Keep a Few: Outsourcing Data While Maintaining Confidentiality

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

[1] DTI - Università degli Studi di Milano, 26013 Crema, Italia
firstname.lastname@unimi.it

[2] CSIS - George Mason University, Fairfax, VA 22030-4444, USA
jajodia@gmu.edu

[3] DIIMM - Università degli Studi di Bergamo, 24044 Dalmine, Italia
parabosc@unibg.it

Abstract. We put forward a novel paradigm for preserving privacy in data outsourcing which departs from encryption. The basic idea behind our proposal is to involve the owner in storing a limited portion of the data, and maintaining all data (either at the owner or at external servers) in the clear. We assume a relational context, where the data to be outsourced is contained in a relational table. We then analyze how the relational table can be fragmented, minimizing the load for the data owner. We propose several metrics and present a general framework capturing all of them, with a corresponding algorithm finding a heuristic solution to a family of NP-hard problems.

1 Introduction

The correct management of data with adequate support for reliability and availability requirements presents extremely significant economies of scale. There is an important cost benefit for individuals and small/medium organizations in outsourcing their data to external servers and delegating to them the responsibility of data storage and management. Important initiatives already operate in this market (e.g., Amazon's S3 service) and a significant expansion in this direction is expected in the next few years. However, while on the one hand there is a desire to outsource data management, there is on the other hand an equally strong need to properly protect data confidentiality. Certain data, or - more often - associations among data, are sensitive and cannot be released to others or be stored outside the owner's control. The success and wide adoption of data outsourcing solutions strongly depends on their ability to properly support such confidentiality requirements.

In the last few years, the problem of outsourcing data subject to confidentiality constraints has raised considerable attention, and various research activities have been carried out, providing the foundation for a large future deployment of these solutions. All existing proposals share the assumption that sensitive information stored at external servers can be protected by proper encryption.

More recent proposals combine encryption with fragmentation. While varying in the amount of encryption required, all existing approaches assume the use of encryption whenever needed for privacy, and operate under the implicit assumption that the owner aims at externally storing the complete database. Encryption is therefore considered a necessary price to be paid for protecting the confidentiality of information. Although cryptographic tools enjoy today a limited cost and an affordable computational complexity, encryption carries however the burden of managing keys, which makes it not applicable for many scenarios. In addition, while the cost of encryption/decryption operations may be negligible, the execution of queries on encrypted data greatly increases the computational effort required to the DBMS, considerably impacting its applicability for real-world applications.

In this paper we propose a paradigm shift for solving the problem, which departs from encryption, thus freeing the owner from the burden of its management. In exchange, we assume that the owner, while outsourcing the major portion of the data at one or more external servers, is willing to locally store a limited amount of data. The owner-side storage, being under the owner control, is assumed to be maintained in a trusted environment. The main observation behind our approach is that often is the association among data to be sensitive, in contrast to the individual data themselves. Like recent solutions, we therefore exploit data fragmentation to break sensitive associations; but, in contrast to them, we assume the use of fragmentation only. Basically, the owner maintains a small portion of the data, just enough to protect sensitive values or their associations. The contribution of this paper is threefold. First, we propose a novel approach to the problem of outsourcing data in the presence of privacy constraints, based on involving the owner as a trusted party for limited storage (Sect. 3). Second, aiming at minimizing the load required to the owner, we investigate possible metrics according to which the owner's load could be characterized (Sect. 4). The different metrics can be applicable in different scenarios, depending on the owner's preferences and/or on the information (on the data or on the system's workload) available at design time. Third, we introduce a new theoretical problem, which is a generalization of a hitting set problem, show how all the problems of minimizing the owner load with respect to the different metrics can be characterized as specific instances of this problem, and present a heuristic algorithm for its solution (Sect. 5).

2 Basic Concepts

We consider a scenario where, consistently with other proposals (e.g., [1,4,6]), 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.

Patient

<u>SSN</u>	Name	DoB	Race	Job	Illness	Treatment	HDate
123-45-6789	White	82/12/09	asian	waiter	laryngitis	antibiotic	09/01/02
987-65-4321	Taylor	75/03/05	white	nurse	diabetes	insulin	09/01/06
963-85-2741	Harris	68/05/11	white	banker	laryngitis	antibiotic	09/01/08
147-85-2369	Ripley	90/02/06	black	waiter	flu	aspirin	09/01/10

(a)

$c_0 = \{$`SSN`$\}$
$c_1 = \{$`Name,Illness`$\}$
$c_2 = \{$`Name,Treatment`$\}$
$c_3 = \{$`DoB,Race,Illness`$\}$
$c_4 = \{$`DoB,Race,Treatment`$\}$
$c_5 = \{$`Job,Illness`$\}$

(b)

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

Protection requirements are represented by *confidentiality constraints*, which express restrictions on the single or joint visibility (association) of attributes in R and are formally defined as follows [1,4].

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 attributes in R ($c \subseteq R$).*

While simple, confidentiality constraints of this form allow the representation of different protection 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 (*association constraint*) states that the *association* among values of given attributes is sensitive and therefore should not be released to an external party.

Example 1. Figure 1 illustrates relation Patient (a) and a set of confidentiality constraints defined over it (b): c_0 is a singleton constraint indicating that the list of `SSN`s of patients is considered sensitive; $c_1 \ldots c_5$ are association constraints stating that the association between all the values assumed by the specified attributes should not be disclosed. Constraints c_3 and c_4 derive from c_1 and c_2, respectively, and from the fact that attributes `DoB` and `Race` together could be exploited to retrieve the name of patients (i.e., they can work as a quasi-identifier [6]).

The satisfaction of a constraint c_i clearly implies the satisfaction of any constraint c_j such that $c_i \subseteq c_j$. We therefore assume the set $\mathcal{C}_f = \{c_1, \ldots, c_m\}$ to be *well defined*, $\forall c_i, c_j \in \mathcal{C}_f : i \neq j \Rightarrow c_i \not\subseteq c_j$.

To satisfy confidentiality constraints, we consider an approach based on data *fragmentation*. Fragmenting R means splitting its attributes into different *fragments* (i.e., different subsets) in such a way that only attributes in the same fragment are visible in association [1,4]. For instance, splitting `Name` and `Illness` into two different fragments offers visibility of the two lists of values but not of

F_o

t_id	SSN	Illness	Treatment
1	123-45-6789	laryngitis	antibiotic
2	987-65-4321	diabetes	insulin
3	963-85-2741	laryngitis	antibiotic
4	147-85-2369	flu	aspirin

F_s

t_id	Name	DoB	Race	Job	HDate
1	White	82/12/09	asian	waiter	09/01/02
2	Taylor	75/03/05	white	nurse	09/01/06
3	Harris	68/05/11	white	banker	09/01/08
4	Ripley	90/02/06	black	waiter	09/01/10

Fig. 2. An example of physical fragments for relation PATIENT in Fig. 1(a)

their association. A fragment is said to *violate* a constraint if it contains all the attributes in the constraint. For instance, a fragment containing both `Name` and `Illness` violates constraint c_1.

3 Rationale of Our Approach

Departing from previous solutions resorting to encryption or unlinkable fragments in the storage of sensitive attributes or associations at the external server, our solution involves the data owner in storing (and managing) a small portion of the data, while delegating the management of all other data to external parties. We consider the management of a small portion of the data to be an advantage with respect to the otherwise required encryption management and computation. We then propose to maintain sensitive attributes at the owner side. Sensitive associations are instead protected by ensuring that not all attributes in an association are stored externally. In other words, for each sensitive association, the owner should locally store at least an attribute. With this fragmentation, the original relation R is then split into two fragments, called F_o and F_s, stored at the data owner and at the server side, respectively.

To correctly reconstruct the content of the original relation R, at the physical level F_o and F_s have a common *tuple identifier* (attribute `t_id` as in Fig. 2) that can correspond to the primary key of the original relation, if it is not sensitive, or can be an attribute that does not belong to the schema of the original relation R and that is added to F_o and F_s after the fragmentation process. We consider this a physical-level property and ignore the common attribute in the reminder of the paper.

Given a set $\mathcal{C}_f$ of confidentiality constraints over relation R, our goal is then to split R into two fragments: F_o, stored at the owner side, and F_s, stored at the server side, in such a way that all sensitive data and associations are protected. It is easy to see that, since there is no encryption, singleton constraints can only be protected by storing the corresponding attributes at the owner side. Therefore, each singleton constraint $c=\{a\}$ is enforced by inserting a into F_o and by not allowing a to appear in F_s. Association constraints are enforced via fragmentation, that is, by splitting the attributes involved in the constraint between F_o and F_s. A fragmentation $\mathcal{F}=\langle F_o, F_s\rangle$ should satisfy the following conditions: *1)* all attributes in R should appear in at least one fragment, to avoid loss of information; *2)* the external fragment should not violate any confidentiality constraint. Note that this condition applies only to F_s, since F_o is accessible only to authorized users and therefore can contain sensitive data and/or associations. These conditions 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, $\mathcal{C}_f=\{c_1,\ldots,c_m\}$ be a well defined set of 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 owner and F_s is stored at a storage server. $\mathcal{F}$ is a* correct fragmentation *for R, with respect to $\mathcal{C}_f$, iff: 1) $F_o \cup F_s = R$ (completeness); 2) $\forall c \in \mathcal{C}_f$, $c \not\subseteq F_s$ (confidentiality); 3) $F_o \cap F_s = \emptyset$ (non-redundancy).*

In addition to the two correctness criteria already mentioned, Definition 2 includes also a condition imposing non redundancy. Besides avoiding usual replication problems, this condition intuitively avoids unnecessary storage at the data owner (there is no need to maintain information that is outsourced).

Given a relation schema $R(a_1,\ldots,a_n)$ and a set $\mathcal{C}_f$ of confidentiality constraints, our goal is then to produce a correct fragmentation that minimizes the owner's workload. For instance, a fragmentation where $F_o=R$ and $F_s=\emptyset$ is clearly correct but it is also undesirable (unless required by the confidentiality constraints), since it leaves to the owner the burden of storing all information and of managing all possible queries.

The owner's workload may be a concept difficult to capture, also since different metrics might be applicable in different scenarios (see Sect. 4). Regardless of the metrics adopted, we can model the owner workload as a weight function $w{:}\mathcal{P}(\mathcal{A}) \times \mathcal{P}(\mathcal{A}) \rightarrow \mathbb{R}^+$ that takes a pair $\langle F_o,F_s\rangle$ of fragments as input and returns the storage and/or the computational load at the owner side due to the management of F_o. Our problem can then be formally defined as follows.

Problem 1 (Minimal Fragmentation). Given a relation schema $R(a_1,\ldots,a_n)$, a set $\mathcal{C}_f=\{c_1,\ldots,c_m\}$ of well defined constraints over R, and a weight function w, determine a fragmentation $\mathcal{F}=\langle F_o,F_s\rangle$ that satisfies the following conditions: *1)* $\mathcal{F}$ is correct according to Definition 2; and *2)*$\nexists \mathcal{F}'$ such that $w(\mathcal{F}')<w(\mathcal{F})$ and $\mathcal{F}'$ is correct.

In the following, we present some possible fragmentation metrics and corresponding weight functions. We then introduce a modeling of the problem (which we prove to be NP-hard) that is able to capture, as special cases, all these weight functions and illustrate a heuristic algorithm for its solution.

4 Fragmentation Metrics

In our scenario, storage and computational resources offered by the external server are considered, for a given level of availability and accessibility, less expensive than the resources within the trust boundary of the owner. The owner has then a natural incentive to rely as much as possible, for storage and computation, on the external server. In the absence of confidentiality constraints, all

	Problem	Metrics	Weight function
Storage	Min-Attr	Number of attributes	$card(F_o)$
	Min-Size	Size of attributes	$\sum_{a \in F_o} size(a)$
Computation/traffic	Min-Query	Number of queries	$\sum_{q \in \mathcal{Q}} freq(q)$ s.t. $Attr(q) \cap F_o \neq \emptyset$
	Min-Cond	Number of conditions	$\sum_{cond \in Cond(\mathcal{Q})} freq(cond)$ s.t. $cond \cap F_o \neq \emptyset$

Fig. 3. Classification of the weight metrics and minimization problems

data would then be remotely stored and all queries would be computed by the external server. In the case of confidentiality constraints, as discussed in Sect. 3, the owner internally stores some attributes, and consequently is involved in some computation.

In this section we discuss several metrics (and corresponding weight functions to be minimized) that could be used to characterize the quality of a fragmentation, and therefore to determine which attributes are stored at the owner and which attributes are outsourced at the external server. The different metrics may be applicable to different scenarios, depending on the owner's preferences and/or on the specific knowledge (on the data or on the query workload) available at design time. We consider four possible scenarios, in increasing level of required knowledge. The first two scenarios support measuring storage, while the latter two scenarios support measuring computation. The scenario and corresponding weight functions are summarized in Fig. 3.

- *Min-Attr*. Only the relation schema (set of attributes) and the confidentiality constraints are known. The only applicable metric aims at minimizing the storage required at the owner side by *minimizing the number of the attributes* in F_o. The weight $w_a(\mathcal{F})$ of a fragmentation $\mathcal{F}$ is the number of attributes in F_o, that is: $w_a(\mathcal{F})=card(F_o)$. For instance, given fragmentation $\mathcal{F}=\langle\{$`SSN`,`Illness`,`Treatment`$\}$, $\{$`Name`,`DoB`,`Race`,`Job`,`HDate`$\}\rangle$ illustrated in Fig. 2, $w_a(\mathcal{F})=3$.
- *Min-Size*. Besides the mandatory knowledge of the relation schema and the confidentiality constraints on it, the size of each attribute is known. In this case, it is possible to produce a more precise estimate of the storage required at the owner side, aiming at *minimizing the physical size* of F_o, that is, the actual storage required by its attributes. The weight $w_s(\mathcal{F})$ of a fragmentation $\mathcal{F}$ is the physical size of the attributes in F_o, that is: $w_s(\mathcal{F})=\sum_{a \in F_o} size(a)$, where $size(a)$ denotes the physical size of attribute a. For instance, with respect to fragmentation $\mathcal{F}$ in Fig. 2 and the attributes size in Fig. 4(a), $w_s(\mathcal{F})=64$.
- *Min-Query*. In addition to the relation schema and the confidentiality constraints, a representative profile of the expected query workload is known. The profile defines for each query, the frequency of execution and the set of attributes evaluated by its conditions. The query workload profile is then a set of triples $\mathcal{Q}=\{(q_1, freq(q_1), Attr(q_1)), \ldots, (q_l, freq(q_l) Attr(q_l))\}$, where

Attribute a	$size(a)$
SSN	9
Name	20
DoB	8
Race	5
Job	18
Illness	15
Treatment	40
HDate	8

(a)

Query q	$freq(q)$	$Attr(q)$	$Cond(q)$
q_1	5	DoB, Illness	⟨Dob⟩, ⟨Illness⟩
q_2	4	Race, Illness	⟨Race⟩, ⟨Illness⟩
q_3	10	Job, Illness	⟨Job⟩, ⟨Illness⟩
q_4	1	Illness, Treatment	⟨Illness⟩, ⟨Treatment⟩
q_5	7	Illness	⟨Illness⟩
q_6	7	DoB, HDate, Treatment	⟨DoB,HDate⟩, ⟨Treatment⟩
q_7	1	SSN, Name	⟨SSN⟩, ⟨Name⟩

(b)

Fig. 4. An example of data (a) and workload (b) knowledge for relation Patient in Fig. 1(a)

$q_1, \ldots, q_l$ are the queries to be executed, for each q_i, $i = 1, \ldots, l$, $freq(q_i)$ is the expected execution frequency of q_i, and $Attr(q_i)$ the attributes appearing in the WHERE clause of query q_i. The first three columns of Fig. 4(b) illustrate a possible workload profile for relation Patient in Fig. 1(a). Knowledge on the workload allows the adoption of a metric evaluating the computational work required to the owner for executing queries. Intuitively, the goal is to *minimize the number of query executions that require processing at the owner*, producing immediate benefits in terms of the reduced level of use of the more expensive and less powerful computational services available at the owner. The weight $w_q(\mathcal{F})$ of a fragmentation $\mathcal{F}$ is then the number of times that the owner needs to be involved in evaluating queries, that is, the sum of the frequencies of queries whose set of attributes in the WHERE clause contain at least an attribute in F_o. Formally, $w_q(\mathcal{F})$=$\sum_{q \in \mathcal{Q}} freq(q)$ *s.t.* $Attr(q) \cap F_o \neq \emptyset$. For instance, with respect to the fragmentation $\mathcal{F}$ in Fig. 2 and the query workload in Fig. 4(b), $w_q(\mathcal{F})$=35.

– *Min-Cond*. In addition to the relation schema and the confidentiality constraints, a complete profile of the expected query workload is known. The complete profile assumes that the specific conditions (not only the attributes on which they are evaluated) appearing in each query are known. We assume SELECT-FROM-WHERE queries where the condition in the WHERE clause is a conjunction of simple predicates of the form $(a_i\ op\ v)$, or $(a_i\ op\ a_j)$, with a_i and a_j attributes in R, v a constant value in the domain of a_i, and op a comparison operator in $\{=, >, <, \leq, \geq, \neq\}$. The query workload profile is then a set of triples $\mathcal{Q}$=$\{(q_1, freq(q_1), Cond(q_1)), \ldots, (q_l, freq(q_l) Cond(q_l))\}$, where $q_1, \ldots, q_l$ are the queries to be executed, for each q_i, $i = 1, \ldots, l$, $freq(q_i)$ is the expected execution frequency of q_i, and $Cond(q_i)$ is the set of conditions appearing in the WHERE clause of query q_i. Each condition is represented as a single attribute or a pair of attributes. The first, second, and fourth columns of Fig. 4(b) illustrate a possible workload profile for relation Patient in Fig. 1(a).

For each condition appearing in some query, we define $freq(cond)$ as its overall frequency in the system; formally: $freq(cond)$=$\sum_q freq(q)$ *s.t.* $cond \in Cond(q)$. For instance, with reference to the workload in Fig. 4(b), $freq($Illness$)$=27. The precise characterization of the workload allows the definition of a metric to *minimize the number of conditions that require* *processing at the owner.* The weight $w_c(\mathcal{F})$ of a fragmentation $\mathcal{F}$ is the number of times that the owner needs to be involved in evaluating conditions in the query execution. Intuitively, this corresponds to the number of times the execution of queries requires evaluating a condition involving an attribute in F_o. Note that conditions are considered separately, hence the evaluation of n different conditions involving some attribute in F_o in a query q will contribute to the weight for $n \cdot \mathit{freq}(q)$. Formally, $w_c(\mathcal{F})=\sum_{cond\in Cond(\mathcal{Q})} \mathit{freq}(cond)$ *s.t.* $cond \cap F_o \neq \emptyset$, where $Cond(\mathcal{Q})$ denotes the set of all conditions of queries in $\mathcal{Q}$. For instance, with respect to the fragmentation $\mathcal{F}$ in Fig. 2 and to the query workload in Fig. 4(b), $w_c(\mathcal{F})$=36. Note that the minimization of the conditions executed at the owner's side has a direct relationship with the minimization of the traffic needed for receiving results of the portion of queries outsourced to the external server. As a matter of fact, minimizing the conditions executed by the owner is equivalent to maximizing the conditions outsourced to the external server, and therefore delegating to it as much computation as possible. In fact, since the result of evaluating a condition on a relation is a smaller relation, the greater the number of conditions outsourced to the external servers, the smaller will be the corresponding results to be received in response.

The different metrics above translate into different instances of Problem 1, by substituting w with the corresponding weight functions. In synthesis, the resulting instances of the problem aim at minimizing, respectively: the number of attributes in F_o (*Min-Attr*); the physical size of fragment F_o (*Min-Size*); the number of times queries requiring access to F_o need to be evaluated (*Min-Query*); the number of times conditions on F_o need to be evaluated (*Min-Cond*). Figure 3 summarizes the metrics previously discussed, indicating the name of the corresponding instantiations of Problem 1.

5 A General Modeling of the Minimization Problems

We start the analysis of the minimization problems previously introduced by first observing that the *Min-Attr* problem directly corresponds to the classical Minimum Hitting Set Problem (MHSP) [10], which can be formulated as follows: *Given a finite set A and a collection C of subsets of A, find a subset S (hitting set) of A such that S contains at least one element from each subset in C and $|S|$ is minimum.* It is easy to see that setting A as the set R of attributes and C as the set $\mathcal{C}_f$ of constraints, the solution S of the MHSP is the set of attributes that must be maintained in fragment F_o, since S contains the minimum number of attributes that must be kept by the owner for breaking all the confidentiality constraints. Analogously, the *Min-Size* problem directly corresponds to the classical Weighted Minimum Hitting Set Problem (WMHSP) [10] formulated as follows: *Given a finite set A, a collection C of subsets of A, a weight function $w : A \rightarrow \mathbb{R}^+$, find a hitting set S such that $w(S) = \sum_{a\in S} w(a)$ is minimum.* The correspondence is given by setting $w(a) = \mathit{size}(a)$, $\forall a\in R$.

Unfortunately, the two problems above (MHSP and WMHSP) are not sufficient for capturing all the different metrics that could be adopted, and therefore the different minimization problems described in the previous section. As a matter of fact, while all problems aim at the identification of a hitting set (as F_o must contain at least an attribute for each constraint) the criteria according to which such a hitting set should be minimized are different. In the following we define a general problem that is able to capture the different metrics.

5.1 The General Problem

We define a new problem, generalization of MHSP and WMHSP, which we call *Weighted Minimum Target Hitting Set Problem* (WMTHSP), as follows.

Problem 2 (WMTHSP). Given a finite set A, a set C of subsets of A, a set $\mathcal{T}$ (target) of subsets of A, and a weight function $w{:}\mathcal{T}{\rightarrow}\mathbb{R}^+$, determine a subset S of A that satisfies the following conditions: *1)* S contains at least one element from each subset in C (S is a hitting set of A); *2)* $\nexists S'$ such that S' is a hitting set of A and $\sum_{t\in\mathcal{T},t\cap S'\neq\emptyset} w(t) < \sum_{t\in\mathcal{T},t\cap S\neq\emptyset} w(t)$.

The weight of a set of attributes is the sum of the weights of the targets intersecting it; a solution of WMTHSP is a hitting set of attributes with minimum weight, that is, it minimizes the sum of the weights of the intersecting targets. As an example, consider the WMTHSP with $A = \{a,b,c,d,e,f,g\}$, $C = \{\{a,b,c\}\{b,c,d\}\{f,g\}\}$, and $\mathcal{T} = \{\{a,e\}\{c,f\}\{g\}\}$ with weights $w(\{a,e\}) = 1$, $w(\{c,f\}) = 3$, and $w(\{g\}) = 2$. A minimal solution to this problem is $S = \{b,g\}$, whose weight is $w(S) = 2$ (b does not intersect any target, while g intersects a target with weight 2).

The WMTHSP is NP-hard since the MHSP can be reduced to this problem by simply defining $\mathcal{T}{=}\{\{a_1\},\ldots,\{a_n\}\}$ and $w(\{a_1\}) = 1$, for all $i \in \{1,\ldots,n\}$. Minimizing $\sum_{t\in\mathcal{T},t\cap S\neq\emptyset} w(t)$ is equivalent to minimizing the cardinality of the hitting set S, since each set t in $\mathcal{T}$ corresponds to an element in A and $w(t) = 1$.

All our minimization problems can be reformulated as instances of the WMTHSP, remaining however NP-hard. The formulation of all our problems as a WMTHSP considers as sets A and C of WMTHSP the set R of attributes and a set $\mathcal{C}_f$ of confidentiality constraints, respectively. The definition of the target set $\mathcal{T}$ and of the corresponding weight function w is different depending on the problem (i.e., the metrics to be minimized). For all the instances of the problem, the solution S of WMTHSP corresponds to fragment F_o of the data owner. Fragment F_s can be simply defined as $R \setminus F_o$. Figure 5 summarizes the definition of the target $\mathcal{T}$ for the different problems, which we now discuss together with their computational complexity.

- *Min-Attr.* Each attribute $a{\in}R$ corresponds to a target with weight 1. Minimizing the sum of the weights in S corresponds therefore to minimize the number of elements in it, and therefore in F_o. As already observed, *Min-Attr* directly corresponds to the classical NP-hard MHSP and is therefore NP-hard.

Problem	**Target** $\mathcal{T}$	$w(t)\ \forall t\in\mathcal{T}$
Min-Attr	$\mathcal{T}=\{\{a\}\vert a\in R\}$	$w(t)=1$
Min-Size	$\mathcal{T}=\{\{a\}\vert a\in R\}$	$w(t)=size(a)\ s.t.\ \{a\}=t$
Min-Query	$\mathcal{T}=\{attr\vert\exists q\in\mathcal{Q},\ Attr(q)=attr\}$	$w(t)=\sum_{q\in\mathcal{Q}}freq(q)\ s.t.\ Attr(q)=t$
Min-Cond	$\mathcal{T}=\{cond\vert\exists q\in\mathcal{Q},\ cond\in Cond(q)\}$	$w(t)=freq(cond)\ s.t.\ cond=t$

Fig. 5. Reductions of the minimization problems to the WMTHSP

– *Min-Size.* Each attribute $a\in R$ corresponds to a target with as weight the size of the attribute. Recalling that the *Min-Size* problem is equivalent to the NP-hard WMHSP by setting $w(a)$ as the size of the attribute a, also the *Min-Size* problem is NP-hard.
– *Min-Query.* Each set *attr* of attributes characterizing some queries corresponds to a target with as weight the number of times the queries need to be evaluated, that is, the sum of the frequencies of the queries characterized by the set. The NP-hardness of *Min-Query* can be directly seen from the fact that the specific instance of workload having a query with frequency 1 for each attribute $a\in R$ (i.e., a query q with $Attr(q)=\{a\}$) corresponds to the *Min-Attr* problem and therefore the MHSP can be reduced to it.
– *Min-Cond.* Each condition *cond* corresponds to a target with as weight the frequency of the conditions, that is, the number of times the conditions need to be evaluated. Note that the specific instance of the *Min-Cond* problem, where all conditions are singleton (i.e., conditions of the form "$a_x\ op\ v$", where v is a constant value), can be formulated as a *Min-Size* problem, considering as the size of each attribute the number of times that conditions on it need to be evaluated. Such a specific instance of the *Min-Cond* corresponds to the WMHSP, and is therefore NP-hard. Consequently, the general *Min-Cond* problem is NP-hard.

5.2 Algorithm

Given the NP-hardness of our minimization problems, that is, of the instances of Problem 2 with respect to the different weight functions, we propose a heuristic algorithm for its solution. While not necessarily minimum, our solution ensures minimality, meaning that moving any attribute from F_o to F_s would violate at least a constraint.

Before illustrating the algorithm, we note that any solution must include all singleton constraints. In other words all attributes involved in singleton constraints must belong to F_o. Given this observation, we remove singleton constraints from the problem to be solved heuristically and implicitly assume their inclusion in the solution. Consistently, the input to the algorithm ignores all the targets including attributes in singleton constraints (intuitively, these targets have been already intersected and therefore there is no further weight to consider for them). In terms of our example, the unique singleton constraint is c_0, which implies that query q_7 is removed from the set $\mathcal{T}$ of targets for the *Min-Query* problem, while condition ⟨SSN⟩ is removed from the set $\mathcal{T}$ of targets for the *Min-Cond* problem.

```
MAIN
$A' := \emptyset$                                      /* initialization of the solution */
$PQ$ := Build_Priority_Queue($\mathcal{A}$,$\mathcal{C}$,$\mathcal{T}$,$w$)   /* initialization of the priority queue */
$E$ := Extract_Min($PQ$)                              /* $E$ is the element in $PQ$ that minimizes $E.w/E.n_c$ */
while ($E \neq$ NULL) $\wedge$ ($E.n_c \neq 0$) do      /* there are still constraints to be solved */
  $A' := A' \cup \{E.a\}$                              /* update the solution */
  $to\_be\_updated := \emptyset$                       /* elements in $PQ$ such that $E.w/E.n_c$ has changed */
  for each $t \in E.T$ do                              /* update $E.w$ due to targets */
    for each $E' \in (t.Att\_Ptr \setminus \{E\})$ do
      $E'.w := (E'.w) - w(t)$
      $E'.T := (E'.T) \setminus \{t\}$
      $to\_be\_updated := to\_be\_updated \cup \{E'\}$
  for each $c \in E.C$ do                              /* update $E.n_c$ due to satisfied constraints */
    for each $E' \in (c.Att\_Ptr \setminus \{E\})$ do
      $E'.n_c := (E'.n_c) - 1$
      $E'.C := (E'.C) \setminus \{c\}$
      $to\_be\_updated := to\_be\_updated \cup \{E'\}$
  for each $E' \in to\_be\_updated$ do                 /* update the priority queue */
    $PQ$ := Delete($PQ$,$E'$)
    $PQ$ := Insert($PQ$,$E'$)
  $E$ := Extract_Min($PQ$)
for each $a \in A'$ do                                 /* scan attributes in reverse order of insertion in $A'$ */
  if Can_Be_Removed($a$,$A'$,$\mathcal{C}$) then       /* check if $a$ is redundant*/
    $A' := A' \setminus \{a\}$
return($A'$)
```

Fig. 6. Algorithm that computes a solution to the WMTHSP

Our algorithm, reported in Fig. 6, takes as input a set $\mathcal{A}$ of attributes not appearing in singleton constraints, a well defined set $\mathcal{C}$ of constraints, a set $\mathcal{T}$ of targets, and a weight function w defined on $\mathcal{T}$, and returns a solution A', corresponding to the set of attributes composing, together with those appearing in singleton constraints, F_o.

The heuristic uses a priority-queue PQ that contains an element E for each attribute a to be considered. Each element E in PQ is a record with the following fields: $E.a$ is the attribute; $E.C$ is the set of pointers to non-satisfied constraints that contain $E.a$; $E.T$ is the set of pointers to the targets non intersecting the solution (i.e., targets with no attribute in the solution) that contain $E.a$; $E.n_c$ is the number of constraints pointed by $E.C$; and $E.w$ is the total weight of targets pointed by $E.T$ (i.e., $E.w = \sum_{t \in E.T} w(t)$). The priority of the elements in the queue is dictated by the value of the ratio $E.w/E.n_c$: elements with lower ratio have higher priority. The ratio $E.w/E.n_c$ reflects the relative cost of including an attribute in the solution, therefore obtained as weight to pay divided by number of constraints that would be solved by including the attribute. Each constraint $c \in \mathcal{C}$ (target $t \in \mathcal{T}$, resp.) is represented by a set $c.Att\_Ptr$ ($t.Att\_Ptr$, resp.) of pointers to the elements in PQ representing the attributes appearing in c (t, resp.). Therefore, there are double linking pointers between the elements in the priority queue and the constraints (and the targets, resp.). At initialization, the set of constraints and weighted targets are those given in input to the problem, the queue contains one element for each attribute to be fragmented, and the other fields of each queue element are calculated according to the input.

As an example, consider relation PATIENT and its confidentiality constraints in Fig. 1 and the data and query profile in Fig. 4. Figure 7 illustrates the initial

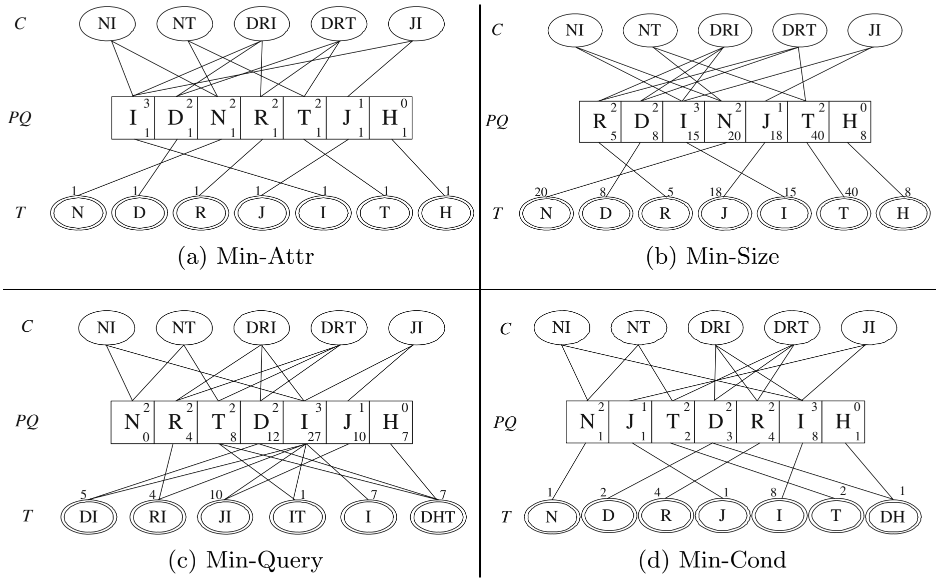


Fig. 7. Data structure initialization for the different problems

configurations of the data structures used by the algorithm, for the different minimization problems. In the figure, attributes are represented by their initials; constraints are represented as ovals; and targets as double-circled ovals, with their weight at the top. Each element E in the priority queue is represented with a box containing $E.a$, with $E.n_c$ and $E.w$ at the right-top and right-bottom corner of the box, respectively.

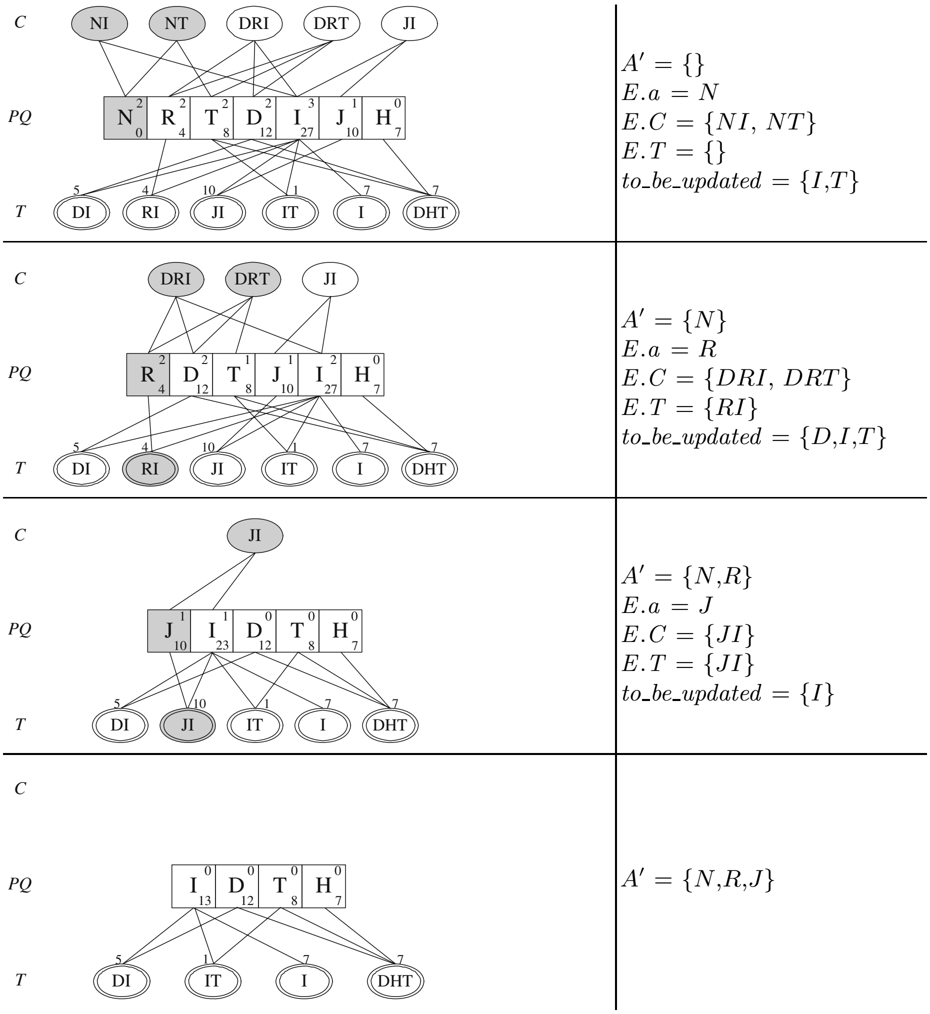
The algorithm performs a **while** loop that, at each iteration, extracts from the queue the element E with highest priority (lowest $E.w/E.n_c$ ratio), and inserts its attribute a into A'. Hence, for each constraint c pointed by $E.C$, it removes all pointers from/to c and elements in the priority queues, consequently adjusting the values of field n_c of all the involved elements. Analogously, for each target t pointed by $E.T$, it removes all pointers from/to t and elements in the priority queues, consequently adjusting the values of field w of all the involved elements. This update to the data structure reflects the fact that inclusion of a in the solution brings satisfaction of all the constraints in which a is involved (which therefore need not be considered anymore) and it carries the weight for all the targets that include a (which therefore need not be considered anymore). The **while** loop terminates if either the queue is empty (i.e., all attributes are in A') or all elements E in it have $E.n_c$=0 (i.e., all constraints have been solved). The set A' obtained at the end of the cycle, might be redundant (as the inclusion of a lower priority attribute might have made unnecessary the inclusion of an attribute, with higher priority, previously inserted in A'). Hence, the algorithm iteratively considers attributes in A' in reverse order of insertion and, for each considered attribute a, it determines if $A' \setminus \{a\}$ still represents a hitting set for $\mathcal{C}$. If it does, a is removed from A'. Note that considering the attributes in A' in reverse order of insertion corresponds to considering them in increasing order of priority. Note also that it is sufficient to check each attribute once (i.e., only a scan of A' needs to be performed).

The final fragmentation $\mathcal{F} = \langle F_o, F_s \rangle$ is then obtained by inserting in F_o, the union of A' with the attributes involved in singleton constraints (which are not considered in the algorithm and, consequently, are not in A'); and by setting $F_s = R \setminus F_o$.

The proposed heuristic algorithm has a polynomial time complexity, and computes a correct fragmentation. To prove its effectiveness, we have run experiments comparing the solutions returned by our heuristic with the optimal solution. We considered varying configurations, with different number of attributes, constraints, and queries. The heuristic algorithm produces solutions always close to the optimum (in many cases returning the optimum) and the maximum error observed is 14%. In terms of execution time, the heuristic algorithm considerably outperforms the exhaustive search. For all the runs, execution times remained below the measurement threshold of 1 *ms*, while the execution times of the exhaustive procedure increase exponentially, as expected.

Example 2. Figure 8 presents the execution, step by step, of the heuristic algorithm to solve problem Min-Query on relation PATIENT with its confidentiality constraints in Fig. 1 assuming the query profile in Fig. 4. The right hand side of Fig. 8 illustrates the evolution of solution A', the values of fields $E.a$, $E.C$, $E.T$, of the element E considered for each step, and the elements in the priority queue whose fields w and/or n_c must be changed (*to_be_updated*). The left hand side of the figure graphically illustrates the evolution of the data structure. At each step, the element with highest priority in the queue, together with the constraints and the targets pointed by it, are highlighted in gray. At the beginning, A' is empty, all constraints and targets need to be considered, and the priority queue is as reported in Fig. 7(c). The element with highest priority, with $E.a{=}N$, is extracted from the queue and placed into A'. Pointed constraints, $c_1{=}NI$ and $c_2{=}NT$, need not be considered anymore and therefore the pointers among them and the elements in the queue are removed, consequently updating field n_c for elements corresponding to attributes I and T, and therefore the priority of the elements in the queue. No update is needed for targets (as N was not involved in any). The subsequent steps proceed in analogous way extracting the elements corresponding to attributes R and J. Inclusion of J in the solution brings all values of n_c to 0; meaning that all constraints are satisfied and the algorithm ends. The computed solution $A' = \{N, R, J\}$ is minimal, since removing any attribute from it would not produce a hitting set. The resulting fragmentation, including in F_o the computed solution as well as all attributes appearing in singleton constraints is: F_o={`SSN`,`Name`,`Race`,`Job`}; F_s={`DoB`,`Illness`,`Treatment`,`HDate`}.

The execution of the algorithm for the other minimization problems returns:
Min-Attr: F_o={`SSN`,`Illness`,`Treatment`}, F_s={`Name`,`DoB`,`Race`,`Job`,`HDate`};
Min-Size: F_o={`SSN`,`Race`,`Illness`,`Name`}, F_s={`DoB`,`Job`,`Treatment`,`HDate`};
Min-Cond: F_o={`SSN`,`Name`,`Race`,`Job`}, F_s={`DoB`,`Illness`,`Treatment`,`HDate`}.



$A' = \{\}$
$E.a = N$
$E.C = \{NI, NT\}$
$E.T = \{\}$
$to\_be\_updated = \{I,T\}$

$A' = \{N\}$
$E.a = R$
$E.C = \{DRI, DRT\}$
$E.T = \{RI\}$
$to\_be\_updated = \{D,I,T\}$

$A' = \{N,R\}$
$E.a = J$
$E.C = \{JI\}$
$E.T = \{JI\}$
$to\_be\_updated = \{I\}$

$A' = \{N,R,J\}$

Fig. 8. An example of algorithm execution

6 Related Work

Previous work is related to the data outsourcing scenario [3,9,11,12,14], where the outsourced data are stored on an external honest-but-curious server and are entirely encrypted for confidentiality protection. Such approaches are typically based on the definition of additional indexing information, stored together with the encrypted data, which can be exploited for the evaluation of conditions at the server side. In [9,12], the authors address the problem of access control enforcement, proposing solutions based on selective encryption techniques for incorporating the access control policy in the data themselves.

The first proposal suggesting the combined use of fragmentation and encryption for enforcing confidentiality constraints has been presented in [1]. This technique is based on the assumption that data are split over two honest-but-curious servers and resorts to encryption any time two fragments are not sufficient for enforcing confidentiality constraints. This proposal also relies on the complete absence of communication between the two servers. The work presented in [4,5] removes this limiting assumption, by proposing a solution that allows storing multiple fragments on a single server and that minimizes the amount of data stored only in encrypted format or the query execution costs. 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 data owner site, thus avoiding the burden of decryption in query execution.

An affinity to the work presented in this paper can be found in [2,8]. 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 also addressed in the literature, with the main goal of improving query evaluation efficiency [13]. 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 confidentiality constraints in data outsourcing. Specifically, we were interested in analyzing the efficient management of data in the presence of a requirement forbidding the use of encryption on the data. The solution presented satisfies this requirement by exploiting the availability at the owner of local trusted storage, which will have to be used efficiently by limiting its use to the representation of the minimal collection of data that are needed to protect the specified confidentiality constraints. Minimization can be defined following several distinct criteria and we presented a general approach able to support, within the same algorithm, the evaluation of alternative metrics. It is to note that this approach in no way intends to make obsolete previous approaches using encryption. Rather, it proposes a novel way that extends the adoption of data outsourcing to scenarios where, in the evaluation of the tradeoff between the advantages and disadvantages of encryption, a strong preference is expressed toward the adoption of an encryption-less solution.

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.

References

1. Aggarwal, G., Bawa, M., Ganesan, P., Garcia-Molina, H., Kenthapadi, K., Motwani, R., Srivastava, U., Thomas, D., Xu, Y.: Two can keep a secret: a distributed architecture for secure database services. In: Proc. of CIDR 2005, Asilomar, CA, USA (January 2005)
2. Biskup, J., Embley, D., Lochner, J.: Reducing inference control to access control for normalized database schemas. IPL 106(1), 8–12 (2008)
3. Ceselli, A., Damiani, E., De Capitani di Vimercati, S., Jajodia, S., Paraboschi, S., Samarati, P.: Modeling and assessing inference exposure in encrypted databases. ACM TISSEC 8(1), 119–152 (February 2005)
4. Ciriani, V., De Capitani di Vimercati, S., Foresti, S., Jajodia, S., Paraboschi, S., Samarati, P.: Fragmentation and encryption to enforce privacy in data storage. In: Biskup, J., López, J. (eds.) ESORICS 2007. LNCS, vol. 4734, pp. 171–186. Springer, Heidelberg (2007)
5. Ciriani, V., De Capitani di Vimercati, S., Foresti, S., Jajodia, S., Paraboschi, S., Samarati, P.: Fragmentation design for efficient query execution over sensitive distributed databases. In: Proc. of ICDCS 2009, Montreal, Canada (June 2009)
6. Ciriani, V., De Capitani di Vimercati, S., Foresti, S., Samarati, P.: k-Anonymity. In: Yu, T., Jajodia, S. (eds.) Secure Data Management in Decentralized Systems. Springer, Heidelberg (2007)
7. Cormode, G., Srivastava, D., Yu, T., Zhang, Q.: Anonymizing bipartite graph data using safe groupings. In: Proc. of VLDB 2008, Auckland, New Zeland (August 2008)
8. Dawson, S., De Capitani di Vimercati, S., Lincoln, P., Samarati, P.: Maximizing sharing of protected information. JCSS 64(3), 496–541 (May 2002)
9. De Capitani di Vimercati, S., Foresti, S., Jajodia, S., Paraboschi, S., Samarati, P.: Over-encryption: Management of access control evolution on outsourced data. In: Proc. of VLDB 2007, Vienna, Austria (September 2007)
10. Garey, M., Johnson, D.: Computers and Intractability; a Guide to the Theory of NP-Completeness. W.H. Freeman and Company, New York (1979)
11. Hacigümüs, H., Iyer, B., Mehrotra, S.: Providing database as a service. In: Proc. of ICDE 2002, San Jose, CA, USA (February 2002)
12. Miklau, G., Suciu, D.: Controlling access to published data using cryptography. In: Proc. of VLDB 2003, Berlin, Germany (September 2003)
13. Navathe, S., Ceri, S., Wiederhold, G., Dou, J.: Vertical partitioning algorithms for database design. ACM TODS 9(4), 680–710 (December 1984)
14. Wang, H., Lakshmanan, L.V.S.: Efficient secure query evaluation over encrypted XML databases. In: Proc. of VLDB 2006, Seoul, Korea (September 2006)