Preserving Multi-relational Outsourced Databases Confidentiality using Fragmentation and Encryption * † ‡

Anis Bkakria§, Frédéric Cuppens, Nora Cuppens-Boulahia
Télécom Bretagne Rennes, France
{anis.bkakria, frederic.cuppens, nora.cuppens}@telecom-bretagne.eu

José M. Fernandez
École Polytechnique de Montréal, Canada
jose.fernandez@polymtl.ca

David Gross-Amblard
Université de Rennes 1 - IRISA, France
david.gross-amblard@irisa.fr

Abstract

Confidentiality and privacy of outsourced data has become one of the pressing challenges in Cloud computing. Outsourced data often includes sensitive personally identifiable information. When data is outsourced, sensitive information will not be under the control of its owners, but under the control of an external service provider. In this paper, we define an approach allowing the protection of confidentiality of sensitive information in outsourced multi-relational databases by improving an existing approach based on a combination of fragmentation and encryption. Then we define a secure and effective technique for querying data hosted on several service providers. Finally, we improve the security of the querying technique in order to protect data confidentiality under a collaborative Cloud storage service providers model.

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

1 Introduction

Management of large amount of information containing sensitive elements is often expensive. Database outsourcing in which organizations outsource the storage and management of their collected data to third party service providers offers a way for these organizations to reduce the cost of controlling complex information structures.

Database as a service models give rise to two significant challenges. The first challenge is how service providers protect outsourced databases from not authorized users. A straightforward solution to protect outsourced databases consists in encrypting data before their outsourcing. Unfortunately, Querying data becomes in this case expensive (heavy computational overheads) and can be impossible for several kind of queries. The second challenge is more complex as it concerns the protection of outsourced databases from the service providers as in this case service providers are not considered to be fully trusted. Therefore, the main focus of this paper is to preserve outsourced data privacy and confidentiality while providing a secure and efficient querying technique.

Journal of Wireless Mobile Networks, Ubiquitous Computing, and Dependable Applications, volume: 4, number: 2, pp. 39-62

*This paper is an extended version of the work originally presented at the 2013 Asian Conference on Availability, Reliability and Security (AsiaARES'13), Gadjah Mada University, Indonesia, March 2013 [1].

†This work has received a French government support granted to the CominLabs excellence laboratory and managed by the National Research Agency in the "Investing for the Future" program under reference ANR-10-LABX-07-01.

‡This work has received a French government support granted to the Frag&Tag project and managed by the Dual Innovation Support Scheme (RAPID) under CONVENTION N° 132906023.

§Corresponding author: 3 rue de la chataigneraie, 35510 Cesson Sevigne, France, Tel: +33665332667

1.1 Related Work

Thanks to advances in the development of fast encryption algorithms, data encryption has been considered as the greatest way for protecting outsourced database. One approach to protect confidentiality and privacy of the outsourced data is based on encrypting all tuples in outsourced databases [2, 3]. Hacigümüs et al. [2] have proposed the first approach aiming at query encrypted data. The proposed technique is based on the definition of a number of buckets on the attribute domain which allow the server-side evaluation of point queries. Hore et al. [4] improve the bucket-based index methods by presenting an efficient method for partitioning the domain of attributes. The authors present a way for building an efficient index, which aims to minimize the number of specious tuples in the result of range and equality queries. There are also some researches [5, 6] proposing an indexing method specific to string attributes. The proposed method improves the hash-based indexing techniques to allow the evaluation of *LIKE* condition directly over encrypted data.

The main drawback of bucket-based and hash-based indexing methods resides on the fact that they expose data to inference attacks [4, 7, 8, 9]. An interesting technique to perform range query is based on Order Preserving Encryption (OPE) schemes [10, 11, 12]. OPE schemes are symmetric-key deterministic encryption schemes which produce cipher-texts that preserve the order of the plain-texts. Xiao et al. [13] show that OPE ensures data secrecy only if the intruder does not know the domain of original attributes or the plain-text database.

One crucial question when using encryption-based approaches is how to efficiently execute queries. Obviously, The use of deterministic encryption techniques permits to evaluate efficiently equality-match queries. It is however more difficult to perform aggregations and range queries as we have to decrypt all records to evaluate this kind of queries. As a result, query execution on the outsourced encrypted data is much more difficult. Thus, in this paper we focus on protecting both confidentiality and privacy of outsourced database while defining a secure technique for querying outsourced data. One interesting approach to achieve this requirement is based on the use of data fragmentation. Basically, data fragmentation aims to improve data manipulation process, optimize storage, and facilitate data distribution. It is not particularly designed for preserving data security. Nevertheless, two significant alternatives have been proposed in the last few years. The first alternative [14, 15] relies only on data fragmentation to protect confidentiality. This approach uses a distribution model composed mainly of two domains: a honest but curious domain in which the data will be hosted, and a trusted local domain from which the data originates. The local domain is used to store fragments that contain highly sensitive data without the need to encrypt them, as it is considered trustworthy. This approach is not so efficient because it forces data owners to always manage and protect fragments containing highly sensitive data. Aggarwal et al. [16] have proposed an approach allowing to protect outsourced data by encrypting only sensitive attributes and splitting sensitive association among several non-communicating servers. This approach has a major drawback in that it assumes that servers are non-communicating.

A more interesting approach [17, 18, 19] is based on combining encryption together with data fragmentation. The main idea in this alternative is to fragment 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). However, these approaches have a major limitation as they assume that data to be outsourced is represented within a single relation schema (or table). 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. Therefore, assuming that data to be outsourced is represented within a single relation schema is too strong and seldom satisfied in real environments.

1.2 Intended Contributions

We aim in this paper to protect the privacy and the confidentiality of sensitive outsourced databases by combining the best features of fragmentation and encryption. Furthermore, we present an approach which is able to deal efficiently with multi-relation normalized databases with which we strive to overcome the previously mentioned limitations of [18, 19]. The problems encountered in one-relation[1] databases take on additional complexity when working with multi-relation normalized databases in a distributed environment, as it gives rise to new problems such as protecting the relationships between relational schemas (relationships between tuples in distinct tables) and defining a secure and efficient technique allowing authorized users to query these sensitive relationships. We will present our approach which uses a practical Private Information Retrieval (PIR) technique allowing to protect data unlinkability of different fragments of the original database by protecting user query privacy. 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 vertically into two or more fragments. Unlinkability of fragments means that despite the fact that an adversary has knowledge about the fragments of a relation, he/she remains unable to link records from different fragments. Furthermore, we evaluate our previously proposed protocol [1] by presenting some experiments using our developed prototype. Afterwards, we use hash table data structures to store the information of each fragments instead of using B+ trees which allows us to improve the effectiveness of the proposed PIR keyword-based technique [1].

1.3 Paper organization

We proceed by describing in Section 2 the problem and the need for an approach like ours through an example. In Section 3, we detail our model architecture, the threat model, security model and assumptions. After that, we describe in Section 4 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. In Section 7, we present the prototype developement and experimentations we conducted. Finally, we conclude the paper in Section 8.

2 Motivating Scenario

In our working scenario, we strive to protect the confidentiality of an outsourced relational hospital database $\mathscr{D}$ composed of two relations (primary keys are underlined and foreign keys indicated by *) :

Patient(Id_P, Name,ZIP, Illness, Id_Doctor *)
Doctor(Id_D, Name, Specialty)

The relationship between the tables *Patient* and *Doctor* is defined between the foreign key of the table *Patient* (*Id_Doctor*) and the primary key of the table *Doctor* (*Id_D*). We assume that the database will be outsourced to a third party. Therefore, sensitive information stored in $\mathscr{D}$ should be protected. One classic solution is to encrypt all information before outsourcing the database, a costly operation. However, if we look carefully, the list of patients and their attributes $(Id\_P, Name, Zip)$ can be considered as insensitive, and also that the list of illnesses could be made public. Nevertheless, data sensitivity arises from the relationship between these two lists (list of patients and list of illnesses). Therefore if we can

[1] Databases composed from a single relation schema.

find a way (e.g. vertical fragmentation [20]) 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 list of doctors and the list of patients are not sensitive. However, the relationship between a patient and his doctor should be protected. The good way to protect the relationship between the two relations *Patient* and *Doctor* consists in encrypting the foreign key *Id_Doctor* or the primary key *Id_D*. The encrypting of the foreign key appears to be more beneficial as a foreign key references only one relation (only the relationship between the two relations is 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 Technical Preliminaries

3.1 Architecture

We consider our architecture of storage and query over distributed fragments illustrated in Figure 1. It is composed of three main components:

- **Users**: They are actually database clients who have permission to query outsourced data. To the *Users*, all operations which will be used in our approach (e.g., fragmentation and encryption) in order to protect sensitive data confidentiality are transparent. That is, *Users* believe that they interact the original database and form their queries against it.
- **Client**: It is a trusted party which transform *Users* 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...). Based on the *Metadata* containing information (e.g., relations, clear attributes, encrypted attributes and selectivity [2] of attributes) about data distribution in different fragments, the *Query Transformation module* construct a QEP which will be executed.
- **Server**: It represents different Cloud Storage Providers in which data fragments are distributed.

3.2 Trust and Attack Model

Cloud servers are considered to be the best options for small companies with limited IT budget allowing to reduce the cost of maintaining computing infrastructure and data-rich applications. However, most of related works [14, 17, 2, 15] on the confidentiality and privacy of outsourced data considered that Cloud service providers are ”honest-but-curious”. The *semi-honest* model is the right fit for our approach, as in this model, the Cloud servers act in an “honest” manner by correctly responding user queries and following the designated protocol specification. In this paper, we consider that Cloud services providers have two levels of curiosity: (1) 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. (2) In the second part of the paper, we will 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 using existing protocols e.g., SSL.

[2] Attribute selectivity is is an estimated number that determines the effectiveness of queries that performs a search over this attribute.

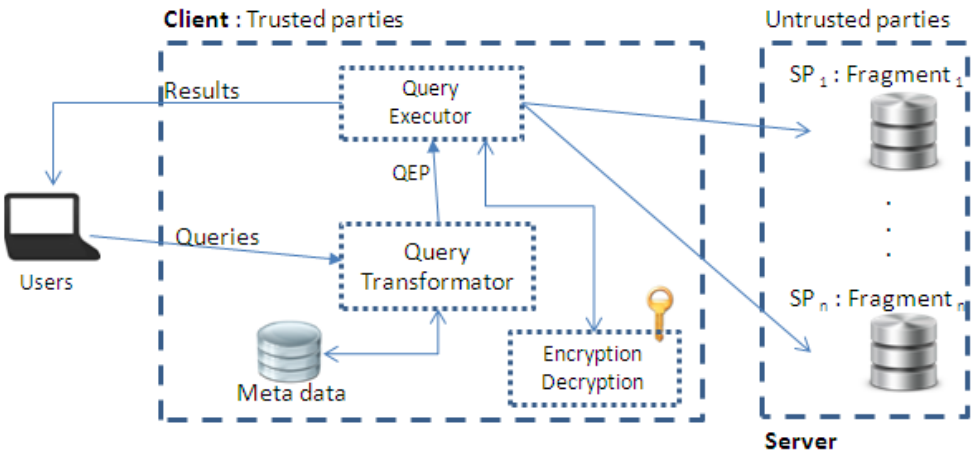

Figure 1: Architecture of the Proposed Model

4 Confidentiality using fragmentation and encryption

Our approach extends in several ways the vertical fragmentation-based approach described in [17, 19]. 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 rule for any typical 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 confidentiality by encryption and fragmentation, and second formalizes the concept of fragmentation in our approach which extends ideas presented in [17, 18, 19].

Definition 1 (Confidentiality Constraint). Consider that data to be secured are represented with a relational database $\mathscr{D}$, 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 $\mathscr{D}$ 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 $\mathscr{D}$. Relations R_i and R_j must 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 on distinct servers of distinct providers. 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 database interrogation

more complicated. A more adapted way to resolve this kind of confidentiality constraint was proposed in [17], 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 involved in that case 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 [19], that efficiently computes data fragments satisfying 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 definitions and theorems 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 enforcement of C implies 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 $\mathscr{D}$ made up of 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 $\mathscr{F} = \{F_1, \ldots F_q\}$ be a fragmentation of $\mathscr{D}$ 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 $\mathscr{F}$.
2. $\forall\, AC_{i,j} = \{a_i, b_j\} \in C, i \in [1,n], j \in [1,m],\ \ if\ \ a_i \in F_k\ \ and\ \ b_j \in F_l\ \ \ then\ k \neq l$

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. □

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 [21] 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.

Example 4.1 : Consider that we have a hospital relational database $\mathscr{D}$ with relations :

Patient($\underline{Id\_patient}, Name, ZIP$)
Doctor($\underline{Id\_doctor}, Name, Specialty$)
Examination($\underline{Id\_examination}, date, medical\_report, Id\_doctor*, Id\_patient*$)

Assume that database owner claims that relationships between a patient and his/their doctor(s) are sensitive and must be secured. Therefore an Inter-table Constraint over relation *Patient* and *Doctor* ($IC = \{Patient, Doctor\}$) must be defined. In this case applying directly transformation as shown in Definition 2 is not possible since relations *Patient* and *Doctor* are connected through *Examination*. So, the pre-transformation step consists in writing the Inter-table Constraint IC using the linking relation *Examination*. Thus, IC will be replaced by $IC_1 = \{Patient, Examination\}$ and $IC_2 = \{Doctor, Examination\}$. Next, both IC_1 and IC_2 will be transformed into a set of Singleton Constraints and Association Constraints according to Definition 2.

Definition 3 : Fragmentation [18]
Let us consider a relational database $\mathscr{D}$ with relations $R_1, \ldots, R_n$ and A the list of all attributes contained in these 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 precisely, we need 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 [17], we define our mechanism called *Secure fragment* to combine fragmentation and encryption.

Definition 4 (Secure Fragment) . Let $\mathscr{D}$ 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 a new relation whose attributes are a subset $A_i \subseteq A_f$. 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 such that 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.

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 $\mathscr{D}$ 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 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 :

```
input :
        𝒟 = {R_1, R_2, ··· , R_n} /* Normalized relational database */
        𝒞 = {C_1, C_2, ··· , C_m} /* Confidentiality constraints */
output:
        ℱ^s = {F_1^s, F_2^s, ··· , F_p^s} /*The set of secure fragments*/

𝒞_f = {C_i ∈ 𝒞 : |C_i| > 1} /* The list of association constraints */
𝒜_fkey = {a ∈ C_i, C_i ∈ 𝒞 : |C_i| = 1 and isForeignKey(a) = True}
/* 𝒜_fkey : The set of foreign keys to be encrypted*/

/* Fragmentation */
ℱ := Fragment(𝒟, 𝒞_f)
foreach F_i = {a_i1, a_i2, ··· , a_in} in ℱ do
    ℛ_f = classifyAttributes(F_i) /* Classify the attributes according to their original relation.*/
    foreach R_fi in ℛ_f do
        foreach r in R_fi do
            /* r : record */
            r^s[salt] := GenerateSalt(R_fi, r)
            r^s[enc] := ℰ_k(t[a_j1, ··· , a_jq] ⊕ r^s[salt])
            /* a_j1, ··· , a_jq = R_i − R_fi */
            foreach a in R_fi do
                /* a : attribute */
                r^s[a] := r[a]
            endfch
            foreach a in A_fkey do
                if a ∈ R_i then
                    /* a : the foreign key of the relation R_i */
                    r^s[a] := ℰ_k(r[a])
                end
            endfch
            InsertRecord(r^s, R^s)
        endfch
        AddRelationToFragment(R^s, F^s)
    endfch
endfch
```

Algorithm 1: Secure fragmentation

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

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

A possible fragmentation of $\mathscr{D}$ 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 *Securefragmentation* 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.

Note that F_3 has not been changed because there is no singleton constraints over the *Doctor* attributes.

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

Figure 2: Secure Fragmentation Results

Lastly data fragments F_1, F_2 and F_3 are distributed to different Cloud storage providers.

5 Query Transformation and Optimization

In our querying model, query transformation is performed by the *Query Transformation* (QT) module 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*. The algorithm 2 shows the query validation, transformation and optimization process.

Example 5.1 (One-fragment query): Assume that we have a relational database $\mathscr{D}$ that contains two relations *Patient* and *Doctor* represented respectively in Table 1 and Table 2, The fragmentation of $\mathscr{D}$ is the list of fragments represented in Figure 2. Consider the following user query:

```
Q1 :  SELECT Name_pat, SSN
      FROM patient
      WHERE Dob='1986-09-14'
      And Illness = 'Illness1';
```

$Q1$ can be executed over either F_1 or F_2 fragments as all attributes required by $Q1$ can be found (in clear or encrypted form) in both fragments.
• QEP$_1$ for $Q1$ over F_1 :

```
Q1_1 :  SELECT Name_pat, salt, enc
        FROM patient
        WHERE Dob='1986-09-14';
Dec :   Decrypt(Result(Q1_1), Key) = R^d(Q1_1)
Q1_2 :  SELECT Name_pat, SSN
        FROM R^d(Q1_1)
        WHERE Illness = 'Illness1';
```

• QEP$_2$ for $Q1$ over F_2 :

```
input :
        Q     /* User Query */
        M     /* Metadata */
output:
        QEP   /*Query execution plan*/

(tables, attributes, conditions) = decomposeQuery(Q)
syntacticChecking(Q)  /* Verify that keywords, object names, operators, delimiters are placed correctly in the query*/
/* Semantic Checking */
if tables ⊈ M or attributes ⊈ M then
    rejectQeury(Q)
end
foreach (attribute, operator, value) in conditions do
    if !isTheSameType(attribute, value) or !match(operator, value) then
        rejectQeury(Q)
    end
endfch
/* Checking if there is a fragment on which the query can be directly executed */
𝔉 = getFragmentSchema(M)
foreach frag in 𝔉 do
    T_frag = getTables(frag)
    A_frag = getAttributes(frag)
    A_c = getAttributesFromConditions(Conditions)
    if tables ⊈ T_frag or attributes ⊈ A_frag or A_c ⊈ A_frag then
        continue
    end
    /* Checking if all conditions attributes are not encrypted in the fragment frag*/
    if areEncrpted(A_c, frag) then
        continue
    end
    /*The query Q can be executed on the fragment frag */
    addOperation(QEP, (Q,frag))
    foreach attr in A_frag do
        if isEncrpted(attr, frag) then
            addOperation(QEP, (Decryption,attr))
        end
    endfch
    return QEP
endfch


/* Multi Fragment Query*/
/* Get the fragments in which conditions attributes are not encrypted */
A_c = getAttributesFromConditions(Conditions)
A_c^s = sortAttributeBySelectivity(M, A_c) /* Sort attributes according to their selectivity*/
foreach attr in A_c^s do
    F_attr = containsInClear(M, attr) /* Set of fragments on which attr appears in clear text*/
    frag = getBestFragment(F_attr) /* Get the best fragment which contains the less number of encrypted attributes */
    A = listOfRetrievedAttributes(frag, attributes) /* The list of attributes that can be retrieved by querying the fragment frag */
    SQ = formulateTheSubQuery(A, attr, Q)
    addOperation(QEP, (Q, frag))
    foreach a in A do
        if isEncrpted(a, frag) then
            addOperation(QEP, (Decryption, a))
        end
    endfch
endfch
/*Add the join operation that combines results returned from subqueries*/
addOperation(QEP, join)
```

Algorithm 2: Query validation and transformation process

$Q1_1$: SELECT salt, enc
FROM patient
WHERE Illness = 'Illness1';
Dec : $Decrypt(Result(Q1_1), Key) = R^d(Q1_1)$
$Q1_2$: SELECT Name_pat, SSN
FROM $R^d(Q1_1)$
WHERE Dob='1986-09-14';

As we can see, 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[3], 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* [22] 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.

Example 5.2 (Multi-fragment query): Assume that we will use the same database $\mathscr{D}$ and fragments used in Example 5.1. Consider the query :

Q2 : SELECT Name_pat, Name_doc
FROM patient, doctor
WHERE Dob='1986-09-14'

A possible QEP for $Q2$ can be :

$Q2_1(F_1)$: SELECT Name_pat, $E_k(Id\_doc)$
FROM patient
WHERE Dob = '1986-09-14';
Dec : $Decrypt(E_k(Id\_doc), Key) = \delta$
$Q2_2(F_3)$: SELECT $Name\_doc$
FROM doctor
WHERE Id_doctor IN δ;
Join : $Result(Q2_1) \bowtie Result(Q2_2)$

Since the relationship between the two relations *Patient* and *Doctor* is protected, these relations are stored in different fragments. Therefore, the query $Q2$ is decomposed into two sub-queries $Q2_1$ and $Q2_2$ executed respectively over fragments F_1 and F_3.

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 and an average attribute-value size (AVS) to each attribute contained in the original database to the $Metadata$ stored in the $Client$. The selectivity of an attribute is the ratio of the number of distinct values to the total number of rows.

$$Selectivity = \frac{DistinctValues}{TotalNumberRows} \quad (1)$$

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

In distributed databases, they may exist several strategies for each query due to the fact that data are stored in different sites. One way to choose the best strategy is based on calculating the expected cost which should include corresponding evaluation and communication cost. The formula of a point query execution costs can be estimated roughly as follows:

$$Query\ execution\ cost = C_E \times NbExRow + C_T \times NbEstRow \tag{2}$$

C_E represents the evaluation cost of a record, C_T is for the transmission cost of a record, $NbExRow$ is the number of rows examined and $NbEstRow$ is for the estimated number of returned rows which is calculated as follows :

$$NbEstRow = \frac{1}{Selectivity} \tag{3}$$

Using the average attribute-value size (AVS) of encrypted attribute *enc*, we can estimate the execution costs of the decryption operation as follows :

$$Decryption\ cost = C_D \times AVS \times Number\ of\ rows \tag{4}$$

C_D represents an estimation of the per-byte decryption costs of the used encryption schemes.

Example 5.3 : Assume that we use the same database $\mathscr{D}$ and fragments of Example 5.1. Let us suppose that the relation patient contains 10^5 tuples and the selectivity estimation of the attribute *Dob* is 0.14 and for *Illness* it is 8×10^{-4}. We suppose also that $AVS_1 = 252$ and $AVS_2 = 152$ are respectively the average attribute-value sizes of encrypted attribute *enc* of the table *patient* stored in the fragments F_1 and F_2. Consider the query $Q1$ used in the Example 5.1. As shown before, there are two possible QEP for this query. Using (2), (3) and (4) the QT will compute the approximative execution cost for each QEP as shown below :

$$QEP_1\ execution\ cost = C_E \times 10^5 + C_T \times 7 + C_D \times 252 \times 7 + C_E \times 7$$
$$QEP_2\ execution\ cost = C_E \times 10^5 + C_T \times 1250 + C_D \times 152 \times 1250 + C_E \times 1250$$

After computing the approximative execution cost of each QEP, the QT will select the best one in terms of execution cost. In our example, QEP_1 has the lowest execution cost.

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 security and privacy problem 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.

Example 6.1: Consider the database, fragments, queries and QEP used in the Example 5.2. The QEP is executed in a sequential manner by the *QueryExecutor*. The next table shows the execution of the QEP.

Operation	Result
$Q2_1$ execution over F_1 :	$\langle N.Baines, E_k(doc\_2)\rangle$
Decryption :	$\delta = \{Decryption(E_k(doc\_2))\}$
$Q2_2$ execution over F_3 :	$\langle D.Annli\rangle$

Assume that F_1 and F_3 are distributed respectively in CSP_1 and CSP_3 which will try together to link tuples stored in the two fragments by correlating the history of user queries, their execution time and their respective responses. In our example, CSP_1 will disclose that a client has executed $Q2_1$ to retrieve the tuple $\langle N.Baines, E_k(doc\_2)\rangle$ at the time t, while CSP_3 will disclose that the same client has executed $Q2_2$ to retrieve $\langle D.Annli\rangle$ at the time $t+n$. Using this information, CSP_1 might be able to infer that $E_k(doc\_2)$ is the encrypted value of 'doc_2'. Therefore CSP_1 can associate all patients having $Id\_doc =$'$E_k(doc\_2)$' to the doctor whose name is '$N.Baines$'.

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 [23] to retrieve data with PIR using keywords search over many data structures such as binary trees and perfect hashing. In the next part of this paper, we will explain how we can use PIR keyword-based technique to ensure our *semi join* queries privacy requirement.

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

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. 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* will create an indexed structure over each indexed attributes in the database. Therefore, We implement over these attributes two types of indices: B+ trees [24] and hash tables [25]. In the following subsections, we present then discuss the PIR keyword-based protocol using both index structures.

6.1.1 PIR based on B+ Trees

Private Block Retrieval (PBR) is a practical extension of PIR in which a user retrieves an n-bit block instead of retrieving only a single bit. Therefore, to be able to use B+ tree structure with the PBR, we consider each node or leaf in the B+ trees as a data block. However, in most cases, B+ tree nodes and leaves do not have the same size, so they cannot be used directly as all PBR approaches require that data blocks are of equal size. Thus, a required stage consists in adding padding data to nodes and leaves in order to have the same size for all B+ tree elements.

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 to the corresponding *PIR server* the Relation schema name and the attribute name over which the *semi join* query has to be performed.

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 Algorithm 3 and Algorithm 4 for a description of the PIR keyword-based protocol algorithms used in the *Server* and the *Client* parts. We illustrate the execution of a semi-Join query using the PIR keyword-based in the example below.

```
input :
        BPT = {B_1, ..., B_n}   /* B-Plus Tree over indexed attributes*/
while True do
    Request ← handle_client_request()
    if Request is PQR then
        /* PQR : Pre-Query Request */
        (TabName, AttriName) ← Request
        B ← GetAssociatedBPT(TabName, AttriName)
        Root_B ← GetRootNode(B)
        ReplyToClient(Root_B)
    end
    if Request is PIRQ then
        /* PIRQ : PIR Query */
        result ← compute(Request)
        ReplyToClient(result)
    end
end
```

Algorithm 3: SemiJoin PIR keywordbased query (server)

```
input :
        tabName, attrName   /* Table and Attribute where the semi-join will be performed */
        value       /* Semi-join condition value */
Node ← send_PQR_request(tabName, attrName)
while Node is not leaf_node do
    foreach elem in Node do
        findLink ← False
        if Key(elem) < value then
            Node ← PIR_Query(IndexOfLeftChild(elem))
            findLink ← True
            break
        end
    endfch
    if findLink = False then
        Node ← PIR_Query(IndexOfRightChild(elem))
    end
end
foreach elem in Node do
    if Key(elem) = value then
        return Data(elem)
    end
endfch
```

Algorithm 4: SemiJoin PIR keywordbased query (client)

Example 6.2 : Consider the query $Q2_2$ used in Example 5.2. We suppose that $\delta = \{doc\_3, doc\_69\}$. the execution of $Q2_2$ using PIR keyword-based protocol over B+ trees data structures is as follows:

1. The *Query Executor* sends $(Doctor, Id\_doctor)$ to the *PIR server*.

2. The *PIR server* sends the root node of the B+ tree corresponding to the received $(Doctor, Id\_doctor)$. Suppose that this root node is as presented in the table below

i_1	doc_11	i_2	doc_5	i_3
•		•		•

   Note that i_1, i_2, i_3 are the indexes to the next level nodes. doc_11 and doc_5 are the root node keys.

3. The *Query Executor* compares the elements of δ with the received nodes keys, the type of the elements of δ and the root node keys is *String*.

   (a) The *Query Executor* wants to retrieve the node containing the key doc_3, due to the fact that $doc\_11 < doc\_3 < doc\_5$ and based on the received root node, the *Query Executor* will retrieve the node indexed by i_2.

   (b) The *Query Executor* needs also to retrieve the node containing the key doc_19, seeing that $doc\_5 < doc\_69$ and based on the received root node, the *Query Executor* will retrieve the node indexed by i_3.

   For each index to be retrieved i_i, the *Query Executor* sends an encoded PIR query $PIR(i_i)$ to the *PIR server*. This process will be executed until the leaves of the B+ tree are reached. From the retrieved leaves, the *Query Executor* gathers tuples in which their keys are element of $\{doc\_3, doc\_69\}$.

Theorem 2. Let $\mathscr{D}$ be a multi-relation normalized database, $\mathscr{F} = \{F_1, F_2\}$ be a fragmentation of $\mathscr{D}$, 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* sub-queries is guaranteed.

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. □

The particularity of B+ tree structures is that data appears only in the leaves while internal nodes is mainly used to guide the search. B+ tree's leaves are linked together to simplify sequential data access which gives the ability to perform in an efficient manner cardinality queries and range queries. However, the use of B+ tree data structure presents two disadvantages : first, as we have shown in the previous example, the *Query Executor* will use PIR queries to run down the tree and privately retrieve blocks (leaves) which contain records that match the semi-join condition. In each layer of the B+ tree, *Query Executor* should send a PIR query to the *PIR server* to get the addresses of the next layer nodes to be retrieved. Thus, for a k-layers B+ tree, *Query Executor* needs at least k PIR queries to reach the leaf layer, which is expensive in terms of communication and execution time. Second, several records which will not be part of the final result of the *semi join* query will be retrieved as a B+ tree leaf may contain several records having different index values.

6.1.2 PIR based on Hash Table

Hash table is a data structure that implements a mapping form keys to values. It is represented by an array in which data is accessed through a special index. The idea behind using hash tables is to map the indexed attribute (Primary key or foreign key) values to the set of corresponding records. These indexed attribute values will be the keywords which are used to search corresponding records stored in the hash tables. Hash tables are composed of set of sequential hash buckets. We will consider each set of records having the same keyword (indexed attribute value) as an hash buckets. The index of each bucket in the hash table is calculated using a minimal perfect hash function [26] that maps n keywords to n consecutive integers.

We describe the use of hash table with the protocol PIR to perform *semi join* queries with the following four steps :

- `Setup step` – The *PIR server* create an hash table over each indexed attributes of the database. This stage is carried out only once as created hash tables will be used for subsequent *semi join* queries.

- `Step 1` – The *Query Executor* sends to the *PIR server* the name of the table and the name of the attribute on which the *semi join* is performed.

- `Step 2` – *PIR server* picks up the hash tree corresponding to the received couple (table, attribute) on which the *semi join* is performed and sends back to the *Query Executor* a set of metadata

allowing the construction of the minimal perfect hash function used for building the corresponding hash table.

- `Step 3` – Using the received metadata, the *Query Executor* derives the minimal perfect hash function and calculates for each constant value in the *semi join* condition, the corresponding bucket index in which records corresponding to that constant value are stored. These calculated indexes are also the blocks numbers in the hash table index on the server. Next, the *Query Executor* will use PIR query to retrieve data blocks having the calculated indexes.

The advantage behind using hash tables lies in the fact that only one PIR query is needed to retrieve a data bloc instead of n PIR query when using a B+ tree (n represents the height of the B+ tree). Moreover, using hash tables, retrieved blocks will contain only records that match the original query of the user.

Example 6.3 : Consider the query $Q2_2$ used in Example 5.2. We suppose that $\delta = \{doc\_3, doc\_69\}$. the execution of $Q2_2$ using PIR keyword-based protocol over Hash tables data structures is as follows:

1. The *Query Executor* sends $(Doctor, Id\_doctor)$ to the *PIR server*.
2. The *PIR server* sends to the *Query Executor* a set of metadata allowing the construction of the minimal perfect hash function f used for building the corresponding hash table.
3. Using received minimal perfect hash function f, the *Query Executor* computes for each element in δ, the corresponding block index in which records corresponding to that element are stored. Suppose that $f(doc\_3) = i_1$ and $f(doc\_69) = i_2$, the *Query Executor* sends to the *PIR server* $PIR(Doctor, i_1)$ and $PIR(Doctor, i_2)$ to privately retrieve blocks having indexes i_1 and i_2.

7 Implementation and Evaluation

We have developed a prototype for our approach, it is composed mainly from two main components: (1) A Client entity written in C++. Using regular expression offered by the *boost* library [27], we give the ability to the client Entity to transform received queries into a QEP. Further, the Client uses the Crypto++ library [28] to perform different cryptographic operations. (2) The server entity. For the Server entity, we have used STX B+ Tree [29] and CMPH (C Minimal Perfect Hash) [30, 31] libraries to build and manipulate B+ tree and hash tables structures used by the PIR protocol. To give PIR functionality to the Server entity, we have used Percy++ [32, 33]. Finally, for relational database server we used MySQL [34].

7.1 Experimental Design

For our benchmarking, we used the relational database schema D composed of three relations as follows:

Patient($\underline{Id\_patient}, Name, SSN, Dob, Gender, ZIP, Illness$)
Doctor($\underline{Id\_doctor}, Name, Specialty$)
Examination($\underline{Id\_examination}, date, medical\_report, Id\_doc*, Id\_pat*$)

Note that the attributes Id_doc* and Id_pat* are two foreign keys that reference respectively the primary key of the table *Patient* ($Id\_patient$) and the primary key of the table *Examination* ($Id\_examination$).

In tables *Patient*, *Doctor* and *Examination*, we inserted respectively 10^6, 10^3 and 10^5 records. Fragments schemes obtained from the application of our secure fragmentation algorithm are represented below:

F_1 : **Patient**($\underline{salt}, enc, Id\_patient, Name, Dob, Gender, ZIP$)
Doctor($\underline{Id\_doctor}, Name, Specialty$)
F_2 : **Examination**($\underline{Id\_examination}, date, medical\_report, E_k(Id\_doc)*, E_k(Id\_pat*)$)
F_3 : **Patient**($\underline{salt}, enc, Illness$)

As we have previously seen, our approach is based on vertical fragmentation, the fragments of the table patient which are stored in F_1 and F_3 will be also composed of 10^6 records. Normally, each fragment F_1, F_2 and F_3 of the database $\mathscr{D}$ should be stored in a different Service Provider. In our experiments, we used three virtual machines, with each representing a Service Provider and running MySQL 5.5.31. All experimentations have been performed on an eight cores server (Intel(R) Xeon(R) CPU x5355, 2.66 GHz) with 12 GB of RAM and running Ubuntu Linux 10.04.

7.2 Evaluation

As we have seen in the section 5, two kinds of queries can be used in order to query distributed fragments: (1) Queries that can be executed over only one fragment. (2) Queries which require the interrogation of several fragments to be evaluated. To evaluate the efficiency of our approach, we tested for each kind of query, and according to the number of retrieved records, the time required to execute the query. To evaluate the query of the type (1), we used the query `Exp_Q1` :

```
Exp_Q1 :  SELECT Name, SSN, Dob
          FROM patient
          WHERE Gender = 'Male'
```

According to the used Fragments schemes, `Exp_Q1` is executed over the fragment F_1. To be able to control the number of returned records, we used the clause SQL `LIMIT` (`LIMIT 0, number_of_record`). The Figure 3 shows the execution costs per number of retrieved records. Note that in all experiments the cost of data transfer between the Server and the Client is negligible because both parties are installed in the same experimentation server.

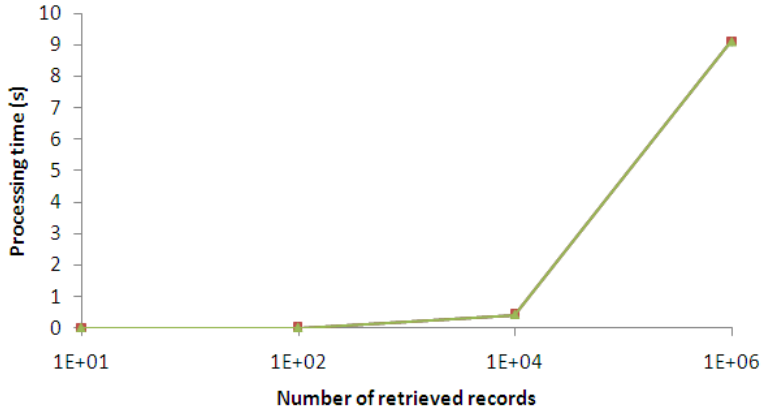

Figure 3: Execution costs per number of retrieved records for the query Exp_Q1

For queries of the type (2), our approach will use *semi_join* with the PIR keyword-based protocol in order to join fragmented data while preserving the protection of sensitive associations. In this case, two kinds of data structure can be used : B+ tree and hash table.

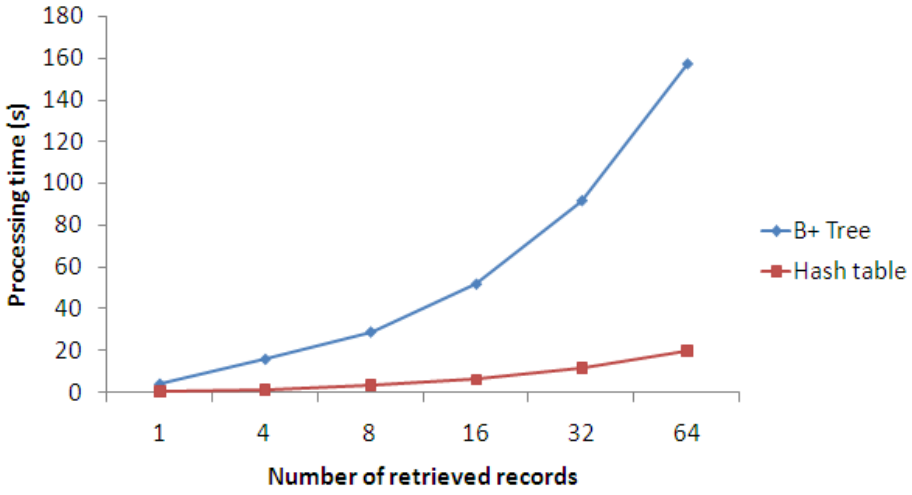

Figure 4: Execution costs per number of retrieved records for the query Exp_Q1 over B+ Tree and Hash Table data structures

For a better comparaison of the use of B+ trees and Hash tables with the keyword-based PIR, we executed the query `Exp_Q2` using the keyword-based PIR over both data structures.

```
Exp_Q2 :  SELECT Name_pat, SSN
          FROM patient, examination e
          WHERE e.date= ?x
```

Figure 4 compares the processing costs for the query `Exp_Q1` using the keyword-based PIR over both data structures.

We make the following observations. First, as expected, the use of hash tables as the data structures used by the PIR keyword-based in order to perform *semi join* queries is much more efficient compared to the use of B+ trees with the PIR keyword-based. This can be explained by the fact that in the case of B+ tree, the *Client* must run down the tree using a PIR query in each level of the tree to retrieve data blocks stored in the leaf level of the B+ tree. While, in the case of hash tables, the *Client* needs to perform only one PIR query to get the data block containing requested records. For instance, the height of the corresponding B+ tree of the table *Patient* is 5, then the *Client* must perform 5 PIR queries to be able to retrieve corresponding records. Second, another reason to the inefficiency of the use of B+ trees compared to the use of hash tables with the PIR keyword-based is due to the fact that the size of constructed B+ trees are much more bigger than constructed hash tables which will introduce an execution overhead when performing PIR queries.

8 Conclusion

In this paper, we have presented an approach based on fragmentation, encryption and query privacy techniques enabling privacy-preserving of outsourced multi-relation databases. We presented different techniques that we have used to decompose multi-relational databases in the aim to protect sensitive associations, then we demonstrate how our decomposition techniques help in achieving data confidentiality. We presented a querying technique that optimizes and executes queries in this distributed system and how we can improve the security of the querying technique in order to protect data confidentiality under a collaborative Cloud storage service providers model. Our future work include enhanced query

optimization and execution techniques to overcome some limitations of our approach, such as processing nested queries.

References

[1] A. Bkakria, F. Cuppens, N. Cuppens-Boulahia, and J. M. Fernandez, "Confidentiality-preserving query execution of fragmented outsourced data," in *Proc. of the 2013 EurAsia Conference on Information and Communicatiaon Technology (ICT-EurAsia'13), Yogyakarta, Indonesia, LNCS*, vol. 7804. Springer-Verlag, March 2013, pp. 426–440.

[2] H. Hacigümüs, B. R. Iyer, C. Li, and S. Mehrotra, "Executing SQL over encrypted data in the database-service-provider model," in *Proc. of the 2002 ACM SIGMOD International Conference on Management of Data, Madison, Wisconsin, USA*. ACM, June 2002, pp. 216–227.

[3] H. Hacigümüs, S. Mehrotra, and B. R. Iyer, "Providing database as a service," in *Proc. of the 18th International Conference on Data Engineering (ICDE'02), San Jose, California, USA*. IEEE, February-March 2002, pp. 29–38.

[4] B. Hore, S. Mehrotra, and G. Tsudik, "A privacy-preserving index for range queries," in *Proc. of the 30th International Conference on Very Large Data Bases (VLDB'04), Toronto, Canada*. Morgan Kaufmann, September 2004, pp. 720–731.

[5] Z.-F. Wang, J. Dai, W. Wang, and B. Shi, "Fast query over encrypted character data in database," in *Proc. of the 1st International Symposium on Computational and Information Science (CIS'04), Shanghai, China, LNCS*, vol. 3314. Springer-Verlag, December 2004, pp. 1027–1033.

[6] Z.-F. Wang, W. Wang, and B. Shi, "Storage and query over encrypted character and numerical data in database," in *Proc. of the 5th International Conference on Computer and Information Technology (CIT'05), Shanghai, China*. IEEE Computer Society, September 2005, pp. 77–81.

[7] E. Damiani, S. D. C. di Vimercati, S. Jajodia, S. Paraboschi, and P. Samarati, "Balancing confidentiality and efficiency in untrusted relational DBMSs," in *Proc. of the 10th ACM Conference on Computer and Communications Security (CCS'03), Washington, D.C., USA*, October 2003, pp. 93–102.

[8] A. Ceselli, E. Damiani, S. D. C. di Vimercati, S. Jajodia, S. Paraboschi, and P. Samarati, "Modeling and assessing inference exposure in encrypted databases," *ACM Transactions on Information and System Security*, vol. 8, no. 1, pp. 119–152, 2005.

[9] E. Damiani, S. D. C. di Vimercati, S. Foresti, P. Samarati, and M. Viviani, "Measuring inference exposure in outsourced encrypted databases," in *Quality of Protection - Security Measurements and Metrics, Advances in Information Security*, D. Gollmann, F. Massacci, and A. Yautsiukhin, Eds. Springer-Verlag, 2006, vol. 23, pp. 185–195.

[10] R. Agrawal, J. Kiernan, R. Srikant, and Y. Xu, "Order-preserving encryption for numeric data," in *Proc. of the 2004 ACM SIGMOD International Conference on Management of Data, Paris, France*. ACM, June 2004, pp. 563–574.

[11] A. Boldyreva, N. Chenette, Y. Lee, and A. O'Neill, "Order-preserving symmetric encryption," in *Proc. of the 28th Annual International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT'09), Cologne, Germany, LNCS*, vol. 5479. Springer-Verlag, April 2009, pp. 224–241.

[12] G. Özsoyoglu, D. A. Singer, and S. S. Chung, "Anti-Tamper Databases: Querying encrypted databases," in *Proc. of the IFIP TC-11 WG 11.3 7th Annual Working Conference on Data and Application Security (DBSec'03), Estes Park, Colorado, USA*. Kluwer, August 2003, pp. 133–146.

[13] L. Xiao and I.-L. Yen, "Security analysis for order preserving encryption schemes," in *Proc. of the 46th Annual Conference on Information Sciences and Systems (CISS'12), Princeton, New Jersey, USA*. IEEE, March 2012, pp. 1–6.

[14] J. Biskup, M. Preuß, and L. Wiese, "On thes inference-proofness of database fragmentation satisfying confidentiality constraints," in *Proc. of the 14th International Conference on Information Security (ISC'11), Xi'an, China, LNCS*, vol. 7001. Springer-Verlag, October 2011, pp. 246–261.

[15] A. Hudic, S. Islam, P. Kieseberg, S. Rennert, and E. Weippl, "Data confidentiality using fragmentation in cloud computing," *International Journal of Pervasive Computing and Communications*, vol. 9, no. 1, 2013.

[16] G. Aggarwal, M. Bawa, P. Ganesan, H. Garcia-Molina, K. Kenthapadi, R. Motwani, U. Srivastava, D. Thomas, and Y. Xu, "Two can keep a secret: A distributed architecture for secure database services," in *Proc. of the 2nd Biennial Conference on Innovative Data Systems Research (CIDR'05), Asilomar, California, USA*, January 2005, pp. 186–199.

[17] V. Ciriani, S. D. C. di Vimercati, S. Foresti, S. Jajodia, S. Paraboschi, and P. Samarati, "Fragmentation and encryption to enforce privacy in data storage," in *Proc. of the 12th European Symposium On Research In Computer Security (ESORICS'07), Dresden, Germany, LNCS*, vol. 4734. Springer-Verlag, September 2007, pp. 171–186.

[18] ——, "Fragmentation design for efficient query execution over sensitive distributed databases," in *Proc. of the 29th IEEE International Conference on Distributed Computing Systems (ICDCS 2009), Montreal, Québec, Canada*. IEEE, June 2009, pp. 32–39.

[19] V. Ciriani, S. D. C. di Vimercati, S. Foresti, G. Livraga, and P. Samarati, "Enforcing confidentiality and data visibility constraints: An OBDD approach," in *Proc. of the 25th Annual WG 11.3 Conference on Data and Applications Security and Privacy (DBSec'11), Richmond, Virginia, USA, LNCS*, vol. 6818. Springer-Verlag, July 2011, pp. 44–59.

[20] S. B. Navathe, S. Ceri, G. Wiederhold, and J. Dou, "Vertical partitioning algorithms for database design," *ACM Transactions on Database Systems*, vol. 9, no. 4, pp. 680–710, 1984.

[21] M. Bellare, M. Fischlin, A. O'Neill, and T. Ristenpart, "Deterministic encryption: Definitional equivalences and constructions without random oracles," in *Proc. of the 28th International Cryptology Conference (CRYPTO'08), Santa Barbara, California, USA, LNCS*, vol. 5157. Springer-Verlag, August 2008, pp. 360–378.

[22] P. A. Bernstein and D.-M. W. Chiu, "Using semi-joins to solve relational queries," *Journal of the ACM*, vol. 28, no. 1, pp. 25–40, 1981.

[23] B. Chor, N. Gilboa, and M. Naor, "Private information retrieval by keywords," Department of Computer Science, Technion, Tech. Rep. TR CS0917, 1997.

[24] A. V. Aho, J. E. Hopcroft, and J. D. Ullman, *Data Structures and Algorithms*. Addison-Wesley, 1983.

[25] O. Amble and D. E. Knuth, "Ordered hash tables," *The Computer Journal*, vol. 17, no. 2, pp. 135–142, 1974.

[26] F. C. Botelho, R. Pagh, and N. Ziviani, "Simple and space-efficient minimal perfect hash functions," in *Proc. of the 10th International Workshop on Algorithms and Data Structures (WADS'07), Halifax, Canada*, vol. 4619. Springer-Verlag, August 2007, pp. 139–150.

[27] J. Maddock, "Regular expressions in c++," http://www.boost.org/, 2001.

[28] W. Dai, "The crypto++ library," http://www.cryptopp.com/, 1995.

[29] T. Bingmann, "STX B+ Tree C++ Template Classes v 0.8.3," http://panthema.net/2007/stx-btree/, 2008.

[30] F. C. Botelho and N. Ziviani, "External perfect hashing for very large key sets," in *Proc. of the 16th ACM Conference on Information and Knowledge Management (CIKM'07), Lisbon, Portugal*. ACM, November 2007, pp. 653–662.

[31] F. C. Botelho and N. Zivian, "CMPH : C minimal perfect hashing library," http://cmph.sourceforge.net/, 2007.

[32] I. Goldberg, "Improving the robustness of private information retrieval," in *Proc. of the 2007 IEEE Symposium on Security and Privacy (S&P'07), Oakland, California, USA*. IEEE, May 2007, pp. 131–148.

[33] ——, "Percy++ / PIR in C++," http://percy.sourceforge.net/, 2007.

[34] O. Corporation, "MySQL," http://www.mysql.com/.

Anis Bkakria received a master's degree in computer security and cryptology from University of Limoges. he is currently a PhD student at Télécom Bretgane of Institut Mines-TELECOM. His research interest is on the specification and deployment of security policies for outsourced data.

Frédéric Cuppens is a full professor at TELECOM Bretagne of Institut Mines-TELECOM and co-responsible of the CNRS LabSTICC SFIIS team on security, reliability and integrity of systems and information. He holds an engineering degree in computer science, a PhD and an HDR (Habilitation to supervise research). He has been working for more 20 years on various topics of computer security including definition of formal models of security policies, access control to network and information systems, intrusion detection, reaction and counter-measures, and formal techniques to refine security policies and prove security properties. He has published more than 150 technical papers in refereed journals and conference proceedings. He served on several conference program committees as member or as general chair. He was the Programme Committee Chair of several conferences including ESORICS 2000, IFIP SEC 2004, SAR-SSI 2006, SETOP 2008, CRISIS 2011, PST 2011 and DBSEC 2012.

Nora Cuppens-Boulahia is an associate researcher at TELECOM Bretagne of Institut Mines-TELECOM. She holds an engineering degree in computer science, a PhD from ENSAE (National Higher School of Aeronautics and Space) and an HDR from University of Rennes 1. Her research interest includes formalization of security properties and policies, cryptographic protocol analysis, formal validation of security properties and threat and reaction risk assessment. Her current research topics include a posteriori access and usage control and traceability, privacy preserving by query rewriting, data fragmentation and security of web services. She has published more than 100 technical papers in refereed journals and conference proceedings. She has been member of several international program committees in information security system domain and the Programme Committee Chair of SETOP 2008, SETOP 2009, SAR-SSI 2009, CRiSIS 2010, DPM 2011, SECOTS 2012, DBSEC 2012, PST 2012 and the co-general chair of ESORICS 2009. She is the French representative of IFIP TC11 "Information Security" and she is the co-responsible of the information system security axis of the French learned society SEE.

José M. Fernandez is an assistant professor in the Department of Computer & Software Engineering at the École Polytechnique de Montréal since 2004, where he heads the Laboratory for Information Security Research (SecSI). His current research interests include malware and botnet analysis, denial of service attacks, intrusion detection systems, security product testing methodologies, security of SCADA systems, and digital privacy. He is a member of the Board of Directors of the Anti-Malware Testing Standards Organization (AMTSO), an international organization trying to establish common standards for security product evaluation. He holds Bachelor's degrees in Mathematics and Computer Science and Engineering from MIT, a Master's in Cryptology from the University of Toronto, and a Ph.D. in Quantum Computing from the Université de Montréal.

David Gross-Amblard is a full professor at the IRISA Lab. and University of Rennes 1. He received MSc and PhD degree in Computer Science from the University of Paris XI, Orsay, France in 1996. He was a member of the Vertigo Team (CEDRIC Lab, Paris) in 2001 and the WebDam ERC Project in 2011. His current research interests are logical methods for data management, data security, social networks a crowdsourcing.