Confidentiality-Preserving Query Execution of Fragmented Outsourced Data

Anis Bkakria[1], Frédéric Cuppens[1], Nora Cuppens-Boulahia[1], and José M. Fernandez [2]

[1] Télécom Bretagne
[2] École Polytechnique de Montréal
{anis.bkakria,frederic.cuppens,nora.cuppens}@telecom-bretagne.eu, jose.fernandez@polymtl.ca

Abstract. Ensuring confidentiality of outsourced data continues to be an area of active research in the field of privacy protection. Almost all existing privacy-preserving approaches to address this problem rely on heavyweight cryptographic techniques with a large computational overhead that makes inefficient on large databases. In this paper, we address this problem by improving on an existing approach based on a combination of fragmentation and encryption. We present a method for optimizing and executing queries over distributed fragments stored in different Cloud storage service providers. We then extend this approach by presenting a Private Information Retrieval (PIR) based query technique to enforce data confidentiality under a collaborative Cloud storage service providers model.

Keywords: Data confidentiality, Privacy-preserving, Data fragmentation, Data outsourcing.

1 Introduction

In the last few years, database outsourcing has become an important tool in IT management, as it offers several advantages to the client companies, especially for small ones with limited IT budget. In most models of database outsourcing, data storage and management (e.g. data backup and recovery) are completely operated by external service providers that take advantages of economies of scale to reduce the cost of maintaining computing infrastructure and data-rich applications in comparison with the cost of in-house data management.

Nonetheless, outsourcing gives rise to major security issues due to the usually sensitive nature of information saved in databases. Storing such sensitive information with external service providers requires them to be trusted and that they should protect data against external threats. Even if these service providers protect data adequately and process queries accurately, they may be curious about the stored data and may attempt to profit by disclosing this sensitive information to third parties. This raises the question of whether it is possible to protect confidentiality and privacy of the outsourced data. A recent study [10]

K. Mustofa et al. (Eds.): ICT-EurAsia 2013, LNCS 7804, pp. 426–440, 2013.

by the Ponemon Institute indicates that since 2005 more than 250 million customer records containing sensitive information have been lost or stolen. The same study reports the average organizational cost of data leakage between 2005 and 2009, to be about $6.6 million. Hence, there seems to be a clear financial motivation to protect data privacy and confidentiality even when records are leaked.

Related Work. One approach to protect confidentiality and privacy of the outsourced data is based on encrypting all tuples in outsourced databases [8, 7]. With this approach, one crucial question is that of how to efficiently execute queries. Clearly, when using deterministic encryption techniques equality-match queries are simple to evaluate. However, range queries and aggregations become more difficult to perform as we must decrypt all records to evaluate this kind of queries, which makes query execution on the outsourced encrypted data much more difficult. Therefore, our main focus in this paper is about preserving both privacy and confidentiality of outsourced database while ensuring a secure way for querying outsourced data. One promising method to meet this requirement is to use data fragmentation. Basically, data fragmentation procedures are not particularly designed for preserving data security, they are aimed to improve data manipulation process, optimize storage, and facilitate data distribution. Nevertheless, in the last few years two significant alternatives have been proposed. The first one [9, 3] relies on data fragmentation alone to protect confidentiality. The distribution model used in this approach is composed mainly of two domains: a trusted local domain from which the data originates, and a honest but curious domain in which the data will be distributed. Because of its trustworthiness, the local domain is used to store fragments that contain highly sensitive data without the need to encrypt them. This is not so efficient in that it forces data owners to always protect and manage fragments containing highly sensitive data. A more promising alternative [4, 6] attempts to combine data fragmentation together with encryption. In this proposal, the main idea is to partition the data to be externalized across two or more independent service providers, and furthermore to encrypt all information which can not be secured by fragmentation (e.g. employees' bank account numbers of a company). While they do constitute an interesting way to ensure confidentiality of outsourced database, these approaches have the major limitation that it assumes that data to be outsourced is represented within a single relation schema (or table). Clearly, this assumption is too strong and seldom satisfied in the real environments, as generally, relational databases are normalized to minimize redundancy and dependency by dividing large tables into smaller (and less redundant) tables and defining relationships between them.

In this paper, we strive to protect the confidentiality and the privacy of sensitive outsourced database using both encryption and fragmentation, combining the best features of both approaches. Furthermore, we aim to overcome the previously mentioned limitations of [4, 6] by presenting an approach which is able to deal efficiently with multi-relation normalized databases. In a distributed

environment, the problems encountered in one-relation[1] databases take on additional complexity when working with multi-relation normalized databases as it gives rise to new problems such as protecting the relatioships between relational schemas and defining a secure and efficient technique allowing authorized users to query these sensitive relationships. We will show how to protect data unlinkability of different fragments of the original database by protecting user query privacy using a practical Private Information Retrieval (PIR) technique. Unlinkability of two items of interest (e.g., records stored into different fragments) means that within the system, from an adversary point of view, these items of interest are no more and no less related. In our approach, a relation containing sensitive information will be fragmented into two or more fragments. Unlinkability of fragments means that despite the fact that an adversary has knowledge about the fragments of a relation, it remains unable to link records from different fragments.

The remainder of the paper is organized as follows. Section 2 illustrates through an example the problem and need for an approach like ours. In Section 3, we detail the threat model, security model, and assumptions. Section 4 describes our approach to enforce privacy and confidentiality of outsourced data. Section 5 presents the query optimization and execution model. In Section 6, we present a PIR-based technique to achieve query privacy and enforce data confidentiality under a collaborative Cloud storage service providers model. Finally, we conclude the paper in section 7.

2 Motivating Example

Consider a relational hospital database D with relations: **Patient**(Id, Name,ZIP, Illness, Id_Doctor *) and **Doctor**(Id, Name, Specialty) where Id_Doctor is a foreign key referencing the column Id of the relation $Doctor$. Let us assume that the hospital is going to outsource the database to a third party. Sensitive information contained in D should be protected. If we look carefully, we can consider that the list of patients and their attributes $(Id, Name, Zip)$ are not sensitive, and also that the list of illnesses could be made public. Nevertheless, the relationship between these two lists (list of patients and list of illnesses) should be protected. Therefore if we can find a way (e.g. vertical fragmentation [11]) to break relationships between patients and their respective illnesses, there is no need to encrypt all records of the $Patient$ relation. On the other hand, the relationship between a patient and his doctor should be protected. Since the list of doctors itself is not sensitive, the simplest way to protect the relationship between the two relations $Patient$ and $Doctor$ consists in encrypting the foreign key Id_Doctor. Actually, we can either encrypt the foreign key Id_Doctor of the relation $Patient$ or the primary key Id of the relation $Doctor$, because in the two cases, relationship between relations $Patient$ and $Doctor$ will be protected. However, encrypting the foreign key seems to be more beneficial as a foreign key references only one relation (only the relationship between the two relations is

[1] Databases composed from a single relation shema.

protected) while a primary key can be referenced by many relations. Therefore, if we encrypt the primary key, we will protect all relationships between the relation containing the primary key and other relations referencing the encrypted primary key. Thus, when the security requirement specifies that only the a relationship between data is sensitive, our apporach is more appropriate than the one based on full encryption.

3 Threat Model and Security Assumptions

Our approach is based on a typical client-server architecture, where servers are managed by different service providers. These service providers are considered "honest-but-curious", in agreement with most related work [3, 4, 7, 9]. Service providers are assumed to be "honest" in that they do not manipulate outsourced data in order to respond incorrectly to user queries. In other words, we suppose that responses to user queries received from these service providers are always accurate. In the first part of this paper, we will assume that service providers are "curious" in that they will try to infer and analyze outsourced data, and will also actively monitor all received user queries and try to derive as much information as possible from these queries. In the second part of the paper, we further assume that service providers can collude and cooperate together to link outsourced data. The client part of this architecture is assumed to be trustworthy and all interactions between the user and the client are secured. Protecting the client part against external attacks is beyond the scope of this article.

4 Confidentiality Using Fragmentation and Encryption

Our approach extends in several ways the vertical fragmentation-based approach described in [4, 6]. This approach considers that all data is stored in a single relation, while in our approach data can be stored in several relations, which is the case in typical database environments. In our approach, we consider that databases to be externalized are normalized so that two relations can be only associated together through a primary key/foreign key relationship. For this purpose, we introduce a new type of confidentiality constraint for fragmentation, the *inter-table fragmentation constraint*. The aim of this new fragmentation constraint is to protect the relationship between relations. This section first presents the different kinds of confidentiality constraints used to achieve our goals of protecting the confidentiality by encryption and fragmentation, and second formalises the concept of fragmentation in our approach which extends ideas presented in [4–6].

Definition 1 (Confidentiality Constraint). Consider that data to be secured are represented with a relational database D, which is composed of a list of relational schemas $R = (R_1, \ldots, R_n)$, with each of these relational schemas R_i containing a list of attributes $A_{R_i} = (a_{1,i}, a_{2,i}, \ldots)$. A confidentiality constraint over D can be one of the following:

Singleton Constraint (SC). It is represented as a singleton set $SC_{R_i} = \{a_{j,i}\}$ over the relation R_i. This kind of confidentiality constraint means that the attribute $a_{j,i}$ of the relational schema R_i is sensitive and must be protected, typically by applying encryption.

Association Constraint (AC). This kind of confidentiality constraint is represented as a subset of attributes $AC_{R_i} = \{a_{1,i}, \ldots, a_{j,i}\}$ over the relational schema R_i. Semantically, it means that the relationship between attributes of the subset AC_{R_i} is sensitive and must be protected.

Inter-table Constraint (IC). It is represented as a couple of relational schemas $IC = \{R_i, R_j\}$ of the relational database D. Relations R_i and R_j should be associated through a primary key/foreign key relationship. The use of this kind of confidentiality constraint ensures protection of the primary key/foreign key relationship between the two relational schemas concerned with the inter-table constraint IC.

Note that protecting the relationship between two tables relies on protecting the primary key/foreign key relationship and storing involved relations separately. The association constraint can also be addressed through encryption (encrypt at least one of attributes involved in the constraint), but clearly this will increase the number of encrypted attributes and make interrogation of the database more complicated. A more adapted way to resolve this kind of confidentiality constraint was proposed in [4], which is based on splitting involved attributes in a manner that their relationships cannot be reconstructed.

In the case of an inter-table confidentiality constraint, protecting the foreign key using encryption is the simplest way to secure the relationship between the two relational schemas. However encrypting only the foreign key is not enough to keep the relationship between relational schemas secure, as service provider may be able to link records in two relational schemas by observing and analyzing user queries over these relational schemas. To overcome this problem, the two relational schemas in question should be split into different fragments, and each of these fragments should be distributed to a different Cloud storage provider. An interesting approach for modeling constraints and resolving the data fragmentation problem was proposed in [6], that efficiently computes data fragments that satisfy the confidentiality constraints. It is based on Boolean formulas and Ordered Binary Decision Diagrams (OBDD) and uses only attribute-based confidentiality constraint (Singleton Constraints and Association Constraints). However, it cannot deal as-is with Inter-table Constraints. In order to use this approach, we define a way to reformulate Inter-table Constraint as a set of Singleton Constraints and Association Constraints. We explain this transformation in the definition and theorem below.

Definition 2 (Inter-table Constraint transformation). Consider a relational database with two relations $R_1(\underline{a_1}, \ldots, a_n)$ and $R_2(\underline{b_1}, \ldots, b_m*)$. Let us assume that R_1 and R_2 are related through a foreign key/primary key relationship in which the foreign key b_m of the relation R_2 references the primary key a_1 of relation R_1. We assume that R_1 and R_2 contain respectively p and q records,

with $p > 1$ and $q > 1$. An Inter-table Constraint $c = \{R_1, R_2\}$ over relations R_1 and R_2 states that the relationship between these two relations must be protected by encrypting the foreign key b_m and by storing R_1 and R_2 in two different fragments. Therefore, the constraint c can be written as follows:

i A singleton constraint $SC = \{b_m\}$ to state that the value of b_m should be protected.
ii A list of $(m \times n)$ association constraints $AC = \{(a_i, b_j) | i \in [1, n],\ j \in [1, m]\}$.

We propose the notion of a correct transformation of Inter-table constraints. A transformation of an Inter-table constraint c to a set of confidentiality constraints C is correct if the satisfaction of C enforce the protection of the unlinkability between records of the two relations involved in c. The following Theorem formalizes this concept.

Theorem 1 (Transformation correctness). Given a relational database $\mathcal{D}$ composed from two relational schemas $R_1(\underline{a_1}, \ldots, a_n)$ and $R_2(\underline{b_1}, \ldots, b_m*)$ related through relationship between the foreign key b_m of R_2 and the primary key a_1 of R_1. Let $c = \{R_1, R_2\}$ be an Inter-table constraint, the set of constraints C be the result of the transformation of c, and $\mathcal{F} = \{F_1, \ldots F_q\}$ be a fragmentation of $\mathcal{D}$ that satisfies C. The Inter-table constraint c is correctly transformed into a set of constraints C if all the following conditions hold :

1. b_m does not appear in clear in any fragment of $\mathcal{F}$.
2. $\forall\ AC_{i,j} = \{a_i, b_j\} \in C, i \in [1, n], j \in [1, m],\quad if\quad a_i \in F_k\quad and\quad b_j \in F_l\quad then\ k \neq l$

The main advantage of the Inter-table constraint is that it allows treatment of multi-table relational databases. In addition, it gives a simple way to formulate confidentiality constraints between relations. As we have seen in Item (i) of Definition 2, the attribute b_m (foreign key of the relation R_2) should be encrypted. However, to be able to query data and construct relationship between relations, the chosen encryption algorithm must be deterministic [1] in order to preserve uniqueness and allow the construction of relationship between relations (e.g. through JOIN queries). As is known, in normalized multi-relation databases, three types of relationship between relations exist: (1) one-to-one, (2) one-to-many and (3) many-to-many relationships. Inter-table constraints over relations associated using (1) or (2) can be simply transformed as shown in Definition 2, while others associated using (3) need a pre-transformation step before applying the transformation of Definition 2, as they are normally linked through a third relation known as a linking table. The pre-transformation steps is described in the example below.

Definition 3. Fragmentation [5]
Let us consider a relational database D with relations $R_1, \ldots, R_n$ and A the list of all attributes contained in different relations. Given A_f the list of attributes to be fragmented, the result of the fragmentation is a list of fragments $F = \{F_1, \ldots, F_m\}$ where each of these fragments satisfies:

i $\forall F_i \in F, i \in [1..m],\ F_i \subseteq A_f$.
ii $\forall a \in A_f, \exists F_i \in F : a \in F_i$.
iii $\forall F_i, F_j \in F, i \neq j : F_i \cap F_j = \emptyset$.

Note that the list of attributes to be fragmented A_f contains all attributes in A, except those concerned with Singleton Constraints (attributes to be encrypted). Condition (i) guarantees that only attributes in A_f are concerned by the fragmentation, condition (ii) ensures that any attribute in A_f appears in clear at least in one fragment and condition (iii) guarantees unlinkability between different fragments.

Logically, to be able to get information about the original database, we should be able to reconstruct original database from fragments. So after defining the fragmentation process, we shall define a mechanism to combine fragmentation and encryption. More clearly, a mechanism to integrate attributes involved in the Singleton Constraints (attributes to be encrypted) in the suitable fragment. These encrypted attributes allow only authorized users (users who know the encryption key) to construct the sensitive relationships. Based on the definition of $Physical fragment$ proposed in [4], we define our mechanism called $Secure fragment$ to combine fragmentation and encryption.

Definition 4. Secure Fragment: Let D be a relational database with a list of relations $R = \{R_1(a_{1,1}, \ldots, a_{j,1}), \ldots, R_n(a_{1,n}, \ldots, a_{k,n})\}$, $F = \{F_1, \ldots, F_m\}$ a fragmentation of D and A_f be the list of fragmented attributes. Each fragment $F_i \in F$ is composed of a subset of attributes $A_i \subseteq A_f$. Each A_i is composed of a subset of attributes of one or more relations $R_j \in R$. We denote by R_{F_i} the list of relations in R which a subset of their attributes belongs to the fragment $F_i \in F$. The secure fragment of F_i is represented by a set of relations schema $R^e_{F_i}$ in which each relation is represented as follows $R^e_j(\underline{salt}, enc, a_1, \ldots, a_k)$ where $\{a_1, \ldots, a_k\} \subset A_i \cap R_j$ and enc is the encryption of all attributes of R_j that do not belong to $\{a_1, \ldots, a_k\}$ (all attributes of R_j involved in a singleton constraint except those concerned by a singleton constraint over the foreign key), combined before encryption in a binary XOR with the salt. All foreign key attributes which are involved in singleton constraints are encrypted using a deterministic encryption algorithm (e.g., AES) to ensure their indistinguishability. The Algorithm 1 shows the construction of secure fragments.The main reason for reporting all original attributes (except foreign keys involved in the Singleton constraints) in an encrypted form for each relation in a fragment, is to guarantee that a query Q over the original relation R_j can be executed by querying a single fragment (which contains R^e_j) while preserving confidentiality of sensitive relationships, so we do not need to reconstruct the original relation R_j to perform the query Q. Furthermore, encrypting foreign keys ensure the protection of sensitive relationships between relations involved into Inter-table constraints. However, using deterministic encryption algorithm has two issues. First, a major advantage is to enforce indistinguishability of records which allows only authorized users who know the encryption key to execute queries associating these relations. Second, a minor drawback is that it allows an adversary to infer information about repeatedly occurring values of the encrypted foreign keys, but this information

does not allow the adversary to break the unlinkability between relations. The attribute *salt* which is used as a primary key of different relations in the secure fragments protects encrypted data against frequential attacks. In addition, there is no need to secure the *salt* attribute because knowledge of the value of this attribute will not give any advantage when attacking encrypted data.

Example 4.2. Assume that we have a relational database D of a medical insurance company that contains two relations *Patient* and *Doctor* represented respectively in Table 1 and Table 2. The insurance company has defined the a set of confidentiality constraints $CC = \{C_1 = \{SSN\}, C_2 = \{Name\_pat, Illness\}, C_3 = \{Patient, Doctor\}\}$. As shown before, the first step in the fragmentation process consists in transforming Inter-table constraint (C_3). Relations *Patient* and *Doctor* are linked through the foreign key *Id_doc* in the relation *Patient*, therefore C_3 will be replaced by $C_4 = \{Id\_doc\}$ and all possible Association constraints composed of an attribute of the relation *Doctor* and an attribute of the relation *Patient* (Guarantee that the relation *Patient* will not be in the same fragment as the relation *Doctor*). In our example, attributes SSN and *Id_doc* of the relation *Patient* are involved in singleton constraints C_1 and C_4 respectively. So they will not be concerned by the fragmentation. As a result C_3 will be replaced by :

- $C_4 = \{Id\_doc\}$
- $C_5 = \{Name\_pat, Id\_doctor\}$
- $C_6 = \{Name\_pat, Name\_doc\}$
- $C_7 = \{Dob, Id\_doctor\}$
- $C_8 = \{Dob, Name\_doc\}$
- $C_9 = \{Illness, Id\_doctor\}$
- $C_{10} = \{Illness, Name\_doc\}$

Table 1. Patient relation

SSN	Name_pat	Dob	Illness	Id_doc
865746129	A. Barrett	20-08-1976	Illness_1	doc_3
591674603	C. Beat	18-01-1981	Illness_2	doc_3
880951264	N. Baines	14-09-1986	Illness_1	doc_2
357951648	S. Brandt	18-01-1981	Illness_3	doc_1

Table 2. Doctor relation

Id_doctor	Name_doc
doc_1	C. Amalia
doc_2	D. Annli
doc_3	P. Amadeus

A possible fragmentation of D that satisfies all confidentiality constraints is the set of fragments $\{F_1, F_2, F_3\}$ with: $F_1 = \{Patient(Name\_pat, Dob)\}$, $F_2 = \{Patient(Illness)\}$ and $F_3 = \{Doctor(Id\_doctor, Name\_doc)\}$. Next step is the $Secure fragmentation$ transformation (Definition 3). We assume that encryption of the protected attributes uses the deterministic encryption algorithm E with the encryption key K. The result of applying the $SecureFragmentation$ over different fragments is represented as follows.

- $F_1 : Patient(\underline{salt}, enc, Name\_pat, Dob, E_k(Id\_doc))$ with $enc = E_K(\langle SSN, Illness\rangle \bigoplus salt)$
- $F_2 : Patient(\underline{salt}, enc, Illness, E_k(Id\_doc))$ with $enc = E_K(\langle SSN, Name\_pat, Dob\rangle \bigoplus salt)$
- $F_3 : Doctor(Id\_doctor, Name\_doc)$

Note that F_3 has not been changed because there is no singleton constraints over the *Doctor* attributes. Lastly data fragments F_1, F_2 and F_3 are distributed to different Cloud storage providers.

5 Query Execution Model

Before discussing techniques for processing query over distributed fragments, we will first present the architecture of our proposed approach. It includes three principal entities : (1) a $User$ which attempts to execute queries over the original database, (2) a *Client* which rewrites user queries by splitting them to create an optimized distributed Query Execution Plan QEP; QEP is a set of sub-queries and other operations (e.g., decryption, join...), it is created by the $Query\ Transformative$ based on the $MetaData$ which contains information (relations, clear attributes, encrypted attributes, selectivity of attributes) about data distribution in different fragments. Furthermore, the *Query Executor* executes each of these sub-queries over the appropriate fragments and sends back the results to the client. (3) *Server* represented by different Cloud storage providers in which data fragments are distributed.

Query Transformation and Optimization. In our querying model, query transformation is performed by the *Query Transformative* (QT) entity on the client side. When receiving a user query, the query is analyzed syntactically and semantically so that incorrect queries are rejected as earlier as possible. Next, based on the $Metadata$ stored on the client side, the QT will attempt to find a fragment on which the user query can be executed, i.e. a fragment in which QT can find all attributes and relations involved in the user query. If such a fragment does not exist, QT will decompose the user query into queries expressed in relational algebra, find out which fragments are involved in the query, and finally transform the user query into a set of fragments queries. Using this set of fragment queries and other operations such as encryption, decryption, join and aggregation, the QT creates a QEP and sends it to the *Query Executor*. A query can have more that one QEP. Logically, each QEP may have a different execution cost. Thus, the QT should have the capability to pick out the best QEP in terms of execution cost. This capability is explained later in the Query Optimization section.

For multi-fragment query[2], QT will use local join operations as it should combine results of execution of subqueries over fragments. There are two different ways to perform local join operation : (1) Execute all sub-queries in a parallel manner, then join the result on the client side. (2) Execute sub-queries in a sequential manner to have the ability to perform *semi-joins* using the result of previous sub-queries. While (1) can be cheaper than (2) in terms of sub-query execution, it is much more costly in the join operation because in (1), sub-queries results might contain a lot of records that will not be part of the final results.

[2] i.e. a query that cannot be executed over only one fragment.

In addition to traditional query optimization methods such as selecting conditions as earlier as possible, the QT attempts to minimize the execution cost of the created QEP by applying the selection condition with the most selective attribute, i.e the selection condition which is satisfied by the smallest number of tuples. To give this ability to the QT, we assign a selectivity[3] to each attribute contained in the original database to the $Metadata$ stored in the $Client$. We apply the optimization method to the example below.

6 Preserving Data Unlinkability

Ensuring data confidentiality is achieved by preserving unlinkability between different data fragments and by encrypting all sensitive information that cannot be protected using only fragmentation. However, we have seen in the previous section that evaluation of some queries may use *semi join* in order to join data from different fragments. This will not be a concern in the case of non-colluding Cloud storage providers, but it becomes a serious concern when Cloud Storage Providers (CSP) can collude. In this section, we present our solution to overcome this privacy concern when we assume that CSP can collude to link data stored in different fragments.

To overcome this problem, the *Client* should have the ability to execute *semi join* queries and retrieve data from a fragment without the CSP (which stores the fragment) learning any information about the *semi join* condition values. To meet this requirement, we will use a Private Information Retrieval keyword-based technique. PIR keyword-based was presented in [2] to retrieve data with PIR using keywords search over many data structures such as binary trees and perfect hashing. Later, [12] investigated the use of SQL for PIR, based on the use of B+ tree. In the next part of this paper, we will explain how we can use technique presented in [12] to ensure our *semi join* queries privacy requirement.

Theorem 2. Let $\mathcal{D}$ be a multi-relation normalized database, $\mathcal{F} = \{F_1, F_2\}$ be a fragmentation of $\mathcal{D}$, and Q be a multi-fragment query that joins records from both fragments F_1 and F_2. Consider that SCPs in which the fragments F_1 and F_2 are stored can collude to link data stored in these fragments, and that Q is evaluated using *semi join* operations. Sensitive relationships between F_1 records and F_2 records remain protected if and only if the privacy of the *semi join* subqueries is guaranteed.

PIR System Design. In the *Client* of our architecture, we give to *Query Executor* the ability to communicate with different Cloud storage providers through the PIR keyword-based protocol. In the *Server*, we add on each CSP a *PIR Server* as a front-end entity to answer *Query Executor*'s PIR queries. An adversary (a Cloud storage provider administrator) who can observe *Query Executor*'s PIR-encoded queries is unable to find out the clear content of the queries. Enforcing integrity on the *PIR server* side is straightforward since we

[3] Provides an approximation of the number of tuples that satisfies a predicate.

assume that PIR servers will not attempt to wrongly answer *Query Executor*'s PIR queries.

Keyword Index Structures. The main purpose for using PIR keyword-based is to ensure the privacy of *semi join* queries. In our approach, this kind of queries is mainly executed over primary or foreign key attributes. Therefore each *PIR Server* will create a B+ Tree over each indexed attribute (Primary or foreign key). The advantage of such an index structure is that data appears only in the leaves while other nodes are used to control the search. On the leaves of B+ tree and for each key in the tree, we store the tuple corresponding to the key as the data part linked to the key. We consider the act of retrieving a node's data as a PIR operation over all nodes in the B+ tree. In all existing PIR schemes, a common assumption is that the client should know the address of the block or the item to be retrieved. To satisfy this assumption in our approach, the *PIR server* after creating the index structure, sends an index helper containing the B+ tree's root node to the client.

Semi-Join PIR Keyword-based Query. Using the PIR keyword-based query requires a setup phase in which the *Query Executor* and the *PIR server* exchange information. This setup phase is divided into two steps:

1. The *Query Executor* sends the Relation schema name and the attribute name over which the *semi join* query is to be performed to the corresponding *PIR server*.
2. When receiving the Relation schema name and the attribute name, the *PIR server* selects the corresponding B+ tree and sends its root node to the *Query Executor*.

After receiving the root node sent by the *PIR server*, the *Query Executor* will compare the list of keys contained in the root node with values used in the condition of the *Semi join* query in order to find out the indexes of the next nodes to be retrieved. The *Query Executor* will subsequently perform PIR queries over chosen indexes to retrieve corresponding nodes. Once all items have been retrieved, The *Query Executor* combines them to build the result of the original *Semi join* query. Refer to Appendix D for a description of the PIR keyword-based protocol algorithms used in the *Client* and the *Server* parts.

7 Conclusion

Existing approaches based on fragmentation and encryption have focused on single-relation schema database which is a strong and rarely encountered assumption in real environment. In this paper, we have presented a new approach based on fragmentation, encryption and query privacy techniques which ensures confidentiality of outsourced multi-relation databases. We have presented also a way to optimize and execute queries over distributed data fragments.

Our future work will include the implementation of our approach. It will also include enhanced query optimization and execution techniques to overcome some limitations of our approach, such as processing nested queries.

References

1. Bellare, M., Fischlin, M., Ristenpart, T.: Deterministic encryption: Definitional equivalences and constructions without random oracles (2008)
2. Benny Chor, N.G., Naor, M.: Private information retrieval by keywords. Cryptology ePrint Archive, Report 1998/003 (1998)
3. Biskup, J., Preuß, M., Wiese, L.: On the Inference-Proofness of Database Fragmentation Satisfying Confidentiality Constraints. In: Lai, X., Zhou, J., Li, H. (eds.) ISC 2011. LNCS, vol. 7001, pp. 246–261. Springer, Heidelberg (2011)
4. Ciriani, V., De Capitani di Vimercati, S., Foresti, S., Jajodia, S., Paraboschi, S., Samarati, P.: Fragmentation and Encryption to Enforce Privacy in Data Storage. In: Biskup, J., López, J. (eds.) ESORICS 2007. LNCS, vol. 4734, pp. 171–186. Springer, Heidelberg (2007)
5. Ciriani, V., De Capitani di Vimercati, S., Foresti, S., Jajodia, S., Paraboschi, S., Samarati, P.: Fragmentation design for efficient query execution over sensitive distributed databases. In: ICDCS, pp. 32–39. IEEE Computer Society (2009)
6. Ciriani, V., De Capitani di Vimercati, S., Foresti, S., Livraga, G., Samarati, P.: Enforcing Confidentiality and Data Visibility Constraints: An OBDD Approach. In: Li, Y. (ed.) DBSec. LNCS, vol. 6818, pp. 44–59. Springer, Heidelberg (2011)
7. Hacigümüs, H., Iyer, B.R., Li, C., Mehrotra, S.: Executing sql over encrypted data in the database-service-provider model. In: SIGMOD Conference, pp. 216–227. ACM (2002)
8. Hacigümüs, H., Mehrotra, S., Iyer, B.R.: Providing database as a service. In: ICDE, pp. 29–38. IEEE Computer Society (2002)
9. Hudic, A., Islam, S., Kieseberg, P., Weippl, E.R.: Data confidentiality using fragmentation in cloud computing. International Journal of Communication Networks and Distributed Systems, IJCNDS (2012)
10. Ponemon Institute. Fourth annual us cost of data breach study (January 2009)
11. Ceri, S., Wiederhold, G., Navathe, S.B., Dou, J.: Vertical partitioning of algorithms for database design. ACM Trans. Database Syst. 9(4), 680–710. 98, 99, 102, 109, 125 (1984)
12. Olumofin, F., Goldberg, I.: Privacy-Preserving Queries over Relational Databases. In: Atallah, M.J., Hopper, N.J. (eds.) PETS 2010. LNCS, vol. 6205, pp. 75–92. Springer, Heidelberg (2010)

A Secure Fragmentation Algorithm

Algorithm 1. Secure fragmentation

Require:
$\mathcal{D} = \{R_1, R_2, \cdots, R_n\}$ /* Normalized relational database */
$\mathcal{C} = \{C_1, C_2, \cdots, C_m\}$ /* Confidentiality constraints */
Ensure:
$\mathcal{F}^s = \{F_1^s, F_2^s, \cdots, F_p^s\}$ /*The set of secure fragments*/
Main
$\mathcal{C}_f = \{C_i \in \mathcal{C} : |C_i| > 1\}$ /* The list of association constraints */

$\mathcal{A}_{fkey} = \{a \in C_i, C_i \in \mathcal{C} : |C_i| = 1 \; and \; \textbf{isForeignKey}(a) = True\}$
/* $\mathcal{A}_{fkey}$: The set of foreign keys to be encrypted*/
$\mathcal{F} := \textbf{Fragment}(\mathcal{D}, \mathcal{C}_f)$
for all $F_i = \{a_{i_1}, a_{i_2}, \cdots, a_{i_n}\}$ in $\mathcal{F}$ **do**
  $\mathcal{R}_f = \textbf{classifyAttributes}(F_i)$ /* Classify attributes*/
  for all R_{f_i} in $\mathcal{R}_f$ **do**
    for all r in R_{f_i} **do**
      $r^s[salt] := \textbf{GenerateSalt}(R_{f_i}, r)$ /* r : record */
      $r^s[enc] := \mathcal{E}_k(t[a_{j_1}, \cdots, a_{j_q}] \oplus r^s[salt])$ /* $a_{j_1}, \cdots, a_{j_q} = R_i - R_{f_i}$ */
      for all a in R_{f_i} **do**
        $r^s[a] := r[a]$ /* a : attribute */
      end for
      for all a in A_{fkey} **do**
        if $a \in R_i$ **then**
          $r^s[a] := \mathcal{E}_k(r[a])$ /* a : the foreign key of the relation R_i */
        end if
      end for
      InsertRecord(r^s, R^s)
    end for
    AddRelationToFragment(R^s, F^s)
  end for
end for

B Proof of Theorem 1

Proof. According to Item (ii) of Definition 2, the Inter-table constraint will be replaced by all possible associations constraint composed from an attribute of relation R_1 and another from relation R_2. Due to the fact that an association constraint between two attributes means that the relationship between these attributes will be protected using fragmentation (each attribute will be stored in different fragments), Item (ii) guarantees that relations R_1 and R_2 will be stored in different fragments which hold condition (2).

Item (i) of Definition 2 creates a singleton constraint over the foreign key b_m of the relation R_2. Thus b_m will be considered as a sensitive attribute and will be protected using encryption, which means that the foreign key b_m will not appear in clear in any fragment. As a result, if an adversary succeeds in having access to the fragments in which R_1 and R_2 have been stored, she is unable to link data stored in these relations.

C Proof of Theorem 2

Proof. To prove the Theorem 2, we will use the following two sketches. The fist sketch proves that without ensuring *semi join* sub-queries privacy, collaborative CSPs can, in some cases, break data unlinkability, while the second sketch proves that, under a collaborative Cloud storage service providers model,

protecting data unlinkability can only be guaranteed with the protection of the privacy of the *semi join* sub-queries.

SKETCH Without Using the PIR Keyword-based Protocol. Suppose that the *Cient* wants to execute a query which joins records from two fragments F_1 and F_2. Let us consider that the sub-query Q_1 executed over the fragment F_1 has returned n tuples. And the semi-join query Q_2 executed over F_2 has returned m tuples. Therefore, if CSPs that store F_1 and F_2 collude together to link tuples from Q_1 and Q_2 results, the probability to guess correctly the relationship between tuples is:

$$PROB[Result(Q_1) \leftrightarrow Result(Q_2)] = \frac{1}{m \times n}$$

Clearly, if m and n are small, CSPs will have a great chance to break data unlinkability.

SKETCH Using the PIR Keyword-based Protocol. Let us consider that the *Client* attempts to perform a query which joins records from two fragments F_1 and F_2. According to our defined PIR keyword-based protocol, the *Client* will execute Q_1 over the fragment F_1 without using the keyword-based protocol. Next, the *Client* will send the table name T and the attribute name a on which the semi-join will be performed, the *Server* replies with the root node of the corresponding B+ tree. It is clear from the previous step that the CSP which stores F_2 can only know the attribute name and the table name on which the semi-join will be performed. After receiving the root node, the *Client* will use the PIR protocol to retrieve internal corresponding nodes until the leaves of the B+ tree are reached. The PIR protocol will ensure that the server will not know which nodes were retrieved by the *Client*. Moreover, all tuples are stored in the leaf level of the B+ tree. Therefore, in order to retrieve each record, the *Client* shall execute the same number of PIR queries. Rightfully, the only revealed information when using the PIR keyword-based protocol is the table name and the attribute name on which the semi-join has been performed. Therefore, if CSPs storing F_1 and F_2 collude together to break data unlinkability, they will be able only to infer that the relation T_1 in F_1 over which Q_1 has been executed is linked to the relation T through the attribute a. Due to the fact that the foreign key in T_1 referencing the attribute a in T is encrypted, linking records is not possible.

D SemiJoin PIR Keywordbased Query Algorithms

Algorithm 2. SemiJoin PIR keywordbased query (server)

Require:
$BPT = \{B_1, \ldots, B_n\}$ /* B-Plus Tree over indexed attributes*/
loop
$Request \leftarrow handle\_client\_request()$
if $Request$ is PQR **then**
/* PQR : Pre-Query Request */
$(TabName,\ AttriName) \leftarrow Request$
$B \leftarrow GetAssociatedBPT(TabName,\ AttriName)$
$Root_B \leftarrow GetRootNode(B)$
$ReplyToClient(Root_B)$
else {$Request$ is PIRQ}
/* PIRQ : PIR Query */
$result \leftarrow compute(Request)$
$ReplyToClient(result)$
end if
end loop

Algorithm 3. SemiJoin PIR keywordbased query (client)

Require:
$tabName, attrName$ /* Table and Attribute where the semi-join will be performed*/
$value$ /* Semi-join condition value*/
$Node \leftarrow send\_PQR\_request(tabName, attrName)$
repeat
for all $elem$ in $Node$ **do**
$findLink \leftarrow$ **false**
if $Key(elem) < value$ **then**
$Node \leftarrow PIR\_Query(IndexOfLeftChild(elem))$
$findLink \leftarrow$ **true**
break /* terminates the for loop*/
end if
end for
if $findLink =$**false then**
$Node \leftarrow PIR\_Query(IndexOfRightChild(elem))$
end if
until $Node$ is leaf_node
for all $elem$ in $Node$ **do**
if $Key(elem) = value$ **then**
return $Data(elem)$
end if
end for