Enforcing Confidentiality Constraints on Sensitive Databases with Lightweight Trusted Clients

Valentina Ciriani[1], Sabrina De Capitani di Vimercati[1], Sara Foresti[1], Sushil Jajodia[2], Stefano Paraboschi[3], and Pierangela Samarati[1]

[1] Università degli Studi di Milano, 26013 Crema, Italia
firstname.lastname@unimi.it
[2] George Mason University, Fairfax, VA 22030-4444, USA
jajodia@gmu.edu
[3] Università degli Studi di Bergamo, 24044 Dalmine, Italia
parabosc@unibg.it

Abstract. Existing approaches for protecting sensitive information stored (outsourced) at external "honest-but-curious" servers are typically based on an overlying layer of encryption that is applied on the whole information, or use a combination of fragmentation and encryption. The computational load imposed by encryption makes such approaches not suitable for scenarios with lightweight clients.
In this paper, we address this issue and propose a novel model for enforcing privacy requirements on the outsourced information which departs from encryption. The basic idea of our approach is to store a small portion of the data (just enough to break sensitive associations) on the client, which is trusted being under the data owner control, while storing the remaining information in clear form at the external (honest-but-curious) server. We model the problem and provide a solution for it aiming at minimizing the data stored at the client. We also illustrate the execution of queries on the fragmented information.

1 Introduction

The design of distributed databases and associated techniques have been a topic of interest in the 1970s, at the birth of relational technology. The scenario that was considered in those years was significantly different from the current scenario: the emphasis was on the implementation of systems owned by a large organization managing information systems in several centers that offered the opportunity for the processing of distributed queries. The current ICT scenario is instead characterized by many important opportunities for the use of distributed databases where previous assumptions do not hold. Two important differences compared to traditional approaches are: *1)* the need to integrate the services of database providers that do not belong to the same organization; *2)* the presence of a variety of platforms, with an increase in the number and availability of devices that have access to a network connection, together with the presence

of powerful servers offering significant computational and storage resources. The first aspect forces the requirement to specify security functions limiting access to the information stored in the databases. The second aspect instead forces an environment where the data and computational tasks are carefully balanced between the lightweight device and a powerful remote server. The two aspects are strictly related, since the servers are typically owned by service providers offering levels of cost, availability, reliability and flexibility difficult to obtain from in-house operations. Note that we classify as "lightweight" any device that exhibits performance or storage features that are significantly worse than what can be offered for the specific application by a remote service provider. Mobile devices certainly fit this description, but the scenario can be extended to generic computational platforms.

The motivation of this work lies in the desire to define novel solutions for the processing of large data collections, which we assume managed by traditional database technology, in a scenario where we are interested in using the services of a honest-but-curious powerful server, with a robust guarantee that confidentiality of information is protected.

In the literature, this problem has been addressed by combining fragmentation and encryption, thus splitting sensitive information among two or more servers and encrypting information whenever necessary [1, 8, 9]. In [1], the sensitive relation is split into *two fragments* stored at *two non communicating servers*, which must not know the identity of each other. This limitation on the number of fragments produced implies that it is not always possible to protect the confidentiality of information by simply splitting the involved attributes into two fragments and therefore can be encrypted. The main limitations of this solution are that: *1)* the absence of communication between the two servers is clearly a strong assumption and difficult to enforce in real environments; *2)* the query evaluation process requires the data owner to interact with both servers for joining (if needed) the two fragments and to decrypt the attributes possibly encrypted that appear in the query. In [8, 9] these limitations are removed since a relation R is split into *two or more fragments possibly stored on one server* and limits the use of encryption for protecting the single sensitive attributes. Furthermore, for query execution efficiency, attributes that are not represented in the clear within a fragment are represented in encrypted form, providing the nice property that each fragment completely represents the original relation. The consequence of this design choice is that to evaluate a query, it is sufficient to access a single fragment, thus avoiding join operations (needed with the previous paradigm), which are quite expensive. This solution however still requires the client to possibly decrypt the attributes appearing in encrypted form in the fragment for evaluating a condition on them or for returning them to the user.

A common assumption of these solutions is that encryption is an unavoidable price to be paid to protect the information. There are, however, situations where encryption may not be applicable for protecting information. One issue can be the computational load imposed by encryption. For instance, in systems more constrained in battery power rather than memory capacity, it is beneficial to

spend some memory space to save on power consumption. Also, in real systems keys can become compromized, and keys can become lost, making the protection of the system dependent on the quality of key management services, rather than on the quality and strength of the cryptographic functions. Since key management is known to be a difficult task, we expect that an encryption-less solution can be of interest for many important applications.

To address these situations, we propose a paradigm shift where information is protected without encryption. The basic idea is that a small portion of the sensitive information can be stored on the client, trusted for both managing the information and for releasing such a sensitive information only to the authorized users, while the remaining information can be stored on an external server. Obviously, from the information stored on the external server it should not be possible to reconstruct a sensitive association (confidentiality constraint) since otherwise a privacy violation occurs. Since we do not want to remove the assumption that the external servers are all honest-but-curious, the client is the only entity in the system that can manage a portion of sensitive data. Sensitive data and associations can then be protected by splitting the original relation R into two fragments, denoted F_o and F_s, stored at the client and at a honest-but-curious storage server, respectively.

In the following sections, we describe how to correctly apply the abovementioned approach. The remainder of the paper is organized as follows. Section 2 presents the basic concepts of the model. Section 3 introduces the idea of a correct and minimal fragmentation. Section 4 analyzes the minimization problem. Section 5 discusses the execution of queries in this architecture. Section 6 presents related work. Finally, Sect. 7 draws some conclusions.

2 Basic Concepts

We consider a scenario where, consistently with other proposals (e.g., [1, 8, 10]), the data to be protected are represented with a single relation r over a relation schema $R(a_1,\ldots,a_n)$. We use the standard notations of the relational database model. Also, when clear from the context, we will use R to denote either the relation schema R or the set of attributes in R.

Privacy requirements are represented by *confidentiality constraints*, which express restrictions on the single or joint visibility (association) of attributes in R. Confidentiality constraints are formally defined as follows [1, 8].

Definition 1 (Confidentiality constraint). *Let $R(a_1, \ldots, a_n)$ be a relation schema, a* confidentiality constraint *c over R is a subset of the attributes in R.*

While simple in its definition, the confidentiality constraint construct supports the definition of different privacy requirements that may need to be expressed. A singleton constraint states that the *values* assumed by an attribute are considered sensitive and therefore cannot be accessed by an external party. A non-singleton constraint states that the *association* among values of given

PATIENT

SSN	Name	DoB	ZIP	Job	Illness	Cause
123-45-6789	Alice	80/02/11	20051	Secretary	asthma	Dust allergy
987-65-4321	Bob	85/05/22	22034	Student	fracture	Car accident
147-85-2369	Carol	73/07/30	22039	Secretary	carpal tunnel	Secretary
963-85-2741	David	75/11/26	20051	Lawyer	hypertension	Stress
789-65-4123	Emma	90/03/15	22035	Student	asthma	Student
123-65-4789	Fred	68/08/07	22034	Accountant	hypertension	Wrong diet

(a)

$c_0 =$ {SSN}
$c_1 =$ {Name,Illness}
$c_2 =$ {Name,Cause}
$c_3 =$ {DoB,ZIP,Illness}
$c_4 =$ {DoB,ZIP,Cause}
$c_5 =$ {Job,Illness}
$c_6 =$ {Job,Cause}

(b)

Fig. 1. An example of relation (a) and of confidentiality constraints over it (b)

attributes is sensitive and therefore should not be outsourced to an external party.

Example 1. Figure 1 illustrates plaintext relation PATIENT (a) and the confidentiality constraints defined over it (b).

- c_0 is a singleton constraint indicating that the list of SSNs of patients is considered sensitive.
- c_1 and c_2 state that the association of patients' names with their illnesses and with the causes of the illnesses, respectively, is considered sensitive.
- c_3 and c_4 state that the association of patients' dates of birth and ZIP codes with their illnesses and with the causes of the illnesses, respectively, is considered sensitive; these constraints derive from c_1 and c_2 and from the fact that DoB and ZIP together could be exploited to infer the name of patients (i.e., they can work as a quasi-identifier [10]).
- c_5 and c_6 state that the association of patients' jobs with their illnesses and causes of the illnesses, respectively, is considered sensitive, since it could be exploited to establish a correlation between the job and the illness of a patient.

The satisfaction of a constraint c_i clearly implies the satisfaction of any constraint c_j such that $c_i \subseteq c_j$. We are therefore interested in enforcing a set $\mathcal{C} = \{c_1, \ldots, c_m\}$ of *well defined* constraints, where $\forall c_i, c_j \in \mathcal{C}, i \neq j,\ c_i \not\subset c_j$.

3 Correct and Minimal Fragmentation

Given a set $\mathcal{C}$ of confidentiality constraints over relation R, our goal is then to split R into two fragments F_o and F_s, in such a way that all sensitive data and associations are protected. F_o is stored at the client (owner) side, while F_s is stored at the external server side. It is easy to see that, since there is no encryption, singleton constraints can be protected only by storing the corresponding attributes at the client side only. Therefore, each singleton constraint $c=\{a\}$ is enforced by inserting a into F_o and by not allowing a to appear in the schema of F_s. Association constraints are enforced via fragmentation, that is, by splitting the attributes composing the constraints between F_o and F_s. The resulting fragmentation $\mathcal{F}=\langle F_o, F_s\rangle$ should then satisfy two important requirements: *1)* all attributes in R should appear in at least one fragment to avoid loss of information; *2)* the confidentiality constraints should be properly protected, meaning that from the fragment stored at the external server (F_s) it should not be possible to reconstruct the content of the original relation R. These two requirements are formally captured by the following definition of *correct fragmentation*.

Definition 2 (Fragmentation correctness). *Let $R(a_1, \ldots, a_n)$ be a relation schema, $\mathcal{C}=\{c_1, \ldots, c_m\}$ be a set of well defined confidentiality constraints over R, and $\mathcal{F}=\langle F_o,F_s\rangle$ be a fragmentation for R, where F_o is stored at the client and F_s is stored at a storage server.*
$\mathcal{F}$ is a correct fragmentation *for R, with respect to $\mathcal{C}$, iff: 1) $F_o \cup F_s = R$ (completeness) and 2) $\forall c \in \mathcal{C}$, $c \not\subseteq F_s$ (confidentiality).*

The first condition requires every attribute in the schema of the original relation R to be represented in at least a fragment. The second condition requires the fragment stored at the external storage server F_s to not be a superset of any constraint. Note that the second condition applies only to F_s since F_o, under the client control and therefore accessible only to authorized users, can contain sensitive data and associations. For instance, fragmentation $\mathcal{F}=\langle\{$`SSN`,`Name`,`ZIP`,`Job`$\}$, $\{$`DoB`,`Illness`,`Cause`$\}\rangle$ is a correct fragmentation for PATIENT in Fig. 1(a), with respect to the confidentiality constraints in Fig. 1(b).

Since the client is supposed to have a more expensive storage, we are interested in computing a correct fragmentation that *minimizes* the client's storage due to the direct management of F_o. We then require that the two fragments F_o and F_s be disjoint, as formally stated by the following definition of *non redundant* fragmentation.

Definition 3 (Non redundant fragmentation). *Let $R(a_1, \ldots, a_n)$ be a relation and $\mathcal{C}$ a set of well defined constraints over R. A fragmentation $\mathcal{F}=\langle F_o,F_s\rangle$ of R is* non redundant *iff $F_o \cap F_s = \emptyset$.*

The non redundancy property does not affect the correctness of a fragmentation (Definition 2). Suppose that $\mathcal{F}=\langle F_o,F_s\rangle$ is a correct fragmentation for R with respect to $\mathcal{C}$, and that $F_o \cap F_s \neq \emptyset$. Any attribute a that appears in both F_o and F_s can be removed from F_o without violating Definition 2,

since a is still represented in F_s and does not violate any constraint. In other words, a correct fragmentation can always be made non redundant by removing from F_o the attributes belonging to the intersection between F_o and F_s. For instance, consider fragmentation $\mathcal{F}'$=⟨{SSN,Name,ZIP,Job,Illness,Cause}, {DoB,Illness,Cause}⟩, which is a correct fragmentation for relation PATIENT in Fig. 1(a) with respect to the confidentiality constraints in Fig. 1(b). Fragmentation $\mathcal{F}$=⟨{SSN,Name,ZIP,Job}, {DoB,Illness,Cause}⟩ obtained by removing the attributes in $F_o \cap F_s$={Illness,Cause} from F_o is still correct and satisfies Definition 3.

Our problem then consists in computing a fragmentation that is correct and non redundant and that minimizes the storage at the client site. Let $size(a)$ be the physical size of attribute a and $size(\mathcal{F})$ be the size of the fragmentation $\mathcal{F}$, computed as the physical size of the attributes composing F_o, that is, $size(\mathcal{F})=\sum_{a \in F_o} size(a)$. Our problem can be formally defined as follows.

Problem 1 (Minimal storage). Given a relation $R(a_1, \ldots, a_n)$ and a set of well defined confidentiality constraints $\mathcal{C}=\{c_1, \ldots, c_m\}$ over R, compute a fragmentation $\mathcal{F}=\langle F_o, F_s \rangle$ that satisfies the following conditions:

1. $\mathcal{F}$ is correct (Definition 2);
2. $\mathcal{F}$ is non redundant (Definition 3);
3. $\nexists \mathcal{F}'$ such that $size(\mathcal{F}') < size(\mathcal{F})$ and $\mathcal{F}'$ satisfies the two conditions above.

Note that, given a relation R and a set of confidentiality constraints $\mathcal{C}$ over it, there may exist different minimal storage fragmentations, all characterized by the same size (i.e., with the same size for F_o, which may however be composed of a different subset of attributes of R). We consider all such solutions equivalent.

Note that, whenever the physical size of the attributes composing R is not known (or the size of the attributes in R is similar and therefore cannot be considered as a discriminating factor for choosing attributes to be stored at the client site), Problem 1 can be exploited to compute a fragmentation that minimizes the number of attributes stored at the client site by simply setting $size(a)$=1, for all $a \in R$.

Example 2. Consider relation PATIENT and the confidentiality constraints on it in Fig. 1 and suppose that: $size$(SSN)=9, $size$(Name)=15, $size$(DoB)=8, $size$(ZIP)=5, $size$(Job)=10, $size$(Illness)=15, and $size$(Cause)=100. Fragmentation $\mathcal{F}$ = ⟨{SSN,Name,ZIP,Job}, {DoB,Illness,Cause}⟩, with $size(\mathcal{F})$= 9+15+ 5+10 = 39, is a minimal storage fragmentation for relation PATIENT with respect to $\mathcal{C}$. Suppose now that for all attributes a in PATIENT, $size(a)$=1. Fragmentation $\mathcal{F}'$ = ⟨{SSN,Illness,Cause}, {Name,DoB,ZIP,Job}⟩ is a minimal (storage) fragmentation, minimizing the number of attributes composing F_o.

In the following, we formally analyze the minimal storage problem, proving that it is NP-hard, and we present how a well-known approximation algorithm can be adapted for its solution.

INPUT
$R(a_1, \ldots, a_n)$ /* relation schema */
$\mathcal{C}=\{c_1, \ldots, c_m\}$ /* well-defined constraints */
$size(a)$, for all $a \in R$ /* physical size of attributes */

OUTPUT
A correct, non-redundant fragmentation $\mathcal{F}=\langle F_o, F_s \rangle$

MAIN
$to\_solve := \mathcal{C}$ /* constraints to be solved */
$F_o := \emptyset$ /* current solution */
while $to\_solve \neq \emptyset$ **do**
  Let $a \in (R \setminus F_o)$ be the attribute maximizing $|\{c \in to\_solve: a \in c\}|/size(a)$
  $to\_solve := to\_solve \setminus \{c \in to\_solve: a \in c\}$
  $F_o := F_o \cup \{a\}$
$F_s := R \setminus F_o$
return$(\langle F_o, F_s \rangle)$

Fig. 2. Approximation algorithm for the minimal storage problem

4 Analysis of the Minimization Problem

The minimal storage problem (Problem 1) directly corresponds to the classical NP-hard *Weighted Minimum Hitting Set* problem [13], which can be formulated as follows: *Given a finite set S, a collection C of subsets of S, a weight function $w : S \to \mathbb{R}^+$, find a hitting set S', that is, a subset S' of S containing at least one element for each subset in C, such that $w(S') = \sum_{s \in S'} w(s)$ is minimum.*

The correspondence follows from the observation that any solution of the minimal storage problem must insert in F_o at least one attribute from each constraint in $\mathcal{C}$ to guarantee fragmentation correctness (Definition 2). The minimal storage problem can then be formulated as the problem of finding the set of attributes with lowest size (to be inserted in F_o) for breaking each constraint. It is then immediate to see that there is a correspondence between the two problems (minimal storage and weighted minimum hitting set) that can be determined by taking R as the finite set S, $\mathcal{C}$ as the collection C, and by taking as a weight function w the attribute size (i.e., $w(a) = size(a)$). Since the two problems are equivalent, the minimal storage problem (Problem 1) is NP-hard.

We also note that, by fixing $size(a)=1$ for all $a \in R$, the minimal storage problem directly corresponds to the classical NP-hard *Minimum Hitting Set* problem [13], which can be formulated as follows: *Given finite set S and a collection C of subsets of S, find a hitting set S', that is, a subset S' of S containing at least one element for each subset in C, such that $|S'|$ is minimum.* The minimum hitting set problem is equivalent to the minimum set covering problem [3]. Therefore, approximation algorithms and nonapproximability results for the minimum set covering problem also apply to the minimum hitting set problem. More precisely, the two problems are approximable within $(1 + \ln |S|)$ in polynomial time [16]. It is interesting to note that this approximation ra-

iteration	*to_solve*	$\lvert\{c \in to\_solve\colon a \in c\}\rvert / size(a)$							F_o
		SSN	Name	DoB	ZIP	Job	Illness	Cause	
1	$c_0,c_1,c_2,c_3,c_4,c_5,c_6$	1/9	2/15	2/8	2/5	2/10	3/15	3/100	ZIP
2	c_0,c_1,c_2,c_5,c_6	1/9	2/15	0	–	2/10	2/15	2/100	ZIP,Job
3	c_0,c_1,c_2	1/9	2/15	0	–	–	1/15	1/100	ZIP,Job,Name
4	c_0	1/9	–	0	–	–	0	0	ZIP,Job,Name,SSN
	$\emptyset$	–	–	0	–	–	0	0	ZIP,Job,Name,SSN

Fig. 3. An example of execution of the algorithm in Fig. 2

tio is particularly accurate, since the minimum set cover, as well as the minimum hitting set, is not approximable within $(1-\varepsilon)\ln|S|$, for any $\varepsilon > 0$ unless NP $\subset$ DTIME$(n^{\log\log n})$ [12]. These results on the approximability of the minimum hitting set problem also apply to the weighted hitting set problem and, consequently, to the minimal storage problem (Problem 1).

The classical approximation algorithm for the weighted minimum hitting set problem [16] (with approximation ratio $(1+\ln|S|)$) follows a greedy strategy. Figure 2 represents this approximation algorithm working on an instance of our minimal storage problem (Problem 1). Initially, the set *to_solve* of constraints to be solved is initialized to $\mathcal{C}$ and F_o is initialized to the empty set. At each iteration of the **while** loop, the algorithm chooses the attribute a that does not belong to F_o and that maximizes the ratio between the number of constraints in *to_solve* in which it is involved, and a's size. (The non-weighted version of the algorithm simply chooses the attribute that maximizes the number of non solved constraints in which it is involved.) Hence, a is inserted into F_o and set *to_solve* of non solved constraints is updated removing the constraints in which the chosen attribute a is involved. The **while** loop terminates when all constraints are solved (*to_solve*=$\emptyset$). Finally, F_s is obtained as the complement of F_o with respect to R.

Example 3. Figure 3 represents the execution, step by step, of the algorithm in Fig. 2 on relation PATIENT in Fig. 1(a), considering the confidentiality constraints in Fig. 1(b), and supposing *size*(SSN)=9, *size*(Name)=15, *size*(DoB)=8, *size*(ZIP)=5, *size*(Job)=10, *size*(Illness)=15, and *size*(Cause)=100. The first column in the table represents the set of constraints that are still unsolved; the subsequent seven columns represent, for each attribute in R, the number of unsolved constraints in which they are involved, divided by the attribute's size; the last column represents the attributes composing F_o. Symbol – associated with an attribute means that the attribute already belongs to F_o and therefore it does not need to be considered anymore. The solution computed by the algorithm is $\mathcal{F} = \langle\{$SSN,Name,ZIP,Job$\}, \{$DoB,Illness,Cause$\}\rangle$, which is a minimal fragmentation for the considered example.

F_o

tid	SSN	Name	ZIP	Job
1	123-45-6789	Alice	20051	Secretary
2	987-65-4321	Bob	22034	Student
3	147-85-2369	Carol	22039	Secretary
4	963-85-2741	David	20051	Lawyer
5	789-65-4123	Emma	22035	Student
6	123-65-4789	Fred	22034	Accountant

F_s

tid	DoB	Illness	Cause
1	80/02/11	asthma	Dust allergy
2	85/05/22	fracture	Car accident
3	73/07/30	carpal tunnel	Secretary
4	75/11/26	hypertension	Stress
5	90/03/15	asthma	Student
6	68/08/07	hypertension	Wrong diet

Fig. 4. Physical fragments corresponding to the fragmentation computed by the approximation algorithm on the relation and confidentiality constraints in Fig. 1

5 Query Execution

The fragmentation of a relation R implies that only the two fragments F_o and F_s, which are stored in place of the original relation to satisfy confidentiality constraints, are used for query execution. However, since users authorized to access the original relation R should not worry about whether or not R has been fragmented, they formulate queries referring to R. Such queries need then to be translated into equivalent queries operating on F_o and/or F_s.

To guarantee the reconstruction of the content of the original relation R (lossless join property), at the physical level the two fragments F_o and F_s must have a common key attribute [2]. We then assume F_o and F_s have a common tuple id (attribute `tid`) that can be either: *1)* the key attribute of the original relation, if it is not sensitive, or *2)* an attribute that does not belong to the schema of the original relation R and that is added to F_o and F_s during the fragmentation process. Figure 4 illustrates the physical fragments for the minimal storage fragmentation of Example 3.

We consider SELECT-FROM-WHERE SQL queries of the form q="SELECT A FROM R WHERE C", where A is a subset of the attributes in R, and $C=\bigwedge_i cnd_i$ is a conjunction of basic conditions of the form $(a_i\ op\ v)$, $(a_i\ op\ a_j)$, or $(a_i$ IN $\{v_1,\ldots,v_k\})$, with a_i and a_j attributes in R, $\{v,v_1,\ldots,v_k\}$ constant values in the domain of a_i, and op a comparison operator in $\{=, >, <, \leq, \geq, \neq\}$. In the following, $Attr(cnd_i)$ is used to denote the attributes on which condition cnd_i operates. We now describe the query translation process.

5.1 Classification of Conditions

Condition $C = \bigwedge_i cnd_i$ in the WHERE clause of a query can be split in three conditions, namely C_o, C_s, and C_{so}, depending on the attributes involved in the subexpression and that determine the party (client and/or storage server) responsible for its evaluation.

- $C_o = \bigwedge_i cnd_i : Attr(cnd_i) \subseteq F_o$ is the conjunction of the conditions that can be evaluated only by the *client*, independently from the server, since they involve only attributes stored at the client.
- $C_s = \bigwedge_i cnd_i : Attr(cnd_i) \subseteq F_s$ is the conjunction of the conditions in C that can be evaluated by the *storage server*, independently from the client, since they involve only attributes stored at the remote server. Note that, since the attributes stored at the server can be safely communicated to the client, C_s could also be evaluated by the client. However, this last option is highly impractical and should be avoided, because it requires to send to the client the projection over F_s of the attributes $Attr(cnd_i)$, for each cnd_i in C_s. This option would reduce the advantages of data outsourcing.
- $C_{so} = \bigwedge_i cnd_i : Attr(cnd_i) \cap F_s \neq \emptyset \wedge Attr(cnd_i) \cap F_s \neq \emptyset$ is the conjunction of conditions in C of the form $(a_i \ op \ a_j)$, where $a_i \in F_o$ and $a_j \in F_s$ or viceversa. Therefore, these conditions require information from the server and from the client to be evaluated, since they involve both attributes stored at the client and attributes stored at the server. Note that the conditions in C_{so} involve attributes of F_o that cannot be released to the external server, since they would possibly violate confidentiality constraints. Therefore, C_{so} can be evaluated only by the client.

Example 4. Consider the fragmentation in Fig. 4 and the query retrieving the names and dates of birth of people affected by asthma, living in area 22034 or 20051, and such that the cause of their illness is their job:

SELECT `Name`, `DoB`
FROM `Patient`
WHERE (`ZIP` IN $\{22034, 20051\}$) AND (`Illness`="asthma") AND (`Cause`=`Job`)

C_o = {`ZIP` IN $\{22034, 20051\}$}, since attribute `ZIP` belongs to F_o; C_s = {`Illness` = "asthma"}, since attribute `Illness` belongs to F_s; and, C_{so} = {`Cause` = `Job`}, since attributes `Job`$\in F_o$ and `Cause`$\in F_s$.

5.2 Query Evaluation Strategies

The evaluation process of a query q on R can follow two different strategies, depending on the order in which conditions C_o, C_s, and C_{so} are evaluated. The *Server-Client* strategy evaluates C_s at the server side and then evaluates both C_o and C_{so} at the client side. The *Client-Server* strategy first evaluates C_o at the client side, evaluates C_s at the server side, and finally checks C_{so} at the client side again. The left-hand side of Fig. 5 illustrates the two algorithms

Algorithm	Example
Server-Client strategy	
1. Let q = SELECT A FROM R WHERE C	q = SELECT Name, DoB FROM Patient WHERE (ZIP IN {22034,20051}) AND (Illness = "asthma") AND (Cause = Job)
2. Split C into subexpressions C_o, C_s, and C_{so}	C_o={ZIP IN{22034,20051}}; C_s={Illness="asthma"}; C_{so}={Cause=Job}
3. A_{qs}:=$(F_s \cap A) \cup \{a \in F_s \mid \exists cnd \in C_{so}, a \in Attr(cnd)\}$	A_{qs} = {DoB, Cause}
4. Send q_s = SELECT tid, A_{qs} FROM F_s WHERE C_s to the storage server	q_s = SELECT tid,DoB,Cause FROM F_s WHERE Illness = "asthma"
5. Receive the result R_s of q_s from the server	
6. Execute q_{so} = SELECT A FROM F_o JOIN R_s ON F_o.tid=R_s.tid WHERE $C_o \wedge C_{so}$	q_{so} = SELECT Name, DoB FROM F_o JOIN R_s ON F_o.tid=R_s.tid WHERE (ZIP IN {22034,20051}) AND (Cause = Job)
7. Return the result R_q of q_{so} to the user	
Client-Server strategy	
1. Let q = SELECT A FROM R WHERE C	q = SELECT Name, DoB FROM Patient WHERE (ZIP IN {22034,20051}) AND (Illness = "asthma") AND (Cause = Job)
2. Split C into subexpressions C_o, C_s, and C_{so}	C_o={ZIP IN{22034,20051}}; C_s={Illness="asthma"}; C_{so}={Cause=Job}
3. A_{qs}:=$(F_s \cap A) \cup \{a \in F_s \mid \exists cnd \in C_{so}, a \in Attr(cnd)\}$	A_{q_s} = {DoB, Cause}
4. Execute q_o = SELECT tid FROM F_o WHERE C_o	q_o = SELECT tid FROM F_o WHERE ZIP IN {22034,20051}
5. Send the result R_o of q_o to the storage server	
6. Send q_s = SELECT tid, A_{qs} FROM F_s JOIN R_o ON F_s.tid=R_o.tid WHERE C_s to the storage server	q_s = SELECT tid,DoB,Cause FROM F_s JOIN R_o ON F_s.tid=R_o.tid WHERE Illness = "asthma"
7. Receive the result R_s of q_s from the server	
8. Execute q_{so} = SELECT A FROM F_o JOIN R_s ON F_o.tid=R_s.tid WHERE C_{so}	q_{so} = SELECT Name, DoB FROM F_o JOIN R_s ON F_o.tid=R_s.tid WHERE Cause = Job
9. Return the result R_q of q_{so} to the user	

Fig. 5. Algorithm for evaluating query q on fragmentation $\mathcal{F}$ and an example of its execution

implementing the Server-Client and Client-Server strategies, respectively, executed by the client for translating and evaluating q. In the following, we briefly illustrate the working of the two algorithms.

– *Server-Client* strategy. The basic idea is that first the server evaluates the conditions in C_s on F_s, and then the client refines the result by filtering out the tuples that do not satisfy either C_o or C_{so}.
The corresponding algorithm receives in input the query q (step 1) to be

evaluated and returns the relation R_q resulting from its evaluation. The algorithm then splits the condition C in the WHERE clause in three subexpressions, C_o, C_s, and C_{so}, on the basis of the attributes involved in the basic conditions composing C (step 2). The algorithm identifies the set A_{q_s} of attributes that belong to F_s, but that are necessary to the client for completing the evaluation of q, since either they belong to A or appear in a basic condition in C_{so} (step 3). The algorithm then defines, and sends to the storage server, the query q_s operating on F_s and evaluating condition C_s (step 4). The SELECT clause of this query is composed of A_{q_s} and attribute `tid`, which is necessary to join the result of q_s with F_o. Then the server computes and returns the result R_s of the evaluation of q_s to the client (step 5). The client defines and directly executes the query q_{so} operating on the join between R_s and F_o and evaluating both C_o and C_{so} (step 6). The relation resulting from the evaluation of query q_{so} corresponds to the result of the original query q, which is then returned to the user (step 7).

- *Client-Server* strategy. The basic idea is that first the client evaluates the conditions in C_o on F_o, the server then refines the result by filtering out the tuples that do not satisfy C_s, and finally the client discards the tuples that do not satisfy C_{so}. The first three steps of the corresponding algorithm are the same as for strategy Server-Client. After having split the conditions (step 2) and identified attributes A_{q_s} (step 3), the client defines and executes the query q_o operating on F_o and evaluating condition C_o (step 4). The SELECT clause of this query is composed of attribute `tid` only, which is the only attribute that can be communicated to the remote server (step 5) and is needed to perform join. The algorithm sends to the storage server the query q_s, operating on the join between R_o and F_s and evaluating condition C_s (step 6) for execution. The SELECT clause of query q_s (as for Server-Client strategy) is composed of A_{q_s} and attribute `tid`. The server computes q_s and returns its result R_s to the client (step 7). The client defines and directly executes query q_{so} operating on the join between R_s and F_o and evaluating C_{so} (step 8). The relation resulting from the evaluation of query q_{so}, corresponding to the result of the original query q, is returned to the requesting user (step 8).

Example 5. Consider relation PATIENT in Fig. 1(a), its fragmentation in Fig. 4, and the query q introduced in Example 4. $A_{q_s} = \{$`DoB`,`Cause`$\}$, since `DoB` belongs to the SELECT clause of the original query, while `Cause` is involved in a condition in C_{so}. The right-hand side of Fig. 5 illustrates the queries generated by the algorithm for translating q in a set of equivalent queries operating on $\mathcal{F}$, with the Server-Client and Client-Server strategies.

The choice between the Server-Client and Client-Server strategies depends on the possible leakage of information that the Client-Server strategy may cause. If the storage server is supposed to know or can infer query q (e.g., because the query is publicly available) the Client-Server strategy cannot be adopted because the storage server can infer that the tuples in R_o are all and only the

tuples that satisfy C_o, thus causing a leakage of information. As an example, consider query q = "SELECT `Illness` FROM `Patient` WHERE `Name` = "Alice"" over relation PATIENT. If we adopt the Client-Server strategy, q_o = "SELECT `tid` FROM F_o WHERE `Name` = "Alice"" returns only one tuple with `tid`=1. Knowing q, the storage server can, from the result, reconstruct the sensitive associations between `Name` and `Illness` and between `Name` and `Cause` (violating c_1 and c_2) for the tuple with `tid`=1. To avoid information leakage, the client can add noise to the result of query q_o, by artificially inserting the id of some tuples that do not satisfy C_o. In this case, query q_{so} should evaluate both C_{so} and C_o, to remove from the final result the tuples artificially added to R_o.

If the storage server does not to know (and cannot infer) q or R_o is adequately protected, both the Server-Client and Client-Server strategies can be adopted without privacy violations. In this case, the choice between the two strategies can only be based on performances. Following the criterion usually adopted by distributed database systems, the most selective condition is evaluated first (i.e., the sub-query with the smallest result is anticipated). Therefore, if C_o is more selective than C_s, we adopt the Client-Server strategy and viceversa, if C_s is more selective than C_o, we adopt the Server-Client strategy. Note that the selectivity of query q_o needs to consider also the possible noise added to R_o for privacy purposes.

6 Related Work

Most of the research on the outsourced data paradigm assume the data to be entirely encrypted, focusing on the design of techniques for the efficient execution of queries (Database As a Service paradigm) [7, 14, 15, 19]. The first proposal suggesting the combined use of fragmentation and encryption for enforcing privacy constraints has been presented in [1]. This technique is based in the assumption that data are split over two non communicating honest-but-curious database servers and resorts to encryption any time two fragments are not sufficient for enforcing confidentiality constraints. The model presented in [8, 9] removes the limiting assumption that the two storage servers must not communicate, by splitting the data over different fragments. The advantage is that only sensitive attributes need to be encrypted, while sensitive associations can always be solved via fragmentation.

In this paper, differently from previous approaches, we aim at solving confidentiality constraints without resorting to encryption, by storing a portion of the sensitive data at the client site. The main advantage of this novel solution is that it simplifies the architecture and produces a system characterized by better maintenance, avoiding the issues related with key management.

On another line of related work, classical proposals on the management of queries in centralized and distributed systems [4, 6, 17] describe how efficient query plans can be obtained. These solutions, however, cannot be applied in our context, since they do not take into account possible information leakage to the storage server, which can violate confidentiality constraints.

An affinity to the work presented in this paper can be found in [5, 11]. Although these approaches share with our problem the common goal of enforcing confidentiality constraints on data, they are concerned with retrieving a data classification (according to a multilevel mandatory policy) that ensures sensitive information is not disclosed and do not consider the fragmentation technique.

The problem of fragmenting relational databases has been addressed in the literature [18], with the main goal of improving query evaluation efficiency. However, these approaches are not applicable to the considered scenario, since they do not take into consideration privacy requirements.

7 Conclusions

The paper presented an approach for the management of relational data where a honest-but-curious server is used to manage all the data that do not permit to violate confidentiality constraints. Given the continuous increase in the size, variety, and accessibility needs of data collections containing sensitive information, together with the availability of novel network-enabled computing platforms, we envision a significant domain for the application of the techniques presented here or their extensions. We see an opportunity for extending this work to more general databases and network applications, like Web-based management of email and office automation tasks, remote backup and data outsourcing services, and rentable computational services.

Acknowledgements

This work was supported in part by the EU within the 7FP project under grant agreement 216483 "PrimeLife". The work of Sushil Jajodia was partially supported by the National Science Foundation under grants CT-0716323, CT-0627493, and IIS-04300402 and by the Air Force Office of Scientific Research under grants FA9550-07-1-0527 and FA9550-08-1-0157. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the sponsoring organizations.

References

1. 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 Second Biennial Conference on Innovative Data Systems Research (CIDR 2005)*, Asilomar, CA, USA, January 2005.
2. P. Atzeni, S. Ceri, S. Paraboschi, and R. Torlone. *Database systems - Concepts, languages and architectures.* McGraw-Hill Book Company, 1999.
3. G. Ausiello, A. D'Atri, and M. Protasi. Structure preserving reductions among convex optimization problems. *Journal of Computer and System Sciences*, 21(1):136–153, August 1980.

4. P. Bernstein, N. Goodman, E. Wong, C. Reeve, and J. J.B. Rothnie. Query processing in a system for distributed databases (SDD-1). *ACM Transactions on Database Systems*, 6(4):602–625, December 1981.
5. J. Biskup, D. Embley, and J. Lochner. Reducing inference control to access control for normalized database schemas. *Information Processing Letters*, 106(1):8–12, March 2008.
6. S. Ceri and G. Pelagatti. *Distributed Databases: Principles and Systems*. McGraw-Hill, 1984.
7. A. Ceselli, E. Damiani, S. De Capitani di Vimercati, S. Jajodia, S. Paraboschi, and P. Samarati. Modeling and assessing inference exposure in encrypted databases. *ACM Transactions on Information and System Security*, 8(1):119–152, February 2005.
8. V. Ciriani, S. De Capitani 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 2007)*, Dresden, Germany, September 2007.
9. V. Ciriani, S. De Capitani di Vimercati, S. Foresti, S. Jajodia, S. Paraboschi, and P. Samarati. Fragmentation design for efficient query execution over sensitive distributed databases. In *Proc. of the 29th International Conference on Distributed Computing Systems (ICDCS 2009)*, Montreal, Canada, June 2009.
10. V. Ciriani, S. De Capitani di Vimercati, S. Foresti, and P. Samarati. k-Anonymity. In T. Yu and S. Jajodia, editors, *Secure Data Management in Decentralized Systems*. Springer-Verlag, 2007.
11. S. Dawson, S. De Capitani di Vimercati, P. Lincoln, and P. Samarati. Maximizing sharing of protected information. *Journal of Computer and System Sciences*, 64(3):496–541, May 2002.
12. U. Feige. A threshold of $\ln n$ for approximating set cover. *Journal of the ACM*, 45(4):634–652, July 1998.
13. M. Garey and D. Johnson. *Computers and intractability; a guide to the theory of NP-completeness*. W.H. Freeman, 1979.
14. H. Hacigümüs, B. Iyer, and S. Mehrotra. Providing database as a service. In *Proc. of the 18th International Conference on Data Engineering (ICDE 2002)*, San Jose, CA, USA, February/March 2002.
15. H. Hacigümüs, B. Iyer, S. Mehrotra, and C. Li. Executing SQL over encrypted data in the database-service-provider model. In *Proc. of the 21st ACM SIGMOD International Conference on Management of Data*, Madison, WI, USA, June 2002.
16. D. Johnson. Approximation algorithms for combinatorial problems. In *Proc. of the ACM Symposium on Theory of Computing (STOC 1973)*, Austin, TX, USA, April/May 1973.
17. D. Kossmann. The state of the art in distributed query processing. *ACM Computing Surveys*, 32(4):422–469, December 2000.
18. S. Navathe and M. Ra. Vertical partitioning for database design: A graphical algorithm. In *Proc. of the ACM SIGMOD International Conference on Management of Data*, Portland, OR, USA, June 1989.
19. H. Wang and L. V. S. Lakshmanan. Efficient secure query evaluation over encrypted XML databases. In *Proc. of the 32nd International Conference on Very Large Data Bases (VLDB 2006)*, Seoul, Korea, September 2006.