transformations are as follows:

- Cipher-Text Only Attack → Query-Only Attack: In a Query-Only Attack, the attacker only has access to the encrypted query log and tries to infer the plain-text values of constants, relation names as well as attribute names[1] of a given encrypted query. For instance, the attacker has access to an encrypted query log and tries to learn information from this that helps him to decrypt a constant in a specific query.
- Known-Plain-Text Attack → Known-Query Attack: In a Known-Query Attack, the attacker has access to a number of plain-text/cipher-text query pairs, and has to distinguish between two new cipher-text queries.
- Chosen-Plain-Text Attack → Chosen-Query Attack: In a Chosen-Query Attack, the attacker has black-box access to an encryption oracle, i.e., the ability to encrypt queries, and has to distinguish between two cipher-text queries that he must not encrypt by using the oracle.

5.1.2 Encryption of SQL Queries. Intuitively, SQL queries can be encrypted in various ways, for instance by encrypting the query string as a whole. However, if we want to hide the names of relations, attributes and values of the attributes in the database only, it is sufficient to encrypt only these parts of the queries. This encryption technique for SQL queries is also known as "encryption-aware query rewriting" and has the feature that it even hides the fact that the log is encrypted [28]. While the idea of encryption-aware query rewriting has been around, it has not been studied so far how to instantiate it so that the encryption scheme for SQL queries is distance preserving. – In the following, *Attr* is the set of database attributes occurring in the SQL log.

Definition 5.1 (Encryption Scheme for SQL Queries). *For*

$$i \in \{Rel, Attr\} \cup \{A.Const \mid A \in Attr\},$$

*let the Tuple $S^i = (\mathsf{Gen}^i, \mathsf{Enc}^i, \mathsf{Dec}^i)$ be an encryption scheme. An encryption scheme for SQL queries is a Tuple (*Gen, Enc, Dec*) where*

- $\mathsf{Gen} = (\mathsf{Gen}^{Rel}, \mathsf{Gen}^{Attr}, \{\mathsf{Gen}^{A.Const} \mid A \in Attr\})$
- $\mathsf{Dec} = (\mathsf{Dec}^{Rel}, \mathsf{Dec}^{Attr}, \{\mathsf{Dec}^{A.Const} \mid A \in Attr\})$ *and*
- $\mathsf{Enc} = (\mathsf{Enc}^{Rel}, \mathsf{Enc}^{Attr}, \{\mathsf{Enc}^{A.Const} \mid A \in Attr\})$.

The Scheme S^i in Definition 5.1 is used to encrypt $i \in \{\text{Rel}, \text{Attr}\} \cup \{A.\text{Const} \mid A \in \text{Attr}\}$, i.e., the names of relations, attributes and the constants belonging to different attributes in an SQL string.

Example 5.1. *For $Q =$* 'SELECT $A1$ FROM R WHERE $A2 > 5$' *the encrypted query is*

$$\begin{aligned}\mathsf{Enc}(Q) = \text{'}&\mathsf{SELECT}\ \mathsf{Enc}^{Attr}(A1)\\ &\mathsf{FROM}\ \mathsf{Enc}^{Rel}(R)\\ &\mathsf{WHERE}\ \mathsf{Enc}^{Attr}(A2) > \mathsf{Enc}^{A2.Const}(5)\text{'}.\end{aligned}$$

Observe that an encryption scheme for SQL queries as in Definition 5.1 is not limited to SQL strings, but works on any string. We will use this generalization to encrypt query results.

[1] Strictly speaking, the thread model in [29] is only defined for constants, but the model can be generalized to relation and attribute names.

5.2 Suitable Equivalence Notions

In this section, we define suitable equivalence notions (cf. Definition 4.1) for different SQL query distance measures. We first define all notions, before we investigate how to implement encryption schemes for SQL queries that fulfill the notions in Section 5.3. By doing so, we follow the DisPE procedure. Table 3 gives an overview of query distance measures from literature and the core results of this and the following section. The measures follow a natural ordering: Within query-string distance, an SQL query is considered simply as a string. Query-structure distance then takes the structure of the query into account. Finally, query-result and query-access-area distance are based on the semantics, i.e., result and execution, of the query. To compute the latter two measures, it is not sufficient to only share the query log. For instance, to calculate query-result distance, the content of the database is needed as well (cf. Table 3). We discuss this issue when introducing the measures.

5.2.1 Token-Based Query-String Distance. In this subsection, we introduce token-based query-string distance and its underlying equivalence notion called token equivalence.

Definition. To define query-string distance, one interprets an SQL query as a string and uses a String-Distance Measure *string_dist* to calculate the distance, cf. Definition 5.2.

Definition 5.2 (Query-String Distance)**.** *Let $Q1$ and $Q2$ be SQL queries. The query-string distance of $Q1$ and $Q2$ is*

$$d_{String}(Q1, Q2) = string\_dist(Q1, Q2).$$

There are different types of string-distance measures [12], but not all of them are adequate for SQL queries:

Edit Distance. The edit distance $d_{\text{edit}}(s_1, s_2)$ is the minimum number of edit operations (insert, delete, substitute) needed to transform s_1 in s_2. However, intuitively, the edit distance is not appropriate for SQL strings. For example, the edit distance of two SQL queries which only differ in the ordering of the relations in the FROM-clause may be very high. Therefore, we exclude the edit distance from our further considerations.

Token-Based. Token-based distance measures divide a string into tokens (words). Thus, every Query Q is represented as a set of tokens, short *tokens*(Q), cf. Example 5.2. We summarized predicates of the form $A \ominus B$ to one token to avoid the intuitive issue that predicates like $A < B$ and $A > B$ results in the same token set. Then set distance measures such as the Jaccard coefficient or cosine distance (via text-to-vector) are used.

Example 5.2. *The token set of the Query*

$$Q = \text{'SELECT } A \text{ FROM } R \text{ WHERE } B > 5\text{'}$$

is given by

$$tokens(Q) = \{\text{SELECT}, A, \text{FROM}, R, \text{WHERE}, B > 5\}.$$

In the remainder of this paper, we focus on token-based query-string distance as stated in Definition 5.3.

Definition 5.3 (Token-Based Query-String Distance)**.** *Let $Q1$, $Q2$ be SQL queries. Then the token-based query-string distance between $Q1$ and $Q2$ is*

$$d_{Token}(Q1, Q2) = 1 - \frac{|\text{tokens}(Q1) \cap \text{tokens}(Q2)|}{|\text{tokens}(Q1) \cup \text{tokens}(Q2)|}.$$

Obviously, to calculate this distance, only the queries itself (i.e., the log file) must be shared.

Token Equivalence. For token-based query-string distance, the characteristic to be preserved is the token set of the queries, i.e., $c = tokens$. Note that in Definition 5.4, the encryption algorithm Enc for SQL queries is used to encrypt a token set. Token-wise encryption using the corresponding scheme according to Definition 5.1 implements this. For instance, in Example 5.4, the Token A is encrypted using Enc^{Rel}.

Definition 5.4 (Token Equivalence)**.** *Let Q be a query. Then* Enc *ensures token equivalence for Q if the following holds:*

$$\text{Enc}(\text{tokens}(Q)) = \text{tokens}(\text{Enc}(Q)).$$

Lemma 5.1. *Let* Enc *be an encryption algorithm for SQL queries. If* Enc *ensures token equivalence, then it is token-based query-string distance-preserving.*

Proof. Let $Q1$ and $Q2$ be SQL queries. If token equivalence holds, we get the same (encrypted) token set if we compute it on the plain-text or on cipher-text queries. Obviously, the cardinalities $|tokens(Q1) \cap tokens(Q2)|$ and $|tokens(Q1) \cup tokens(Q2)|$ remain the same. Hence, d_{Token} remains the same. □

5.2.2 Query-Structure Distance. The next distance measure for SQL queries is query-structure distance.

Definition. In [19], the authors extract semantically important features from a query, dubbed *features*(Q). A feature of a query is a tuple representing a part of its structure. Table 4 is a complete list of features an SQL query can have and Example 5.3 shows an example.

Example 5.3. *Consider the Query Q from Example 5.2. The feature set of Q is given by*

$$\text{features}(Q) = \{(\text{SELECT}, A), (\text{FROM}, R), (\text{WHERE}, B >)\}.$$

Now, query-structure distance is defined as stated in Definition 5.5.

Definition 5.5 (Query-Structure Distance)**.** *Let $Q1$, $Q2$ be SQL queries. Then the query-structure distance between $Q1$ and $Q2$ is*

$$d_{Struc}(Q1, Q2) = 1 - \frac{|\text{features}(Q1) \cap \text{features}(Q2)|}{|\text{features}(Q1) \cup \text{features}(Q2)|}.$$

To calculate query-structure distance, only the queries themselves (i.e., the log file) must be shared between the organization and the service provider.

Structural Equivalence. Structural equivalence ensures that there is no difference if we first compute the features of a query and then encrypt the feature set or vice versa. In Definition 5.6, we encrypt features of queries, which is realized as explained for token equivalence in Section 5.2.1.

Definition 5.6 (Structural Equivalence)**.** *Let Q be a query.* Enc *ensures structural equivalence for Q if the following holds:*

$$\text{Enc}(\text{features}(Q)) = \text{features}(\text{Enc}(Q)).$$

Lemma 5.2. *Let* Enc *be an encryption algorithm for SQL queries. If* Enc *ensures structural equivalence, then* Enc *is query-structure distance-preserving.*

Proof. Analogously to the proof of Lemma 5.1, as the same arguments regarding set cardinalities hold. □

Table 3: Overview of Query-Distance Measures.

Distance Measure	Shared Information			Equivalence Notion	c	$\mathsf{Enc}^{\mathrm{Rel}}$	$\mathsf{Enc}^{\mathrm{Attr}}$	$\mathsf{Enc}^{A.\mathrm{Const}}$
	Log	DB-Content	Domains					
Token-Based Query-String Distance	✓	✗	✗	Token Equivalence	*tokens*	DET	DET	DET
Query-Structure Distance	✓	✗	✗	Structural Equivalence	*features*	DET	DET	PROB
Query-Result Distance	✓	✓	✗	Result Equivalence	*result_tuples*	DET	DET	via CryptDB [27]
Query-Access-Area Distance	✓	✗	✓	Access-Area Equivalence	$access_A$	DET	DET	via CryptDB, excp. HOM

Table 4: Features of an SQL Query [19].

Feature	Explanation
(FROM, R)	for every Relation, View and Relation-Valued Aggregate Function R in the FROM-clause
(C, A)	for every Attribute A in the Clause C $\in$ {SELECT, WHERE, GROUP BY}
(SELECT, aggr($A_1, .., A_n$)	for every Aggregate Function *aggr* and list of Attributes $A_1, .., A_n$ in the SELECT-clause
(WHERE, $A_1 \Theta A_2$)	for every pair of Attributes A_1, A_2 and Operator Θ that appears in the WHERE-clause
(WHERE, $A\Theta$)	for every Attributes A and Operator Θ in the WHERE-clause (representing predicates of the form $A\ \Theta$ const)
(subquery, o)	for every Subquery *subquery* that appears in WHERE-clause in combination with a Set Operator $o \in$ {All, Any, Some, In, Exists} in a predicate

5.2.3 Query-Result Distance. The third SQL query distance measure proposed in the literature is query-result distance.

Definition. An SQL Query Q can be represented as the set of tuples in its result, *result_tuples*(Q) [3]. A result tuple is defined by the unique combination of attribute values. Since *result_tuples*(Q) is a set, there are no duplicates. Every query is interpreted as a query with the DISTINCT-operator. Query-result distance is defined as stated in Definition 5.7.

Definition 5.7 (Query-Result Distance)**.** *Let $Q1$, and $Q2$ be two SQL queries. Then the query-result distance between $Q1$ and $Q2$ is*

$$d_{Res}(Q1, Q2) = 1 - \frac{|\text{result\_tuples}(Q1) \cap \text{result\_tuples}(Q2)|}{|\text{result\_tuples}(Q1) \cup \text{result\_tuples}(Q2)|}.$$

It is important that the tuples in *result_tuples*(Q) depend on the state of the database. To calculate them, we must requery the database. Thus, besides the query log itself, it is necessary to share parts of the *encrypted* database as well. In particular, one needs to share the content of all attributes that are accessed by at least one query in the log. I.e., indices and constraints are not needed. In Table 3, we refer to the part of the database that needs to be shared as DB-Content.

Result Equivalence. For query-result distance, the Characteristic c to be preserved is c = *result_tuples*.

Definition 5.8 (Result Equivalence)**.** *Let Q be a query. Then* Enc *ensures result equivalence for Q if*

$$\mathsf{Enc}(\text{result\_tuples}(Q)) = \text{result\_tuples}(\mathsf{Enc}(Q)).$$

Lemma 5.3. *Let* Enc *be an encryption algorithm for SQL queries. If* Enc *ensures result equivalence, it is result distance-preserving.*

Proof. Intuitively, in case result equivalence holds, the number of tuples in the Sets *result_tuples*($Q1$) $\cap$ *result_tuples*($Q2$) and *result_tuples*($Q1$) $\cup$ *result_tuples*($Q2$) is the same for plain-text and cipher-text queries. Thus, the analogous argumentation as stated in Lemma 5.1 holds here as well. □

5.2.4 Query-Access-Area Distance. The last query-distance measure from literature is query-access-area distance, which is based on the part of the data space accessed by a query.

Definition. The access area of a Query Q is the part of the data space accessed by Q. Hence, it is a generalization of the result of a query that is independent from the current state of the database.

Definition 5.9 (Access Area [23])**.** *Let Q be a query. The access area of Q,* acccess*(Q), contains all tuples that (1) might exist in at least one state allowed by the database, and (2) whose removal from at least one state would change the result of Q.*

Definition 5.10 (Attribute-Access Set of a Query [23])**.** *An Attribute A is in the Attribute-Access Set* Attr_Q *of a Query Q if A occurs in the* FROM*-(as join attribute),* WHERE*-,* GROUP BY*-,* HAVING*- or* ORDER BY*-clause of Q.*

Definition 5.9 defines the access area of *one* query. With the query-access-area distance, we want to compare the access areas of *two* queries. According to [23], this comparison takes place attribute-wise, and every attribute that is in the attribute-access set of one query is considered. Note that Definition 5.10 leaves aside attributes that occur only in the SELECT-clause of a query. Next, we compare access areas regarding the attributes in the access sets.

Definition 5.11 (Access Area Regarding Attribute)**.** *For a Query Q, the access area regarding an Attribute A,* $\text{acccess}_A(Q)$*, is the part of the domain of A accessed by Q.*

To compare the access areas regarding attributes, we distinguish between three cases, as given in Definition 5.12, with an overlap-quantification Value x, which is a design parameter.

Definition 5.12 (Query-Access-Area Distance)**.** *Let $Q1, Q2$ be SQL queries and $Attr_{Q1,Q2}$ the set of attributes accessed by $Q1$ or $Q2$. Then the access-area distance of $Q1$ and $Q2$ is*

$$d_{AE}(Q1, Q2) = \frac{1}{|Attr_{Q1,Q2}|} \cdot \sum_{A \in Attr_{Q1,Q2}} \delta_A(Q1, Q2)$$

where

$$\delta_A(Q1,Q2) = \begin{cases} 0 \text{ if } \text{acccess}_A(Q1) = \text{acccess}_A(Q2) \\ x \text{ if } \text{acccess}_A(Q1) \cap \text{acccess}_A(Q2) \neq \emptyset \\ 1 \text{ otherwise} \end{cases}$$

for $x \in (0,1)$ *with a default value of* 0.5.

To calculate access-area distance, in contrast to result distance, the content of the underlying database is not needed, but the constraints defining the domain of the attributes. See Table 3 and Example 5.4.

Example 5.4. *Consider the Queries* $Q1$ *and* $Q2$ *defined as follows:*

$Q1$ = 'SELECT * FROM R WHERE $A > 5$',
$Q2$ = 'SELECT * FROM R WHERE $A > 3$'.

If the domain of Attribute A in the database is given by $dom(A) = [0, 100]$*, then* $d_{AE}(Q1,Q2) = \delta_A(Q1,Q2) = x$*. In case* $dom(A) = [10, 100]$*, it is* $d_{AE}(Q1,Q2) = \delta_A(Q1,Q2) = 0$.

Access-Area Equivalence. With access-area equivalence, it does matter whether we first encrypt the query and then compute its access area regarding all attributes or vice versa.

Definition 5.13 (Access-Area Equivalence)**.** *Let* Q *be a query and* acccess_Q *its attribute-access set. Then* Enc *ensures access-area equivalence for* Q *if the following holds:*

$$\forall A \in Attr_Q : \text{Enc}(\text{acccess}_A(Q)) = \text{acccess}_A(\text{Enc}(Q)).$$

Lemma 5.4. *Let* Enc *be an encryption algorithm for SQL queries. If* Enc *ensures access-area equivalence, then* Enc *is query-access-area distance-preserving.*

Proof. Definition 5.12 states that Enc is query-access-area distance-preserving if $\delta_A(Q1,Q2) = \delta_A(\text{Enc}(Q1), \text{Enc}(Q2))$ for every Attribute A. If Enc ensures access-area equivalence, this equality is given, with the analogous arguments as in Lemma 5.1. □

5.3 Ensuring Equivalence Notions

In this section, we show how to select the appropriate property-preserving encryption class (cf. Definition 4.2) to ensure all equivalence notions. To this end, we implement the encryption algorithms $\text{Enc}^{A.\text{Const}}$, Enc^{Attr} and Enc^{Rel} for each equivalence notion.

5.3.1 Token Equivalence. For token-based string-distance-preserving encryption, we must ensure token equivalence. Lemma 5.5 lists the appropriate encryption classes.

Lemma 5.5 (Appropriate Encryption Classes for Token Equivalence). *To ensure token equivalence, the following encryption classes are appropriate:*

- $\text{Enc}^{A.\text{Const}}$ = DET,
- Enc^{Attr} = DET and
- Enc^{Rel} = DET.

Proof. For every encryption algorithm, according to Definition 4.2, we have to show that (1) the class ensures token equivalence, and that (2) we cannot use an encryption class with a higher security level. We show this as follows:

Algorithm $\text{Enc}^{A.\text{Const}}$. A constant in a query is an element of the token set of the query, as illustrated in Example 5.2. To facilitate token equivalence, which is an equality check of two sets, we cannot rely on the most secure probabilistic classes (i.e., PROB, SEARCH or HOM), because two constants with the same value will have different cipher-texts. Thus, the constants must be encrypted with a deterministic scheme, and $\text{Enc}^{A.\text{Const}}$ = DET for every Attribute A is appropriate.

Algorithm Enc^{Attr}. With analogous arguments as for $\text{Enc}^{A.\text{Const}}$, Enc^{Attr} = DET is appropriate.

Algorithm Enc^{Rel}. With analogous arguments as for $\text{Enc}^{A.\text{Const}}$ and Enc^{Attr}, Enc^{Rel} = DET is appropriate. □

To conclude, we can find appropriate classes for all three algorithms.

5.3.2 Structural Equivalence. Lemma 5.6 states the appropriate encryption classes for structural equivalence.

Lemma 5.6 (Appropriate Encryption Classes for Structural Equivalence). *To ensure structural equivalence, the following encryption classes are appropriate:*

- $\text{Enc}^{A.\text{Const}}$ = PROB,
- Enc^{Attr} = DET and
- Enc^{Rel} = DET.

Proof. For the three algorithms, the following arguments hold:

Algorithm $\text{Enc}^{A.\text{Const}}$. As Table 4 shows, features of queries do not contain constants. In particular, feature extraction removes the constants from the predicates. Thus, since constants are irrelevant for feature sets of queries, the choice of $\text{Enc}^{A.\text{Const}}$ = PROB for any Attribute A is appropriate, which is the encryption class with the highest level of security.

Algorithm Enc^{Attr} and Algorithm Enc^{Rel}. Attribute and relation names are part of features (cf. Table 4). Analogously to token equivalence, deterministic schemes appropriate. □

To conclude, we can find appropriate classes for all three algorithms.

5.3.3 Result Equivalence. A recent proposal dubbed CryptDB proposed in [27] already ensures result equivalence. A core observation in [27] is that the selection of the encryption class of $\text{Enc}^{A.\text{Const}}$ depends on Attribute A and its use in query predicates.

Example 5.5 (Ensuring Result Equivalence)**.** *Consider the Predicate 'A = 5'. Here, a deterministic encryption scheme for* $\text{Enc}^{A.\text{Const}}$ *is sufficient. Next, consider the Predicate 'A ≤ 5'. Now, an order-preserving scheme is needed. If A is used within the* SUM*-aggregate function, i.e.,* SUM*(A),* and *in a range query, a homomorphic* and *an order-preserving scheme is needed. In this case,* $\text{Enc}^{A.\text{Const}}$ *consists of two schemes. I.e., in the database, the attribute is encrypted with both encryption schemes, and during encrypting the query, the correct scheme depending on the predicate is used.*

To conclude, we can use CryptDB to find appropriate encryption classes for $\text{Enc}^{A.\text{Const}}$. However, CryptDB is a *database* encryption approach and supports ad-hoc queries over encrypted data. As we encrypt *fixed* SQL logs, we can modify CryptDB's approach, resulting in a higher level of security, as we will describe later on. For instance, we can use stateful schemes to instantiate the classes.

Lemma 5.7 (Appropriate Encryption Classes for Result Equivalence). *To ensure result equivalence, the following encryption classes are appropriate:*

- $\text{Enc}^{A.\text{Const}}$ as stated in CryptDB [27],
- Enc^{Attr} = DET and
- Enc^{Rel} = DET.

Proof. For the three algorithms, the following arguments hold:

Algorithm $\mathsf{Enc}^{A.\mathbf{Const}}$. Acording to the occurrences of Attribute A in predicates, as described in CryptDB [27].

Algorithms $\mathsf{Enc}^{\mathbf{Attr}}$ and $\mathsf{Enc}^{\mathbf{Rel}}$. To ensure result equivalence, one must preserve the names of relations and attributes, to ensure that query execution accesses the correct relations and attributes. Thus, a deterministic scheme is appropriate. □

Query-result distance differs from the previous measures: For the previous ones, one only has to share the log. For result distance in turn, the encrypted content of the database must be shared as well. To this end, it is necessary to encrypt not only the log, but also the database. Thus, we use $\mathsf{Enc}^{A.\mathrm{Const}}$, $\mathsf{Enc}^{\mathrm{Attr}}$ and $\mathsf{Enc}^{\mathrm{Rel}}$ for database encryption as well. As we will see in Section 5.4, our security model captures the security of the log file *and* the database.

Differences to CryptDB. We see two differences to our approach that give way to achieve a higher level of security:

- As stated in Section 2.1.1, we can use stateful or stateless encryption schemes. In turn, CryptDB cannot rely on stateful encryption schemes. For attributes encrypted with an OPE scheme, this yields a higher security level. Experimental results in [27] indicate that up to 15.4% of the attributes occurring in real-world logs are encrypted with an OPE scheme.
- In our setting, the organization does not have to encrypt/share the whole database, but only the relations/attributes accessed by queries in the log.

Discussion. In contrast to token and structural equivalence, not all SQL queries allowed by the SQL standard can be encrypted in a way that ensures result equivalence [27]. This is an inherent issue we must be aware of. For instance, queries containing arbitrary pattern-matching LIKE predicates besides keyword search, i.e., predicates that SEARCH schemes can handle, are not supported. However, the results in [27] suggest that this affects only a small share of queries occurring in a real-world log. Experiments conducted there show that, except for a query log over a calendar database containing many string operations, at most 1.2% of the attributes in the database cannot be encrypted, over five data sets.

5.3.4 Access-Area Equivalence. In contrast to result equivalence, to ensure access-area equivalence, one has to preserve the result tuples of every query with regard to every possible state of the database. Nevertheless, we can choose nearly the same encryption classes as for result equivalence, see Lemma 5.8. The only special cases are attributes that occur only in the arithmetic aggregate functions SUM and AVG in the SELECT-clause, i.e., not anywhere else in a query in the log. Hence, this generally leads to a higher security level. However, as described in Section 2.2, the difference is only relevant when considering active attacks.

Lemma 5.8 (Appropriate Encryption Classes for Access-Area Equivalence). *To ensure access-area equivalence, the following encryption classes are appropriate:*

- $\mathsf{Enc}^{A.\mathrm{Const}} = \begin{cases} \mathsf{PROB} & \text{if } A \text{ occurs in SUM/AVG in SELECT only} \\ \text{as stated in CryptDB [27]} & \text{otherwise} \end{cases}$
- $\mathsf{Enc}^{\mathrm{Attr}} = \mathsf{DET}$ and
- $\mathsf{Enc}^{\mathrm{Rel}} = \mathsf{DET}$.

Proof. The same arguments as in Lemma 5.7 apply, as the proof is based on the semantics of the operators of the relational algebra (e.g., selections) whose access space is independent from the state of the database. The only exception from Lemma 5.7 is the first case of $\mathsf{Enc}^{A.\mathrm{Const}}$. The reason is as follows: For result equivalence, we must choose the HOM-class, if an arithmetic aggregate function is used in a query in the log. For access-area equivalence, we do not have to and can use (the most secure) PROB-class, because the semantics of the aggregate function in the SELECT-clause does not have any influence on the access area. This area only depends on the grouping applied. This also holds if there is no grouping. Thus, in case of an attribute to which only queries with an aggregate function in the SELECT-clause refer to, we use the PROB class. □

As for result equivalence, we can use stateful encryption to instantiate the encryption classes. In contrast to result equivalence, we do not have to encrypt the content of the database, but only need to include the minimum and maximum values of the attribute domains of the attributes, cf. Example 5.4.

5.4 Security Assessment

As we have used known property-preserving encryption schemes from literature, their level of security is known, and we can reduce the security of the schemes to the security of the encryption schemes for SQL queries we have just specified.

Example 5.6 (Security Assessment)**.** *As the Tables 1 and 3 show, we often use the* DET*-class for encryption and recommend the* AES *scheme to instantiate it. As mentioned in Section 2.2,* DET *schemes are not secure against chosen-plain-text attacks. But so far, there does not exist any successful Known-Plain-Text or Cipher-Text-Only Attack against* AES[2] *that is executable in reasonable time, so we can use the scheme here without any disadvantage.*

In particular, as shown in [29] and stated once more in Lemma 5.9, an encrypted database is as least as secure against a query log attack as our SQL query encryption schemes are against attacks on the database. In other words, if the same encryption schemes are used, an attacker cannot infer more information from the encrypted query log than from an encrypted database.

Lemma 5.9 (Security Ass of Query Log Attacks [29]). *Let $\mathcal{A} \in$ {Cipher-Text-Only Attack, Known-Plain-Text Attack, Chosen-Plain-Text Attack}. Then a database is at least as secure against $\mathcal{A}$ in terms of query-log attacks as against $\mathcal{A}$ in terms of database attacks.*

As we have used property-preserving encryption schemes known from literature, their level of security is known. Thus, we can reduce the security of the schemes to the security of the encryption schemes for SQL queries we have just specified [29]. In particular, one can adapt the results from [22] analyzing the security of databases encrypted with property-preserving encryption schemes. This is intended – executing a full security analysis for organizations that want to outsource data analysis is practically impossible.

To conclude, we have achieved distance-preserving encryption for every distance measure with the highest security level possible.

[2] https://www.schneier.com/blog/archives/2012/03/can_the_nsa_bre.html (accessed 09/2017)

6 GENERALIZATION OF OUR STUDY

Now, we examine how to reuse our concepts when designing distance-preserving encryption schemes for data different from SQL logs. Our main objective is to see which parts of the proof are generally applicable, and which extra effort is necessary to apply DisPE. We illustrate our thoughts by strictly following the steps of DisPE with two example data sets. The first one are logs of queries in another language, namely XQuery, and the second one is data stored in a database relation, called relational data in the remainder.

6.1 Security Model

As mentioned, the security model definition step consists of two parts: (1) specifying the threat model, and (2) definition of a high-level encryption scheme containing all algorithms one has to implement. Regarding the first part, the justification why we consider passive attackers only is independent of the data and only depends on the scenario. Hence, it is generally valid.

The high-level encryption highly depends on the type of data considered. However, a complex data item consists of two parts: (a) a complex type consisting of several elements, which in turn may have a complex or atomic type and is formed according to a grammar and (b) the data associated with the atomic elements. In most settings, it is sufficient to encrypt (b), e.g., the relation and attribute names and the constants in an SQL query, as they contain the confidential information. Examples 6.1 and 6.2 illustrate this.

Example 6.1 (High-Level Encryption Scheme for XQuery)**.** *An XQuery statement contains names of XML-elements and attributes as well as attribute values that are confidential. Thus, we define the high-level encryption scheme for (1) XML-element names, (2) attribute names and (3) attribute values for all attributes. As a result, we have a tuple of encryption schemes with three components, the same as for SQL queries.*

Example 6.2 (High-Level Encryption Scheme for Relational Data)**.** *Think of customer data stored in a table with relation and attribute names as well as attribute values that are confidential. Thus, we specify the high-level encryption scheme consisting of an encryption scheme for (1) relation names, (2) attribute names and (3) values of every attribute. As a result we have a tuple of encryption schemes with three components, very similar to the one for SQL queries.*

6.2 Suitable Equivalence Notions

An equivalence notion is always defined for a specific distance measure. However, as Examples 6.3 and 6.4 illustrate, the distance measures for SQL queries may be useful for other data sets as well in many cases, or variations are used, as Example 6.5 illustrates. In other words, the equivalence notions introduced earlier are meaningful in broader contexts.

Example 6.3 (Equivalence Notions for XQuery.)**.** *Token-based string distance, structure and result distance also are suitable distance measures for XQuery logs. Therefore, we can leverage the respective equivalence notions. Access-area distance in turn is not suitable here: In general, there is no underlying schema. Hence, it is not obvious at all how to generalize the notion of "access area" for the XQuery setting.*

Example 6.4 (Equivalence Notions for Relational Data – Same Distance Measures)**.** *For relational data, token and structure equivalence are applicable, while result and access-area equivalence are specific for query languages.*

Example 6.5 (Equivalence Notions for Relational Data – New Distance Measure)**.** *Suppose that one wants to use the token-based distance to analyze the data, but Attribute A should be excluded, i.e., is not part of the token set. Thus, we have a variant of the token-based distance measure and need a variant of token equivalence.*

6.3 Ensuring Equivalence Notions

We cover three cases, describing the relationship of the equivalence notions defined in Step 2 to our SQL equivalence notions, ordered by the extent of reuse options, in a decreasing manner:

a) Same Equivalence Notions. If the token or structural equivalence is used, one can use the same encryption schemes as in our case study in Section 5. For instance, this is the case for all distance measures or equivalence notions for the customer data. For result and access-area equivalence, one cannot directly adapt the schemes for $\mathsf{Enc}^{A.\mathrm{Const}}$ for SQL queries, i.e., the CryptDB-approach, because of different execution semantics.

Example 6.6 (Ensuring Equivalence Notions for XQuery)**.** *A core difference between SQL and XQuery is the use of path expressions, i.e., XPath. For result equivalence, one must ensure that the evaluation of XPath location steps is the same on plain-text and on cipher-text data. As we preserve the tree structure of the XML-trees (cf. Example 6.1), "preserving the locations steps" only implies the usage of deterministic schemes for relation and attribute names and ensuring the correct evaluation of predicates, as for SQL.*

b) Variants of the Equivalence Notions. As Example 6.5 illustrates, one tends to use variants of the equivalence notions introduced in our study. In this case, one can reuse parts the encryption schemes the high-level scheme consists of.

Example 6.7 (Ensuring Equivalence for Relational Data)**.** *Consider the equivalence notion from Example 6.5. It differs from token equivalence in that the distance measure does not use Attribute A. Thus, we can encrypt the attribute values of the Attribute A with a probabilistic scheme, i.e.,* $\mathsf{Enc}^{A.Const} = \mathsf{PROB}$*, and the other encryption algorithms are the same as for SQL queries.*

c) Other Equivalence Notions. In case other equivalence notions are used, one must implement the encryption scheme anew. To this end, one can rely on our notions of appropriateness (Definition 4.2) and encryption-class taxonomy.

6.4 Security Assessment

As long as one uses well-known security schemes to ensure the equivalence notions, such as the ones in our taxonomy in Figure 1, the assessment of the security is known. In particular, one can adapt the reduction of Lemma 5.9 to other types of data. Otherwise, a full-fledged security analysis is needed, as, for instance, in [9] for order-preserving encryption.

7 EXPERIMENTS

Section 3.2 shows with pathological examples that ensuring the equality of distances upon encryption is necessary to guarantee that we have the same mining results on plain-text and cipher-text data. In this section, we substantiate these findings with experiments on real data. We show that preserving the exact distances upon encryption is necessary to deploy encryption schemes independent of the specific data-mining algorithm in use. In the following, we first describe our experimental setup, then the results.

7.1 Experimental Setup

We describe the experiments, the data and algorithms used.

Experiments. To substantiate our findings from Section 3.2, we conduct two Experiments *E1* and *E2* on both alternative notions:

(*E1*) We quantify how much preserving an ϵ-approximation of the distances upon encryption falsifies the results of distance-based data mining.

(*E2*) We quantify how much preserving the relative order of distances upon encryption falsifies the results of distance-based data mining.

Both experiments follow the following procedure: First, we determine the distance matrix of the items of our data sets. We name this matrix the *reference matrix*. Then we generate *distorted matrices* as follows: In *E1*, we distort the values in the reference matrix by adding noise so that we have approximations of the distances, for different approximation values ϵ. In *E2*, inspired by [33], we distort the values by using the i-stretching Function $s_i(v) = v^i$. Since $d(x, y) \geq 0$, the Function $s_i(d(x, y))$ increases monotonically, thus, distortion preserves the relative order of the distances. Finally, we compare the results of different distance-based mining algorithms on (a) the reference matrix and (b) distorted matrices. We always use the same parameter settings of the algorithms for (a) and (b).

Data Sets. In total, we conduct both experiments on three data sets. We use two synthetic data sets – a Uniform and a Gaussian-distributed data set – as well as one real-world data set [5, 30]. All three have Dimensionality dim = 16 and consist of 10,992 data items. We normalize the distances with min/max normalization to the interval [0,1]. Figure 2a graphs the distribution of the distances in the reference matrices. We see that this distribution in the reference matrices are different for our three data sets. This indicates that our results will be somewhat general. To get an intuition regarding the influence of the distortion in our experiments, see Figure 2b for Experiment *E1* and 2c for Experiment *E2*. The distortion in *E1* leads to a uniformer distribution of the distances, the one in *E2* to denser distribution of the data items. The Figures 2b and 2c only graph the distributions for the real world data set, but the distributions of the distorted distance matrices for the Uniform and Gaussian distributed data set are similar.

Data-Mining Algorithms. We conduct the experiments with two types of distance-based data mining algorithms: clustering and outlier detection. For clustering, we use the k-medoids [26] algorithm with Euclidean distance and Parameter $k = 10$. For outlier detection, we use the OPTICS-OF [11] algorithm. We select the parameters of OPTICS-OF so that they are optimal for the original data.

7.2 Results

Figure 3 graphs the results of our experiments for all algorithms and data sets. We use the following error measures: For k-medoids algorithm, we determine the fraction of data items assigned to the wrong cluster. For OPTICS-OF algorithm, we use the false negative rate (FNR), i.e., the rate of outlier classified as inlier. – We now describe the results of both experiments, followed by a discussion.

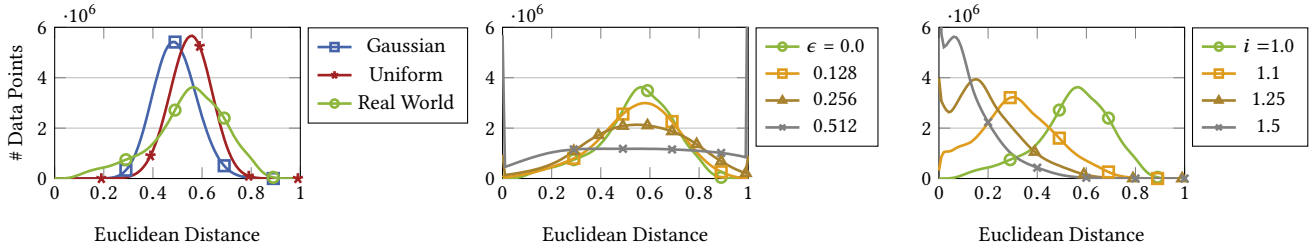
Experiment E1. Figure 3a graphs the results of *E1*. As expected, for almost all data sets and algorithms, the error increases with increasing distortion, i.e., ϵ-values, from 0 until (nearly) 1. An error value of 1 is the highest possible error value. Thus, even for small values of ϵ, the falsification of the clustering result is high.

Experiment E2. Figure 3b shows the results of *E2*. Again, the error increases with higher distortion. Differently from *E1*, the error values do not exploit the full range of [0,1]. In particular, the error of k-medoids clustering remains small, i.e., ≤ 0.3. In contrast, the error with OPTICS-OF is nearly constantly 1, i.e., maximal. The reason is that stretching leads to smaller distances (cf. Figure 2c), i.e., denser regions. The data items are moving closer together, and the outliers move inwards as well.

Discussion. Preserving approximation distances or the relative ordering of the distances upon encryption falsifies the results of distance-based data mining algorithms, but to very different extents. While preserving approximation distances leads to high falsification for both algorithms, the falsification while preserving the relative ordering depends on the specific algorithm. This implies that, if one does not preserve the exact distances upon encryption, finding an appropriate encryption scheme not only depends on the distance measure, but *also on the specific algorithm in use*. This is undesirable: Organizations typically need different analyses, with different algorithms, from a service provider, and the necessity of transmitting several encrypted variants of the data would be difficult to impossible to convey. Our conclusion is that distance-preserving encryption is the only feasible option.

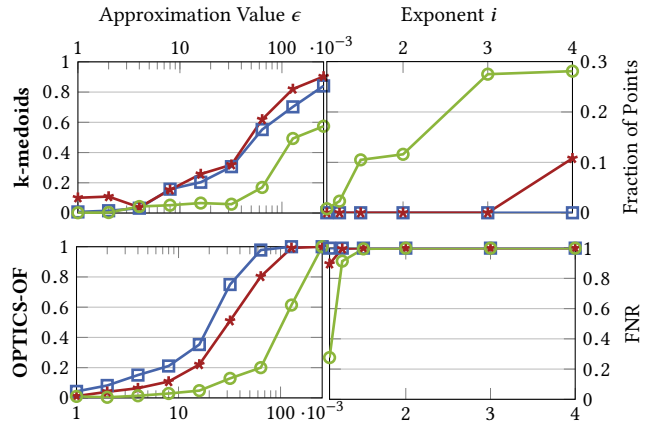
8 CONCLUSIONS

Many organizations have data that contains valuable information, but cannot analyze it themselves. Therefore, they outsource the analysis to a service provider. To ensure confidentiality, they are willing to transfer their data only if it is encrypted. To this end, it is important that the encryption preserves the mining results. Popular distance-based data mining algorithms such as k-medoids [26] are particularly relevant. Therefore, it is important to preserve the mining results for them. To overcome this issue, we introduce distance-preserving encryption (DPE). With formal arguments, as well as experiments, we prove that preserving the exact distances is the only feasible option. To deploy (DPE) schemes, we present the DisPE-procedure. It says how to design a DPE scheme for arbitrary data and distance measures. The procedure involves defining and ensuring equivalence notions, which capture a characteristic of data that should be preserved upon encryption. We then instantiate this procedure for SQL query logs. In this study, we find appropriate DPE schemes for all four prominent distance measures from literature. In any case, we use well-known property-preserving encryption classes to implement the DPE schemes and assess their security.



(a) Reference Matrix. **(b) Distorted Matrices in *E1*.** **(c) Distorted Matrices in *E2*.**

Figure 2: Distribution of the Distances in the Reference Matrices, and the Distorted Matrices for the Real-World Data Set.



(a) Experiment E1. **(b) Experiment E2.**

Figure 3: Results of our Experiments.

This is different from approaches supporting ad-hoc queries like CryptDB [27] and gives a way to higher security levels. Finally, we demonstrate how to reuse of parts of our study by the examples of XQuery and of relational data. Furthermore, we can even reuse the equivalence notions defined and their DPE schemes beyond DisPE when mining encrypted data according to other paradigms. For instance, result equivalence for SQL queries is also useful for association-rule mining over encrypted SQL logs [4]. Studying the applicability of equivalence notions in different contexts also offers interesting opportunities for future work. For example, ensuring token equivalence is appropriate for cleaning SQL antipatterns over encrypted data [6].

ACKNOWLEDGMENTS

This work was partially supported by the German Research Foundation (DFG; ref. nb. BO 2129/13-1) and the Federal Ministry of Education and Research (BMBF; ref. nb. 02K15A024).

REFERENCES

[1] M. Agrawal and P. Mishra. 2012. A Comparative Survey on Symmetric Key Encryption Techniques. *IJCSE* (2012).
[2] R. Agrawal et al. 2004. Order-Preserving Encryption for Numeric Data. In *SIGMOD*. ACM.
[3] J. Akbarnejad et al. 2010. SQL QueRIE Recommendations. *PVLDB* (2010).
[4] J. Aligon et al. 2011. Mining Preferences from OLAP Query logs for Proactive Personalization. In *ADBIS*. Springer.
[5] F. Alimoglu and E. Alpaydin. 1996. Methods of Combining Multiple Classifiers Based on Different Representations for Pen-Based Handwritten Digit Recognition. In *TAINN*. Citeseer.
[6] N. Arzamasova, M. Schäler, and K. Böhm. 2018. Cleaning Antipatterns in an SQL Query Log. *TKDE* (2018).
[7] M. Bellare et al. 1997. A Concrete Security Treatment of Symmetric Encryption. In *FOCS*. IEEE.
[8] A. Boldyreva et al. 2009. Order-Preserving Symmetric Encryption. In *EUROCRYPT*. Springer.
[9] A. Boldyreva, N. Chenette, and A. O'Neill. 2011. Order-Preserving Encryption Revisited: Improved Security Analysis and Alternative Solutions. In *CRYPTO*. Springer.
[10] D. Boneh et al. 2004. Public Key Encryption with Keyword Search. In *EUROCRYPT*. Springer.
[11] M. Breunig et al. 1999. OPTICS-OF: Identifying Local Outliers. In *PKDD*. Springer.
[12] W. Cohen, P. Ravikumar, and S. Fienberg. 2003. A Comparison of String Metrics for Matching Names and Records. In *KDD Workshop*.
[13] J. Daemen and V. Rijmen. 2013. *The Design of Rijndael: AES-the Advanced Encryption Standard*. Springer.
[14] D. Defays. 1977. An Efficient Algorithm for a Complete Link Method. *Comput. J.* (1977).
[15] W. Diffie and M. Hellman. 1976. New Directions in Cryptography. *Trans. Inf. Theory* (1976).
[16] M. Ester et al. 1996. A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases With Noise. In *KDD*. AAAI Press.
[17] C. Fontaine and F. Galand. 2007. A Survey of Homomorphic Encryption for Nonspecialists. *EURASIP* (2007).
[18] J. Han, J. Pei, and M. Kamber. 2011. *Data Mining: Concepts and Techniques*. Elsevier.
[19] N. Khoussainova et al. 2010. SnipSuggest: Context-aware Autocompletion for SQL. *PVLDB* (2010).
[20] E. M. Knorr et al. 2000. Distance-Based Outliers: Algorithms and Applications. *VLDBJ* (2000).
[21] J. Li et al. 2015. L-EncDB: A Lightweight Framework for Privacy-Preserving Data Queries in Cloud Computing. *Knowl.-Based Syst.* (2015).
[22] M. Naveed, S. Kamara, and C. V. Wright. 2015. Inference Attacks on Property-Preserving Encrypted Databases. In *SIGSAC*. ACM.
[23] H. Nguyen et al. 2015. Identifying User Interests Within the Data Space - A Case Study with SkyServer. In *EDBT*.
[24] P. Paillier et al. 1999. Public-Key Cryptosystems Based on Composite Degree Residuosity Classes. In *EUROCRYPT*. Springer.
[25] O. Pandey and Y. Rouselakis. 2012. Property-Preserving Symmetric Encryption. In *EUROCRYPT*. Springer.
[26] H.-S. Park and C.-H. Jun. 2009. A Simple and Fast Algorithm for K-Medoids Clustering. *Expert Syst. Appl.* (2009).
[27] R. Popa et al. 2011. CryptDB: Protecting Confidentiality with Encrypted Query Processing. In *SOSP*. ACM.
[28] D. Purnamasari et al. 2016. Query Rewriting and Corpus of Semantic Similarity as Encryption Method for Documents in Indonesian Language. In *ICESTI*. Springer.
[29] T. Sanamrad and D. Kossmann. 2013. Query Log Attack on Encrypted Databases. In *SDM Workshop*. Springer.
[30] M. Schäler et al. 2013. QuEval: Beyond High-Dimensional Indexing à la Carte. *PVLDB* (2013).
[31] D. X. Song, D. Wagner, and A. Perrig. 2000. Practical Techniques for Searches on Encrypted Data. In *Security and Privacy*. IEEE.
[32] C. Tex, M. Schäler, and K. Böhm. 2018. Distance-Based Data Mining Over Encrypted Data. In *ICDE*. IEEE.
[33] W. K. Wong et al. 2009. Secure KNN Computation on Encrypted Databases. In *SIGMOD*. ACM.