Combining Fragmentation and Encryption to Protect Privacy in Data Storage

VALENTINA CIRIANI, SABRINA DE CAPITANI DI VIMERCATI, and SARA FORESTI
Università degli Studi di Milano
SUSHIL JAJODIA
George Mason University
STEFANO PARABOSCHI
Università degli Studi di Bergamo
and
PIERANGELA SAMARATI
Università degli Studi di Milano

The impact of privacy requirements in the development of modern applications is increasing very quickly. Many commercial and legal regulations are driving the need to develop reliable solutions for protecting sensitive information whenever it is stored, processed, or communicated to external parties. To this purpose, encryption techniques are currently used in many scenarios where data protection is required since they provide a layer of protection against the disclosure of personal information, which safeguards companies from the costs that may arise from exposing their data to privacy breaches. However, dealing with encrypted data may make query processing more expensive.

In this article, we address these issues by proposing a solution to enforce the privacy of data collections that combines data fragmentation with encryption. We model privacy requirements as confidentiality constraints expressing the sensitivity of attributes and their associations. We then

This article extends the previous work by the authors appeared under the title Fragmentation and Encryption to Enforce Privacy in Data Storage in *Proceedings of the 12th European Symposium On Research In Computer Security*, [Ciriani et al. 2007a].
This work was supported in part by the EU within the 7FP project under grant agreement 216483 "PrimeLife." The work of S. 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.
Authors' addresses: V. Ciriani, S. De Capitani di Vimercati, S. Foresti, P. Samarati, DTI, Università degli Studi di Milano, Via Bramante, 65, 26013 Crema, Italy, email: {firstname.lastname}@unimi.it; S. Jajodia, CSIS, George Mason University, Fairfax, VA 22030-4444, email: jajodia@gmu.edu; S. Paraboschi, DIIMM, Università degli Studi di Bergamo, Viale Marconi, 5, 24044 Dalmine, Italy; email: parabosc@unibg.it.

DOI 10.1145/1805974.1805978 http://doi.acm.org/10.1145/1805974.1805978

use encryption as an underlying (conveniently available) measure for making data unintelligible while exploiting fragmentation as a way to break sensitive associations among attributes. We formalize the problem of minimizing the impact of fragmentation in terms of number of fragments and their affinity and present two heuristic algorithms for solving such problems. We also discuss experimental results, comparing the solutions returned by our heuristics with respect to optimal solutions, which show that the heuristics, while guaranteeing a polynomial-time computation cost are able to retrieve solutions close to optimum.

Categories and Subject Descriptors: H.2.7 [**Database Management**]: Database Administration—*Security, integrity, and protection*; H.2.4 [**Database Management**]: Systems—*Relational databases*; H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval—*Query formulation*; K.6.5 [**Management of Computing and Information Systems**]: Security and Protection

General Terms: Security, Design

Additional Key Words and Phrases: Privacy, fragmentation, encryption

ACM Reference Format:
Ciriani, V., De Capitani di Vimercati, S., Foresti, S., Jajodia, S., Paraboschi, S., and Samarati, P. 2010. Combining fragmentation and encryption to protect privacy in data storage. ACM Trans. Info. Syst. Sec. 13, 3, Article 22 (July 2010), 33 pages.
DOI = 10.1145/1805974.1805978 http://doi.acm.org/10.1145/1805974.1805978

1. INTRODUCTION

Currently, information is probably the most important and valued resource. Private and governmental organizations are increasingly gathering and maintaining vast amounts of data, which often include sensitive personally identifiable information. In such a scenario, guaranteeing the privacy of the data, being stored in the system or communicated to external parties, becomes a primary requirement.

Individuals, privacy advocates, and legislators are currently putting more and more attention on the support of privacy over collected information. Regulations are increasingly being established to respond to these demands and force organizations to provide privacy guarantees over sensitive information when storing, processing, or sharing it with others. Most recent regulations (e.g., CA SB 1386 [2002] and Personal Data Protection Code [2003]), require that specific categories of data (e.g., data disclosing health and sex life, or data such as ZIP and date of birth that can be exploited to uniquely identify an individual [Samarati 2001]) be either encrypted or kept separate from other personally identifiable information (to prevent their association with specific individuals). Information privacy guarantees may also derive from the need of preventing possible abuses of critical information. For instance, the "Payment Card Industry (PCI) Data Security Standard" [PICDSS 2006] forces all the business organizations managing credit card information (e.g., VISA and MasterCard) to apply encryption measures when storing data. The standard also explicitly forbids the use of storage encryption as natively offered by operating systems, requiring that access to the encryption keys be separated from the operating system services managing user identities and privileges.

Currently, this demand for encryption is luckily coupled with the fact that the realization of cryptographic functions presents increasingly lower costs

in a computer architecture, where the factor limiting system performances is typically the capacity of the channels that transfer information within the system and among separate systems. Cryptography then becomes an inexpensive tool that supports the protection of privacy when storing or communicating information.

From a data access point-of-view, however, dealing with encrypted information represents a burden since encryption makes it not always possible to efficiently execute queries and evaluate conditions over the data. As a matter of fact, a straightforward approach to guarantee privacy to a collection of data could consist in encrypting all the data. This technique is, for example, adopted in the database outsourcing scenario [Damiani et al. 2003; Hacigümüs et al. 2002a], where a protective layer of encryption is wrapped around sensitive data, thus counteracting outside attacks as well as the curiosity from the server itself. The assumption underlying approaches applying such an encryption wrapper is that all the data are equally sensitive; therefore, encryption is a price to be paid to protect them. This assumption is typically an overkill in many situations where data are not sensitive per se; what is sensitive is their association with other data. As a simple example, in a hospital, the list of illnesses cured or the list of patients could be made publicly available, while the association of specific illnesses to individual patients is sensitive and must be protected. Hence, there is no need to encrypt both illnesses and patients if there are alternative ways to protect the association between them.

In this article, we propose an approach that couples encryption together with data fragmentation. We apply encryption only when explicitly demanded by the privacy requirements. The combined use of encryption and data fragmentation has first been proposed in the context of data outsourcing [Aggarwal et al. 2005]. In that proposal, privacy requirements are enforced by splitting information over two independent database servers (so to break associations of sensitive information) and by encrypting information whenever necessary. While presenting an interesting idea, the approach in Aggarwal et al. [2005] suffers from several limitations. The main limitation is that privacy relies on the complete absence of communication between the two servers, which have to be completely unaware of each other. This assumption is clearly too strong and difficult to enforce in real environments. A collusion among the servers (or the users accessing them) easily breaches privacy. Also, the assumption of two servers limits the number of associations that can be solved by fragmenting data, often forcing the use of encryption.

Our solution, sketched in Figure 1, overcomes the previously described limitations as follows. The information to be protected is first split into different fragments (i.e., different pieces of information) in such a way to break the sensitive associations represented through confidentiality constraints and to minimize the amount of information represented only in encrypted format. The resulting fragments may be stored at the same server or at different servers. Finally, the encryption key is given to the authorized users needing to access the information. Users that do not know the encryption key as well as the storing server(s) are unable to access sensitive information or to reconstruct the sensitive associations.

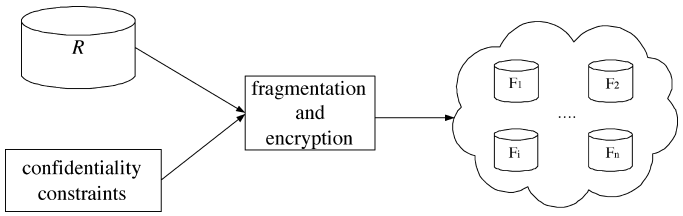

Fig. 1. Data protection scenario.

We frame our work in the context of relational databases, since they are by far the most common solution for the management of the data subject of privacy regulations. Also, they are characterized by a clear data model and a simple query language that facilitate the design of a solution. We note, however, that our model could be easily adapted to the protection of data represented with other data models (e.g., records in files or XML documents).

The contribution of this article can be summarized as follows. First, we introduce confidentiality constraints as a simple, yet powerful, way to capture privacy requirements (Section 2). Second, we provide a model formalizing the application of data fragmentation and encryption, which captures properties related to the correct representation of the data while minimizing encryption and fragmentation (Sections 3 and 4). Third, we propose two heuristic algorithms for the concrete identification of a fragmentation that satisfies the properties specified. In particular, the first algorithm computes a solution in such a way to avoid an excessive fragmentation of the data, that is, to limit the number of fragments (Section 5). The second algorithm is based on the definition of affinity between attributes and computes a fragmentation that exhibits good affinity value (Sections 6 and 7). Fourth, we illustrate how queries formulated on the original data are mapped into equivalent queries operating on fragments (Section 8). Fifth, to empirically verify the soundness of the technique proposed, we implement the two heuristic algorithms and compare the computed results in terms of both quality of the solution obtained and computational time required (Section 9).

2. CONFIDENTIALITY CONSTRAINTS

We model, in a quite simple and powerful way, the privacy requirements through *confidentiality constraints*, which are sets of attributes, as follows.

Definition 2.1 (*Confidentiality Constraint*). Let $\mathcal{A}$ be a set of attributes, a *confidentiality constraint* is a subset $c \subseteq \mathcal{A}$.

The semantics of a confidentiality constraint c is that the (joint) visibility of values of all the attributes in c should be protected, that is, the association between all their values should not be disclosed (e.g., triples {`DoB`,`ZIP`,`Illness`} are considered sensitive and cannot be released). Disjoint subsets of the attributes, unless protected by other constraints, can be released (e.g., pair {`DoB`,`ZIP`} and individual illness's values can be separately released). When the constraint is a singleton set, the semantics is that the individual attribute must be protected,

that is, the list of the attribute values itself is confidential (e.g., phone numbers or e-mail addresses can be considered sensitive values even if not associated with any identifying information).

While simple, the previously described definition allows the expression of the different confidentiality requirements that may need to be considered. Note that constraints specified on the association among attributes can derive from different requirements: They can correspond to explicit protection of an association (as in the case of names and illnesses described earlier) or to associations that could cause inference on other sensitive information. As an example of the latter, suppose that the names of patients are considered sensitive and, therefore, cannot be stored in the clear, and that the association of `DoB` together with the `ZIP` code can work as a quasi-identifier [Ciriani et al. 2007b; Samarati 2001] (i.e., `DoB` and `ZIP` can be used, possibly in association with external information, to help identifying patients and, therefore, to infer, or reduce uncertainty about, their names). This inference channel can be simply blocked by specifying a constraint protecting the association of `DoB` with the `ZIP` code. As another example, consider the case where attribute `Name` is not considered sensitive, but its association with `Illness` is. Suppose again that `DoB` together with the `ZIP` code can work as a quasi-identifier, that is, linking them with external information one can restrict uncertainty on the corresponding patients' names. In this case, an association constraint will be specified protecting the association among `DoB`, `ZIP`, and `Illness`, implying that the three attributes should never be accessible together in the clear. The definition of confidentiality constraints is then a complex problem that should take into account the different relationships among attributes. Such a problem is, however, outside the scope of this article, and we assume that the data owner correctly defines confidentiality constraints.

We are interested in enforcing a set of well-defined confidentiality constraints, formally defined as follows.

Definition 2.2 (*Well-Defined Constraints*). A set of confidentiality constraints $\mathcal{C} = \{c_1, \ldots, c_m\}$ is said to be *well-defined* if and only if $\forall c_i, c_j \in \mathcal{C}, i \neq j$, $c_i \not\subset c_j$.

According to this definition, a set of constraints $\mathcal{C}$ over $\mathcal{A}$ cannot contain a constraint that is a subset of another constraint. The rationale behind this property is that whenever there are two constraints c_i, c_j and c_i is a subset of c_j, the satisfaction of constraint c_i implies the satisfaction of constraint c_j (see Section 3) and, therefore, c_j is redundant.

To model the problem of enforcing a set of well-defined confidentiality constraints, we assume standard notations from the relational database model. Formally, let $\mathcal{A}$ be a set of attributes and $\mathcal{D}$ be a set of domains. A relation schema R is a finite set of attributes $\{a_1, \ldots, a_n\} \subseteq \mathcal{A}$, where each a_i is defined on a domain $D_i \in \mathcal{D}, i = 1, \ldots, n$. Notation $R(a_1, \ldots, a_n)$ represents a relation schema R over the set $\{a_1, \ldots, a_n\}$ of attributes. A tuple t over a set of attributes $\{a_1, \ldots, a_n\}$ is a function that associates with each attribute a_i a value $v \in D_i$. Notation $t[a]$ denotes value v associated with attribute a in t. A relation

MEDICALDATA

SSN	Name	DoB	ZIP	Illness	Physician
123-45-6789	A. Hellman	81/01/03	94142	hypertension	M. White
987-65-4321	B. Dooley	53/10/07	94141	obesity	D. Warren
246-89-1357	C. McKinley	52/02/12	94139	hypertension	M. White
135-79-2468	D. Ripley	81/01/03	94139	obesity	D. Warren

(a)

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

(b)

Fig. 2. An example of plaintext relation (a) and its well-defined constraints (b).

r over relation schema $R(a_1, \ldots, a_n)$ is a set of tuples over the set of attributes $\{a_1, \ldots, a_n\}$. In the following, when clear from the context, we will use R to denote either the relation schema R or the set of attributes in R.

For simplicity and consistently with other proposals [Aggarwal et al. 2005; Samarati 2001], we consider a single relation r, over a relation schema $R(a_1, \ldots, a_n)$, containing all the sensitive information that needs to be protected.

Example 2.3. Figure 2 illustrates an example of relation together with some confidentiality constraints on it: c_0 states that the list of SSNs of patients is considered sensitive; $c_1, \ldots, c_4$ state that the association of patients' names with any other piece of stored information is considered sensitive; c_5 and c_6 state that DoB and ZIP together can be exploited to infer the identity of patients (i.e., they can work as a quasi-identifier), consequently their association with other pieces of information is considered sensitive.

Note also that the association of patients' Name and SSN is sensitive and should be protected. However, such a constraint is not specified, since it is redundant, given that SSN by itself has been declared sensitive (c_0): protecting SSN as an individual attribute implies automatic protection of its associations with any other attribute.

3. FRAGMENTATION AND ENCRYPTION FOR CONSTRAINT SATISFACTION

Our approach to satisfy confidentiality constraints is based on the use of two techniques: encryption and fragmentation. Consistently with how the constraints are specified, encryption applies at the attribute level, that is, it involves an attribute in its entirety. Encrypting an attribute means encrypting (tuple by tuple) all its values. To protect encrypted values from frequency attacks [Schneier 1996], we assume that a *salt*, which is a randomly chosen value, is applied to each encryption (similarly to the use of nonces in the protection of messages from replay attacks). Fragmentation, like encryption, applies at the attribute level, that is, it involves an attribute in its entirety. Fragmenting means splitting sets of attributes so that they are not visible together, that is, the associations among their values are not available without access to the encryption key.

It is straightforward to see that singleton constraints can be solved only by encryption. By contrast, an association constraint could be solved by either: (i) encrypting any (one suffices) of the attributes involved in the constraint,

f_1^e

salt	enc	Name
s_1	α	A. Hellman
s_2	β	B. Dooley
s_3	γ	C. McKinley
s_4	δ	D. Ripley

(a)

f_2^e

salt	enc	DoB	ZIP
s_5	ε	81/01/03	94142
s_6	ζ	53/10/07	94141
s_7	η	52/02/12	94139
s_8	θ	81/01/03	94139

(b)

f_3^e

salt	enc	Illness	Physician
s_9	ι	hypertension	M. White
s_{10}	κ	obesity	D. Warren
s_{11}	λ	hypertension	M. White
s_{12}	μ	obesity	D. Warren

(c)

Fig. 3. An example of physical fragments for the relation in Figure 2(a).

so to prevent joint visibility, or (ii) fragmenting the attributes involved in the constraint so that they are not visible together. In the following, we use the term *fragment* to denote any subset of a given set of attributes. A fragmentation is a set of fragments, as captured by the following definition.

Definition 3.1 (*Fragmentation*). Let R be a relation schema, a *fragmentation* of R is a set of fragments $\mathcal{F} = \{F_1, \ldots, F_m\}$, where $F_i \subseteq R$, $i = 1, \ldots, m$.

For instance, with respect to the plaintext relation in Figure 2(a), a possible fragmentation is $\mathcal{F} = \{\{\texttt{Name}\},\{\texttt{DoB,ZIP}\},\{\texttt{Illness,Physician}\}\}$.

At the physical level, a fragmentation translates to a combination of fragmentation and encryption. Each fragment F is mapped into a physical fragment containing all the attributes of F in the clear, while all the other attributes of R are encrypted. The reason for reporting all the original attributes (in either encrypted or clear form) in each of the physical fragments is to guarantee that any query can be executed by querying a single physical fragment (see Section 8). For the sake of simplicity and efficiency, we assume that all the attributes not appearing in the clear in a fragment are encrypted all together (encryption is applied on subtuples). Physical fragments are then defined as follows.

Definition 3.2 (*Physical Fragment*). Let R be a relation schema and $\mathcal{F} = \{F_1, \ldots, F_m\}$ be a fragmentation of R. For each $F_i = \{a_{i_1}, \ldots, a_{i_n}\} \in \mathcal{F}$, the *physical fragment* of R over F_i is a relation schema $F_i^e(\underline{salt}, enc, a_{i_1}, \ldots, a_{i_n})$, where *salt* is the primary key, *enc* represents the encryption of all the attributes of R that do not belong to the fragment, combined before encryption in a binary XOR (symbol $\oplus$) with the salt.

At the level of instance, given a fragment $F_i = \{a_{i_1}, \ldots, a_{i_n}\}$ and a relation r over schema R, the physical fragment F_i^e of F_i is such that each plaintext tuple $t \in r$ is mapped into a tuple $t^e \in f_i^e$ where f_i^e is a relation over F_i^e and:

—$t^e[enc] = E_k(t[R - F_i] \oplus t^e[salt])$
—$t^e[a_{i_j}] = t[a_{i_j}]$, $j = 1, \ldots, n$

Figure 3 illustrates an example of physical fragments for the relation schema in Figure 2(a) that correctly enforce the well-defined constraints in Figure 2(b).

The algorithm in Figure 4 shows the construction and population of physical fragments to be executed at initialization. When the size of the attributes exceeds the size of an encryption block, we assume that the encryption of the protected attributes is performed by applying the Cipher Block Chaining (CBC) mode [Schneier 1996], with the salt used as the Initialization Vector (IV), or,

INPUT
A relation r over schema R
$\mathcal{C} = \{c_1, \ldots, c_m\}$ /* well-defined constraints */
OUTPUT
A set of physical fragments $\{F^e_1, \ldots, F^e_i\}$
A set of relations $\{f^e_1, \ldots, f^e_i\}$ over schemas $\{F^e_1, \ldots, F^e_i\}$

MAIN
$\mathcal{C}_\mathcal{F} := \{c \in \mathcal{C} : |c| > 1\}$ /* association constraints */
$\mathcal{A}_\mathcal{F} := \{a \in R\colon \{a\} \notin \mathcal{C}\}$
$\mathcal{F} :=$ **fragment**$(\mathcal{A}_\mathcal{F}, \mathcal{C}_\mathcal{F})$ /* function **fragment** computes a correct fragmentation $\mathcal{F}$ of R*/
for each $F = \{a_{i_1}, \ldots, a_{i_l}\} \in \mathcal{F}$ **do** /* define physical fragments */
  define relation F^e with schema: $F^e(\underline{salt}, enc, a_{i_1}, \ldots, a_{i_l})$
  for each $t \in r$ **do** /* populate physical fragments instances */
    $t^e[salt] :=$ **generatesalt**(F, t)
    $t^e[enc] := E_k(t[a_{j_1} \ldots a_{j_p}] \oplus t^e[salt])$ /* $\{a_{j_1}, \ldots, a_{j_p}\} = R - F$ */
    for each $a \in F$ **do** $t^e[a] := t[a]$
    insert t^e in f^e

Fig. 4. Algorithm that correctly fragments R and populates the corresponding physical fragments.

alternatively, we can use a semantically secure encryption function. In the CBC mode, the clear text of the first block is encrypted after it has been combined in binary XOR with the IV.

Note that the salts, which we conveniently use as primary keys of physical fragments (ensuring no collision in their generation), need not be secret because knowledge of the salts does not help in attacking the encrypted values as long as the encryption algorithm is secure and the key remains protected.

Given a relation r over schema R and a set of confidentiality constraints $\mathcal{C}$ on it, our goal is to produce a fragmentation that satisfies the constraints. However, we must also ensure that no constraint can be violated by recombining two or more fragments. In other words, there cannot be attributes that can be exploited for linking. Since encryption is differentiated by the use of the salt, the only attributes that can be exploited for linking are the plaintext attributes. Consequently, ensuring that fragments are protected from linking translates into requiring that no attribute appears in clear form in more than one fragment, as formally captured by the following definition.

Definition 3.3 (*Fragmentation Correctness*). Let R be a relation schema, $\mathcal{F}$ be a fragmentation of R, and $\mathcal{C}$ be a set of well-defined constraints over R. $\mathcal{F}$ *correctly enforces* $\mathcal{C}$ if and only if the following conditions are satisfied.

(1) $\forall F \in \mathcal{F}, \forall c \in \mathcal{C} : c \not\subseteq F$ (each individual fragment satisfies the constraints);
(2) $\forall F_i, F_j \in \mathcal{F}, i \neq j : F_i \cap F_j = \emptyset$ (fragments do not have attributes in common).

Note that Condition 1, requiring fragments not to be a superset of any constraint, implies that attributes appearing in singleton constraints do not appear in any fragment (i.e., as already noted, they appear only in encrypted form).

4. MINIMAL FRAGMENTATION

The availability of plaintext attributes in a fragment permits an efficient execution of queries. Therefore, we aim at minimizing the number of attributes that are not represented in the clear in any fragment because queries using those attributes will be generally processed inefficiently. In other words, we prefer fragmentation over encryption whenever possible and always solve association constraints via fragmentation.

The requirement on the availability of a plain representation for the maximum number of attributes can be captured by imposing that any attribute not involved in a singleton constraint must appear in the clear in at least one fragment. This requirement is formally represented by the definition of maximal visibility as follows.

Definition 4.1 (*Maximal Visibility*). Let R be a relation schema and $\mathcal{C}$ be a set of well-defined constraints. A fragmentation $\mathcal{F}$ of R *maximizes visibility* if and only if $\forall a \in R, \{a\} \notin \mathcal{C}$: $\exists F \in \mathcal{F}$ such that $a \in F$.

Note that the combination of maximal visibility together with the second condition of Definition 3.3 imposes that each attribute that does not appear in a singleton constraint must appear in the clear in exactly one fragment (i.e., at least for Definition 4.1, at most for Definition 3.3).

Another important aspect to consider when fragmenting a relation to satisfy a set of constraints is to avoid excessive fragmentation. As a matter of fact, the availability of more attributes in the clear in a single fragment allows a more efficient execution of queries on the fragment. Indeed, a straightforward approach for producing a fragmentation that satisfies the constraints while maximizing visibility is to define as many (singleton) fragments as the number of attributes not appearing in singleton constraints. Such a solution, unless demanded by the constraints, is, however, undesirable because it makes any query involving conditions on more than one attribute inefficient.

We are interested in finding a fragmentation that makes query execution efficient. A simple strategy to achieve this goal consists in finding a *minimal fragmentation*, that is, a correct fragmentation that maximizes visibility, while minimizing the number of fragments. This problem can be formalized as follows.

Problem 4.2 (*Minimal Fragmentation*). Given a relation schema R, a set $\mathcal{C}$ of well-defined constraints over R, find a *fragmentation* $\mathcal{F}$ of R such that all the following conditions hold.

(1) $\mathcal{F}$ correctly enforces $\mathcal{C}$ (Definition 3.3).
(2) $\mathcal{F}$ maximizes visibility (Definition 4.1).
(3) $\nexists \mathcal{F}'$ satisfying the first two conditions such that the number of fragments of $\mathcal{F}'$ is less than the number of fragments of $\mathcal{F}$.

The minimal fragmentation problem is *NP-hard*, as formally stated by the following theorem

THEOREM 4.3. *The minimal fragmentation problem is NP-hard.*

PROOF. The proof is a reduction from the NP-hard problem of minimum hypergraph coloring [Garey and Johnson 1979], which can be formulated as follows: given a hypergraph $\mathcal{H}(V, E)$, determine a minimum coloring of $\mathcal{H}$, that is, assign to each vertex in V a color such that adjacent vertices have different colors, and the number of colors is minimized.

Given a relation schema R and a set $\mathcal{C}$ of well-defined constraints, the correspondence between the minimal fragmentation problem and the hypergraph coloring problem can be defined as follows. Any vertex v_i of the hypergraph $\mathcal{H}$ corresponds to an attribute $a_i \in R$ such that $\{a_i\} \notin \mathcal{C}$. Any edge e_i in $\mathcal{H}$, which connects $v_{i_1}, \ldots, v_{i_c}$, corresponds to a constraint $c_i = \{a_{i_1}, \ldots, a_{i_c}\} \in \mathcal{C}$ and c_i is not a singleton constraint. A fragmentation $\mathcal{F} = \{F_1(a_{1_1}, \ldots, a_{1_k}), \ldots, F_p(a_{p_1}, \ldots, a_{p_l})\}$ of R satisfying all constraints in $\mathcal{C}$ corresponds to a solution S for the corresponding hypergraph coloring problem. Specifically, S uses p colors in such a way that all vertices corresponding to attributes in F_i are colored with the i-th color, for $i = 1, \ldots, p$. As a consequence, any algorithm finding a minimal fragmentation can be exploited to solve the hypergraph coloring problem. □

The hypergraph coloring problem has been extensively studied in the literature, reaching interesting theoretical results. In particular, assuming $NP \neq ZPP$, there are no polynomial time approximation algorithms for coloring k-uniform hypergraphs with approximation ratio $O(n^{1-\epsilon})$ for any fixed $\epsilon > 0$ [Krivelevich and Sudakov 2003; Hofmeister and Lefmann 1998].[1] In the next section, we present a heuristic for solving Problem 4.2. The heuristic is based on the definition of *vector minimality*, which is then exploited to efficiently find a correct fragmentation maximizing visibility.

5. A HEURISTIC APPROACH TO MINIMIZE FRAGMENTATION

We first characterize the set of all possible fragmentations by defining a dominance relationship among them and by introducing the definition of vector-minimal fragmentation. We then describe a heuristic algorithm for Problem 4.2 that computes a vector-minimal fragmentation.

5.1 Vector-Minimal Fragmentation

To formally define the vector-minimal fragmentation, we first introduce the concept of fragment vector as follows.

Definition 5.1 (*Fragment Vector*). Let R be a relation schema, and $\mathcal{F} = \{F_1, \ldots, F_m\}$ be a correct fragmentation of R. The *fragment vector* $V_\mathcal{F}$ of $\mathcal{F}$ is a vector of fragments with an element $V_\mathcal{F}[a]$ for each $a \in \bigcup_{i=1}^{m} F_i$, where the value of $V_\mathcal{F}[a]$ is the unique fragment $F_j \in \mathcal{F}$ containing attribute a.

[1] In a minimization framework, an approximation algorithm with approximation ratio p guarantees that the cost C of its solution is such that $C/C^* \leq p$, where C^* is the cost of an optimal solution [Garey and Johnson 1979]. On the contrary, we cannot perform any evaluation on the result of a heuristic.

Example 5.2. Let $\mathcal{F} = \{\{\texttt{Name}\}, \{\texttt{DoB,ZIP}\}, \{\texttt{Illness,Physician}\}\}$ be a fragmentation of the relation schema in Figure 2(a). The fragment vector is the vector $V_{\mathcal{F}}$ such that:

—$V_{\mathcal{F}}[\texttt{Name}] = \{\texttt{Name}\}$;
—$V_{\mathcal{F}}[\texttt{DoB}] = V_{\mathcal{F}}[\texttt{ZIP}] = \{\texttt{DoB,ZIP}\}$;
—$V_{\mathcal{F}}[\texttt{Illness}] = V_{\mathcal{F}}[\texttt{Physician}] = \{\texttt{Illness,Physician}\}$.

Fragment vectors can be exploited to define a partial order between fragmentations as follows.

Definition 5.3 (*Dominance*). Let R be a relation schema, and $\mathcal{F}$ and $\mathcal{F}'$ be two fragmentations of R maximizing visibility. Let $\mathcal{A}$ be the (equal) set of attributes in the two fragmentations. We say that $\mathcal{F}'$ *dominates* $\mathcal{F}$, denoted $\mathcal{F} \preceq \mathcal{F}'$, if and only if $V_{\mathcal{F}}[a] \subseteq V_{\mathcal{F}'}[a]$, for all $a \in \mathcal{A}$. We say $\mathcal{F} \prec \mathcal{F}'$ if and only if $\mathcal{F} \preceq \mathcal{F}'$ and $\mathcal{F} \neq \mathcal{F}'$.

Definition 5.3 states that fragmentation $\mathcal{F}'$ dominates fragmentation $\mathcal{F}$ if $\mathcal{F}'$ can be computed from $\mathcal{F}$ by merging two (or more) fragments composing $\mathcal{F}$.

Example 5.4. Let $\mathcal{F}_1 = \{\{\texttt{Name}\}, \{\texttt{DoB,ZIP}\}, \{\texttt{Illness,Physician}\}\}$ and $\mathcal{F}_2 = \{\{\texttt{Name}\}, \{\texttt{DoB}\}, \{\texttt{ZIP}\}, \{\texttt{Illness,Physician}\}\}$ be two fragmentations of the relation schema in Figure 2(a). Since $\mathcal{F}_1$ can be obtained from $\mathcal{F}_2$ by merging fragments $\{\texttt{DoB}\}$ and $\{\texttt{ZIP}\}$, it results that $\mathcal{F}_2 \prec \mathcal{F}_1$ (see Definition 5.3).

We can formally define a *vector-minimal fragmentation* as a fragmentation $\mathcal{F}$ such that it is correct, it maximizes visibility, and all fragmentations that can be obtained from $\mathcal{F}$ by merging any two fragments in $\mathcal{F}$ violate at least one constraint.

Definition 5.5 (*Vector-Minimal Fragmentation*). Let R be a relation schema, $\mathcal{C}$ be a set of well-defined constraints, and $\mathcal{F}$ be a fragmentation of R. $\mathcal{F}$ is a *vector-minimal fragmentation* if and only if all the following conditions are satisfied.

(1) $\mathcal{F}$ correctly enforces $\mathcal{C}$ (Definition 3.3).
(2) $\mathcal{F}$ maximizes visibility (Definition 4.1).
(3) $\nexists \mathcal{F}'$ satisfying the first two conditions such that $\mathcal{F} \prec \mathcal{F}'$.

According to this definition of minimality, it easy to see that while a minimal fragmentation is also vector-minimal, the vice versa is not necessarily true.

Example 5.6. Consider fragmentations $\mathcal{F}_1$ and $\mathcal{F}_2$ of Example 5.4, and the set of constraints in Figure 2(b). Since $\mathcal{F}_2 \prec \mathcal{F}_1$, $\mathcal{F}_2$ is not vector-minimal. By contrast, $\mathcal{F}_1$ is vector-minimal. As a matter of fact, $\mathcal{F}_1$ contains all attributes of relation schema MEDICALDATA in Figure 2(a) except SSN (maximal visibility); satisfies all constraints in Figure 2(b) (correctness); no fragmentation obtained from it by merging any pair of fragments satisfies the constraints.

FUNCTION 5.7 VECTOR-MINIMAL FRAGMENTATION.

```
FRAGMENT(A_ToPlace,C_ToSolve)
F := ∅
for each a∈A_ToPlace do /* initialize arrays Con[] and N_con[] */
  Con[a] := {c ∈ C_ToSolve| a ∈ c}
  N_con[a] := |Con[a]|
repeat
  if C_ToSolve ≠ ∅ then
    let attr be an attribute with the maximum value of N_con[]
    for each c ∈ (Con[attr] ∩ C_ToSolve) do
      C_ToSolve := C_ToSolve − {c} /* adjust the constraints */
      for each a ∈ c do N_con[a] := N_con[a]−1 /* adjust array N_con[] */
  else /* since all the constrains are satisfied, choose any attribute in A_ToPlace */
    let attr be an attribute in A_ToPlace
  endif
  A_ToPlace := A_ToPlace − {attr}
  inserted := false /* try to insert attr into the existing fragments */
  for each F ∈ F do /* evaluate if F ∪ {attr} satisfies the constraints */
    satisfies := true
    for each c ∈ Con[attr] do
      if c ⊆ (F ∪ {attr}) then
        satisfies := false /* choose the next fragment */
        break
      endif
    if satisfies then
      F := F ∪ {attr} /* attr has been inserted into F */
      inserted := true
      break
    endif
  if NOT inserted then /* insert attr into a new fragment */
    add {attr} to F
  endif
until A_ToPlace = ∅
return(F)
```

Fig. 5. Function that finds a vector-minimal fragmentation.

5.2 Function Fragment for Computing a Vector-Minimal Fragmentation

The definition of vector-minimal fragmentation allows us to design a heuristic approach for Problem 4.2 that works in polynomial time and computes a fragmentation that even if it is not necessarily a minimal fragmentation, it is, however, near to the optimal solution, as the experimental results show (see Section 9).

Our heuristic method starts with an empty fragmentation and, at each step, selects the attribute involved in the highest number of unsolved constraints. The rationale behind this selection criterion is to bring all constraints to satisfaction in a few steps. The selected attribute is then inserted into a fragment that is determined in such a way that there is no violation of the constraints involving the attribute. If such a fragment does not exist, a new fragment for the selected attribute is created. The process terminates when all attributes have been inserted into a fragment. Figure 5 illustrates function **fragment** that implements this heuristic method. As input, the function takes a set *A_ToPlace* of attributes to be fragmented and a set *C_ToSolve* of well-defined nonsingleton constraints. It computes a vector-minimal fragmentation $\mathcal{F}$ of *A_ToPlace* as follows.

First, the function initializes $\mathcal{F}$ to the empty set and creates two arrays *Con*[] and *N_con*[] that contain an element for each attribute a in *A_ToPlace*. Element *Con*[a] contains the set of constraints on a, and element *N_con*[a] is the number of nonsolved constraints involving a (note that, at the beginning, *N_con*[a] coincides with the cardinality of *Con*[a]). The function then executes a **repeat-until** loop that, at each iteration, places an attribute *attr* into a fragment as follows. If there are constraints still to be solved (*C_ToSolve* $\neq \emptyset$), *attr* is selected as an attribute appearing in the highest number of unsolved constraints. Then, for each constraint c in *Con*[*attr*] $\cap$ *C_ToSolve*, the function removes c from *C_ToSolve* and, for each attribute a in c, decreases *N_con*[a] by one. Otherwise, that is, all constraints are solved (*C_ToSolve* $= \emptyset$), the function chooses *attr* by randomly extracting an attribute from *A_ToPlace*. Then, the function removes *attr* from *A_ToPlace* and looks for a fragment F in $\mathcal{F}$ in which *attr* can be inserted without violating any constraint, including *attr*. If such a fragment F is found, *attr* is inserted into F, otherwise a new fragment $\{attr\}$ is added to $\mathcal{F}$. Note that the search for a fragment terminates as soon as a fragment is found (*inserted* $=$ *true*). Also, the control on constraint satisfaction terminates as soon as a violation to constraints is found (*satisfies* $=$ *false*).

Example 5.8. Figure 6 presents the execution, step by step, of function **fragment** applied to the example in Figure 2. Here, for simplicity, we represent attributes with their initials. The left-hand side of Figure 6 illustrates the evolution of variables *attr*, $\mathcal{F}$, *C_ToSolve*, and *A_ToPlace*, while the right-hand side graphically illustrates the same information through a matrix with a row for each attribute and a column for each constraint. If an attribute belongs to an unsolved constraint c_i, the corresponding cell is set to $\times$; otherwise, if c_i is solved, the cell is set to $\checkmark$. At the beginning, $\mathcal{F}$ is empty, all constraints are unsolved, and all attributes need to be placed. In the first iteration, function **fragment** chooses attribute n, since it is the attribute involved in the highest number of unsolved constraints. The constraints in *Con*[n] become now solved, array *N_con*[] is updated accordingly, and fragment $\{n\}$ is added to $\mathcal{F}$. Function **fragment** proceeds in an analogous way by choosing attributes d, z, i, and p. The final solution is represented by the relations in Figure 3.

5.3 Correctness and Complexity

The correctness and complexity of our approach are stated by the following theorems.

THEOREM 5.9 (CORRECTNESS). *Function* **fragment** *in Figure 5 terminates and finds a vector-minimal fragmentation* (*Definition 5.5*).

PROOF. Given a relation schema R, and a set $\mathcal{C}$, of well-defined constraints, $\mathcal{C}_F = \{c \in \mathcal{C} : |c| > 1\}$ and $\mathcal{A}_F = \{a \in R\colon \{a\} \notin \mathcal{C}\}$.

Function 5.7 terminates since at each iteration of the **repeat-until** loop an attribute is extracted from *A_ToPlace*, which is initialized to $\mathcal{A}_F$, and the loop is performed until *A_ToPlace* is not empty.

We now prove that a solution $\mathcal{F}$ computed by this function over $\mathcal{A}_F$ and $\mathcal{C}_F$ is a vector-minimal fragmentation. According to Definition 5.5, a fragmentation

$\mathcal{F}$=∅
$C\_ToSolve$={c_1,c_2,c_3,c_4,c_5,c_6}
$A\_ToPlace$={n,d,z,i,p}

	c_1	c_2	c_3	c_4	c_5	c_6	$N\_con[a_i]$
n	×	×	×	×			4
d	×				×	×	3
z		×			×	×	3
i			×		×		2
p				×		×	2
$ToSolve$	yes	yes	yes	yes	yes	yes	

$attr = n$
$Con[n]$={c_1,c_2,c_3,c_4}
$\mathcal{F}$ = {{n}}
$C\_ToSolve$ = {c_5,c_6}
$A\_ToPlace$ = {d,z,i,p}

	c_1	c_2	c_3	c_4	c_5	c_6	$N\_con[a_i]$
n	✓	✓	✓	✓			0
d	✓				×	×	2
z		✓			×	×	2
i			✓		×		1
p				✓		×	1
$ToSolve$	✓	✓	✓	✓	yes	yes	

$attr = d$
$Con[d]$={c_1,c_5,c_6}
$\mathcal{F}$ = {{n},{d}}
$C\_ToSolve$ = ∅
$A\_ToPlace$ = {z,i,p}

	c_1	c_2	c_3	c_4	c_5	c_6	$N\_con[a_i]$
n	✓	✓	✓	✓			0
d	✓				✓	✓	0
z		✓			✓	✓	0
i			✓		✓		0
p				✓		✓	0
$ToSolve$	✓	✓	✓	✓	✓	✓	

$attr = z$
$Con[z]$={c_2,c_5,c_6}
$\mathcal{F}$ = {{n},{d,z}}
$C\_ToSolve$ = ∅
$A\_ToPlace$ = {i,p}

	c_1	c_2	c_3	c_4	c_5	c_6	$N\_con[a_i]$
n	✓	✓	✓	✓			0
d	✓				✓	✓	0
z		✓			✓	✓	0
i			✓		✓		0
p				✓		✓	0
$ToSolve$	✓	✓	✓	✓	✓	✓	

$attr = i$
$Con[i]$={c_3,c_5}
$\mathcal{F}$ = {{n},{d,z},{i}}
$C\_ToSolve$ = ∅
$A\_ToPlace$ = {p}

	c_1	c_2	c_3	c_4	c_5	c_6	$N\_con[a_i]$
n	✓	✓	✓	✓			0
d	✓				✓	✓	0
z		✓			✓	✓	0
i			✓		✓		0
p				✓		✓	0
$ToSolve$	✓	✓	✓	✓	✓	✓	

$attr = p$
$Con[p]$={c_4,c_6}
$\mathcal{F}$ = {{n},{d,z},{i,p}}
$C\_ToSolve$ = ∅
$A\_ToPlace$ = ∅

	c_1	c_2	c_3	c_4	c_5	c_6	$N\_con[a_i]$
n	✓	✓	✓	✓			0
d	✓				✓	✓	0
z		✓			✓	✓	0
i			✓		✓		0
p				✓		✓	0
$ToSolve$	✓	✓	✓	✓	✓	✓	

Fig. 6. An example of the execution of Function 5.7.

$\mathcal{F}$ is *vector-minimal* if and only if (i) it is correct, (ii) it maximizes visibility, and (iii) $\nexists \mathcal{F}'$: $\mathcal{F} \prec \mathcal{F}'$ that satisfies the first two conditions. A fragmentation $\mathcal{F}$ computed by function **fragment** satisfies these three properties.

(1) Function **fragment** inserts *attr* into a fragment F if and only if $F \cup \{attr\}$ satisfies the constraints in $Con[attr]$. By induction, we prove that if $F \cup \{attr\}$ satisfies constraints in $Con[attr]$, it satisfies all constraints in $\mathcal{C}_F$. If $\{attr\}$ is the first attribute inserted into F, $F \cup \{attr\} = \{attr\}$ that satisfies all constraints in $\mathcal{C}_F$. Otherwise, if we suppose that F already contains at least one attribute and that it satisfies all constraints in $\mathcal{C}_F$, then, by adding *attr* to F the constraints that may be violated are only the constraints in $Con[attr]$. Consequently, if $F \cup \{attr\}$ satisfies all these constraints, it satisfies all constraints in $\mathcal{C}_F$.
We can, therefore, conclude that $\mathcal{F}$ is a correct fragmentation.

(2) Since each attribute a in $\mathcal{A}_F$ is inserted exactly into one fragment, function **fragment** produces correctly a fragmentation $\mathcal{F}$ that satisfies the condition of maximal visibility.

(3) By contradiction, let $\mathcal{F}'$ be a fragmentation satisfying the constraints in $\mathcal{C}_F$, maximizing visibility, and such that $\mathcal{F} \prec \mathcal{F}'$. Let $V_{\mathcal{F}}$ and $V_{\mathcal{F}'}$ be the fragment vectors associated with $\mathcal{F}$ and $\mathcal{F}'$, respectively. First, we prove that $\mathcal{F}'$ contains a fragment $V_{\mathcal{F}'}[a_i]$ that is the union of two different fragments, $V_{\mathcal{F}}[a_i]$ and $V_{\mathcal{F}}[a_j]$, of $\mathcal{F}$. Second, we prove that function **fragment** cannot generate two different fragments whose union does not violate any constraint. These two results generate a contradiction, since $V_{\mathcal{F}'}[a_i]$, which contains $V_{\mathcal{F}}[a_i] \cup V_{\mathcal{F}}[a_j]$, is a fragment of $\mathcal{F}'$, and thus it does not violate the constraints.
   (a) Since $\mathcal{F} \prec \mathcal{F}'$, there exists a fragment such that $V_{\mathcal{F}}[a_i] \subset V_{\mathcal{F}'}[a_i]$, and then there exists an attribute a_j (with $j \neq i$) such that $a_j \in V_{\mathcal{F}'}[a_i]$ and $a_j \notin V_{\mathcal{F}}[a_i]$. Note that $a_j \neq a_i$ because, by definition, $a_i \in V_{\mathcal{F}}[a_i]$ and $a_i \in V_{\mathcal{F}'}[a_i]$. $V_{\mathcal{F}}[a_j]$ and $V_{\mathcal{F}'}[a_j]$ are the fragments that contain a_j. We now show that the whole fragment $V_{\mathcal{F}}[a_j] \subset V_{\mathcal{F}'}[a_i]$. Since $a_j \in V_{\mathcal{F}'}[a_j]$ and $a_j \in V_{\mathcal{F}'}[a_i]$, we have that $V_{\mathcal{F}'}[a_j] = V_{\mathcal{F}'}[a_i]$, but since $V_{\mathcal{F}}[a_j] \subset V_{\mathcal{F}'}[a_j] = V_{\mathcal{F}'}[a_i]$, we have that $(V_{\mathcal{F}}[a_i] \cup V_{\mathcal{F}}[a_j]) \subseteq V_{\mathcal{F}'}[a_i]$.
   (b) Let F_h and F_k be the two fragments computed by function **fragment**, corresponding to $V_{\mathcal{F}}[a_i]$ and $V_{\mathcal{F}}[a_j]$, respectively. Assume, without loss of generality, that $h < k$ (since the proof in the case $h > k$ immediately follows by symmetry). Let a_{k_1} be the first attribute inserted into F_k by the function. Recall that the function inserts an attribute into a new fragment if and only if the attribute cannot be inserted into the already-existing fragments (e.g., F_h) without violating constraints. Therefore, the set of attributes $F_h \cup \{a_{k_1}\}$ violates a constraint as well as the set $V_{\mathcal{F}}[a_i] \cup V_{\mathcal{F}}[a_j]$ that contains $F_h \cup \{a_{k_1}\}$.
   This generates a contradiction.

We can conclude that function **fragment** computes a vector-minimal fragmentation. □

THEOREM 5.10 (COMPLEXITY). *Given a set $\mathcal{C} = \{c_1, \ldots, c_m\}$ of constraints and a set $\mathcal{A} = \{a_1, \ldots, a_n\}$ of attributes the complexity of function* **fragment**$(\mathcal{A},\mathcal{C})$ *in Figure 5 is $O(n^2m)$ in time.*

PROOF. To choose attribute *attr* from *A_ToPlace*, in the worst case, function **fragment** scans array *N_con*[] and adjusts array *N_con*[] for each attribute involved in at least one constraint with *attr*. This operation costs $O(nm)$ for each chosen attribute. Each attribute is then inserted into a fragment. Note that the number of fragments is $O(n)$ in the worst case. To choose the right fragment that will contain *attr*, in the worst case the function tries to insert it into all fragments $F \in \mathcal{F}$, and compares $F \cup \{attr\}$ with the constraints in $Con[attr]$. Since the sum of the number of attributes in all the fragments is $O(n)$, then $O(n)$ attributes will be compared with the $O(m)$ constraints containing *attr*, giving, in the worst case, a $O(nm)$ complexity for each *attr*. Thus, the complexity of choosing the right fragment is $O(n^2m)$. We can then conclude that the overall time complexity is $O(n^2m)$. □

6. TAKING ATTRIBUTE AFFINITY INTO ACCOUNT

The computation of a minimal fragmentation exploits the basic principle according to which the presence of a high number of plaintext attributes permits an efficient execution of queries. Although this principle may be considered acceptable in many situations, other criteria can also be applied for computing a fragmentation. Indeed, depending on the use of the data, it may be useful to preserve the associations among some attributes. As an example, consider the fragmentation in Figure 3 and suppose that physicians should be able to explore the link between a specific `Illness` and the age (`DoB`) of patients. The computed fragmentation, however, does not make visible the association between `Illness` and `DoB`, thus making the required analysis not possible (as it would violate the constraints). In this case, a fragmentation where these two attributes are stored in clear form in the same fragment is preferable to the computed fragmentation. The need for keeping together some specific attributes in the same fragment may not only depend on the use of the data but also on the queries that need to be frequently executed on the data. Indeed, given a query Q and a fragmentation $\mathcal{F}$, the execution cost of Q varies according to the specific fragment used for computing the query. This implies that, with respect to a specific query workload, different fragmentations may be more convenient than others in terms of query performance.

To take into consideration both the use of the data and the query workload in the fragmentation process, we exploit the concept of *attribute affinity*, which is traditionally applied to express the advantage of having pairs of attributes in the same fragment in distributed DBMSs [Özsu and Valduriez 1999]. Attribute affinity may be, therefore, adopted by schema design algorithms that use the knowledge of a representative *workload* for computing a suitable partition. In our context, attribute affinity is also a measure of how strong the need of keeping the attributes in the same fragment is. In general, the identification of affinity values starts from a consideration of the expected profile of queries that will be executed. By assuming that the set of attributes in R that appear in

	n	d	z	i	p
n		10	5	25	15
d			5	20	30
z				10	5
i					15
p					

Fig. 7. An example of affinity matrix.

nonsingleton constraints are first arbitrarily ordered and that notation a_i denotes the i-th attribute, the affinity between attributes is represented through an *affinity matrix*. The matrix, denoted M, has a row and a column for each attribute appearing in nonsingleton constraints, and each cell $M[a_i, a_j]$ represents the benefit obtained by having attributes a_i and a_j in the same fragment. Clearly, the affinity matrix contains only positive values and is symmetric with respect to its main diagonal. Also, for all attributes a_i, $M[a_i, a_i]$ is not defined. The affinity matrix can then be represented as a triangular matrix, where only cells $M[a_i, a_j]$, with $i < j$, are represented. Figure 7 illustrates an example of affinity matrix for relation MEDICALDATA in Figure 2. The attributes have been ordered according to the same order with which they have been listed in the schema of the considered relation.

The consideration of attribute affinity naturally applies to fragments and fragmentations. Fragmentations that maintain together attributes with high affinity are to be preferred. To reason about this, we define the concept of *fragmentation affinity*. Intuitively, the affinity of a fragment is the sum of the affinity of the different pairs of attributes in the fragment; the affinity of a fragmentation is the sum of the affinity of the fragments in it. The use of the sum to compose different affinities represents a simple and effective mechanism, supported by the experience on the use of this model. While other, more sophisticated, composition rules could be adopted, we note that the use of the sum is consistent with the nature of the affinity matrix, which aims at modeling complex relations in a compact and effective way. The concept of fragmentation affinity is formalized as follows.

Definition 6.1 (*Fragmentation Affinity*). Let R be a relation schema, M be an affinity matrix for R, $\mathcal{C}$ be a set of well-defined constraints, and $\mathcal{F} = \{F_1, \ldots, F_n\}$ be a correct fragmentation of R. The affinity of $\mathcal{F}$, denoted $\mathit{affinity}(\mathcal{F})$, is defined as: $\mathit{affinity}(\mathcal{F}) = \sum_{k=1}^{n} \mathit{aff}(F_k)$, where $\mathit{aff}(F_k) = \sum_{a_i, a_j \in F_k, i<j} M[a_i, a_j]$ is the affinity of fragment F_k, $k = 1, \ldots, n$.

As an example, consider the affinity matrix in Figure 7 and fragmentation $\mathcal{F} = \{\{\texttt{Name}\}, \{\texttt{DoB,Illness,Physician}\}, \{\texttt{ZIP}\}\}$. Then, $\mathit{affinity}(\mathcal{F}) = \mathit{aff}(\{\texttt{Name}\}) + \mathit{aff}(\{\texttt{DoB,Illness,Physician}\}) + \mathit{aff}(\{\texttt{ZIP}\}) = 0 + (M[d, i] + M[d, p] + M[i, p]) + 0 = 0 + (20 + 30 + 15) + 0 = 65$. With the consideration of affinity, the problem becomes, therefore, to determine a correct fragmentation that has maximum affinity. This is formally defined as follows.

Problem 6.2 (*Maximum Affinity*). Given a relation schema R, a set $\mathcal{C}$ of well-defined constraints over R, and an affinity matrix M, find a *fragmentation* $\mathcal{F}$ of R such that all the following conditions hold.

(1) $\mathcal{F}$ correctly enforces $\mathcal{C}$ (Definition 3.3).
(2) $\mathcal{F}$ maximizes visibility (Definition 4.1).
(3) $\nexists \mathcal{F}'$ satisfying the first two conditions such that *affinity*$(\mathcal{F}') >$ *affinity*$(\mathcal{F})$.

We can now prove that also the maximum affinity problem is *NP-hard*.

THEOREM 6.3. *The maximum affinity problem is NP-hard.*

PROOF. The proof is a reduction from the NP-hard minimum hitting set problem [Garey and Johnson 1979], which can be formulated as follows: Given a collection C of subsets of a set S, determine the smallest subset S' of S such that S' contains at least one element from each subset in C.

Let $R = S \cup \{a_c\}$, where a_c is an element different from any element in S. We can observe that any singleton set in C corresponds to an element that is part of a solution S'; therefore, it can be directly inserted in S'. Also, whenever there are two sets s_i,s_j in C with $s_i \subset s_j$, s_j is redundant and can be removed from C, since a solution S' that includes an element in s_i includes also an element s_j. Thus, let $\mathcal{C}_F = \{s \in C\colon |s| > 1 \text{ and } \forall s_i, s_j \in C, i \neq j,\ s_i \not\subset s_j\}$ be the set of constraints, and let $\mathcal{A}_F = \{a \in R\colon \{a\} \not\in C\}$ be the set of attributes to be fragmented. It is easy to see that the construction of an instance of the maximum affinity problem from an instance of the minimum hitting set problem is polynomial in C. Also, a_c is an attribute that is not involved in any constraint in $\mathcal{C}_F$. Consider now an affinity matrix such that $M[a_i, a_j] = 1$ if and only if $a_i = a_c$ or $a_j = a_c$; $M[a_i, a_j] = 0$, otherwise.

Since $M[a_i, a_j] = 0$ when $a_i, a_j \neq a_c$, a fragmentation algorithm that maximizes the affinity computes a fragmentation where fragment F_c containing a_c includes the maximum number of attributes that can be inserted without violating the constraints; the affinity of the computed fragmentation corresponds to the cardinality of F_c. It is also easy to see that maximizing the number of attributes of F_c is equivalent to minimizing the size of the set S' of attributes that contains at least one attribute from each constraint. Consequently, a maximal affinity fragmentation $\mathcal{F}$ of R, with respect to M, satisfying all constraints in $\mathcal{C}_F$ corresponds to a solution for the minimum hitting set problem. In particular, given fragment F_c, the solution of the minimum hitting set problem is $S' = R - F_c$. □

In the following, we describe a heuristic approach for Problem 6.2.

7. A HEURISTIC APPROACH TO MAXIMIZE AFFINITY

Our heuristic approach to determine a fragmentation that maximizes affinity exploits a greedy approach that, at each step, combines fragments that have the highest affinity. The heuristic starts by putting each attribute to be fragmented into a different fragment. The affinity between pairs of fragments is the affinity between the attributes contained in their union (as dictated by the affinity matrix). Then, the fragments with the highest affinity, let us call them F_i and F_j, are merged together (if this does not violate constraints), and F_i is updated by adding the attributes of F_j, while F_j is removed. The affinity of the new version of F_i with respect to any other fragment F_k is the sum of the affinities

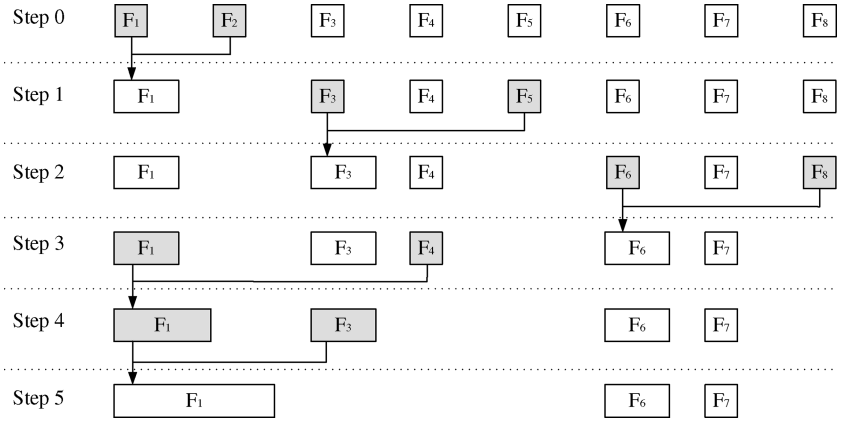

Fig. 8. Graphical representation of the working of Function 7.1.

that F_k had with the old version of F_i and F_j. The heuristic proceeds in a greedy way iteratively merging, at each step, the fragments with highest affinity until no more fragments can be merged without violating the constraints. Figure 8 gives a graphical representation of our heuristic approach; at each step, light grey boxes denote the pair of fragments with highest affinity. The correctness of the heuristics lies in the fact that, at each step, the affinity of the resulting fragmentation can only increase. As a matter of fact, it is easy to see that affinity is monotonic with respect to the dominance relationship (see Lemma 7.3 in Section 7.2).

The following section describes the function implementing this heuristic approach. In the function, instead of controlling constraints when determining whether two fragments can be merged, we exploit the affinity matrix and set to -1 the affinity of fragments whose merging would violate the constraints (thus, ignoring them in the evaluation of fragments to be merged).

7.1 Computing a Vector-Minimal Fragmentation with the Affinity Matrix

Function 7.1 takes as input a set *A_ToPlace* of attributes to be fragmented and a set *C_ToSolve* of well-defined nonsingleton constraints. It computes a vector-minimal fragmentation $\mathcal{F}$ of *A_ToPlace*, where at each step, the fragments to be merged are chosen according to their affinity. In the following, with a slight abuse of notation, we use $M[F_i, F_j]$ to denote the cell in the affinity matrix identified by the smallest attribute in F_i and F_j, according to the order set on attributes appearing in nonsingleton constraints.

First, the function initializes $\mathcal{F}$ to a fragmentation having a fragment F_i for each of the attributes a_i in *A_ToPlace* and creates a set *FragmentIndex* that contains the index i of each fragment $F_i \in \mathcal{F}$. The function also checks all constraints in *C_ToSolve* composed of two attributes only, and sets to -1 the corresponding cells in the affinity matrix. These constraints are then removed from *C_ToSolve*. In general, at each iteration of the algorithm, for each $i < j$, $M[F_i, F_j]$ is equal to -1 if fragment $F_i \cup F_j$ violates some constraints.

FUNCTION 7.1 VECTOR-MINIMAL FRAGMENTATION WITH THE AFFINITY MATRIX.

```
FRAGMENT(A_ToPlace, C_ToSolve)
/* initial solution with a fragment for each attribute */
F := ∅
FragmentIndex := ∅
for i=1...|A_ToPlace| do
   F_i := {a_i}
   F := F ∪ {F_i}
   FragmentIndex := FragmentIndex ∪ {i}
/* cells in M corresponding to constraints are invalidated */
for each {a_x,a_y} ∈ C_ToSolve do
   M[F_min(x,y),F_max(x,y)] := −1
   C_ToSolve := C_ToSolve − {{a_x,a_y}}
/* extract the pair of fragments with maximum affinity */
Let [F_i,F_j], i<j and i, j ∈ FragmentIndex, be the pair of fragments with maximum affinity
while |FragmentIndex| > 1 AND M[F_i, F_j] ≠ −1 do /* merge the two fragments */
   F_i := F_i∪F_j
   F := F − {F_j}
   FragmentIndex := FragmentIndex − {j}
   /* update the affinity matrix */
   for each k∈FragmentIndex : k≠i do
      if M[F_min(i,k),F_max(i,k)]=−1 OR M[F_min(j,k),F_max(j,k)]=−1 then
         M[F_min(i,k),F_max(i,k)] := −1
      else
         for each c∈C_ToSolve do
            if c⊆(F_i∪F_k) then
               M[F_min(i,k),F_max(i,k)] := −1
               C_ToSolve := C_ToSolve − {c}
            endif
         if M[F_min(i,k),F_max(i,k)]≠−1 then
            M[F_min(i,k),F_max(i,k)] := M[F_min(i,k),F_max(i,k)] + M[F_min(j,k),F_max(j,k)]
         endif
      endif
   Let [F_i,F_j], i<j and i, j ∈ FragmentIndex, be the pair of fragments with maximum affinity
return(F)
```

Fig. 9. Function that finds a vector-minimal fragmentation with maximal affinity.

Function **fragment** then executes a **while** loop that, at each iteration, merges two fragments in $\mathcal{F}$ as follows. If there are still pairs of fragments that can be merged, that is, there are still cells in M different from -1, the function identifies the cell $[F_i, F_j]$ (with $i < j$) with the maximum value in M. Then, F_i is updated to the union of the two fragments and F_j is removed from $\mathcal{F}$. Also, j is removed from *FragmentIndex*, since the corresponding fragment is no more part of the solution. The function then updates M. In particular, for each fragment F_k, $k \in \{FragmentIndex - i\}$, cell $M[F_i, F_k]$ is set to -1 if either cell $M[F_i, F_k]$ or cell $M[F_j, F_k]$ is -1, or if $F_i \cup F_k$ violates at least a constraint still in *C_ToSolve*. In this latter case, the violated constraints are removed from *C_ToSolve*. Otherwise, cell $M[F_i, F_k]$ is summed with the value in cell $M[F_j, F_k]$.

Example 7.2. Figure 10 presents the execution, step by step, of function **fragment** represented in Figure 9, applied to the example in Figure 2 and considering the affinity matrix in Figure 7. The left-hand side of Figure 10 illustrates the evolution of fragments and of the chosen pair F_i, F_j. The central part of Figure 10 illustrates the evolution of matrix M, where dark-grey columns represent fragments merged with other fragments, and thus removed from the set of fragments. The right-hand side of Figure 10 illustrates the set

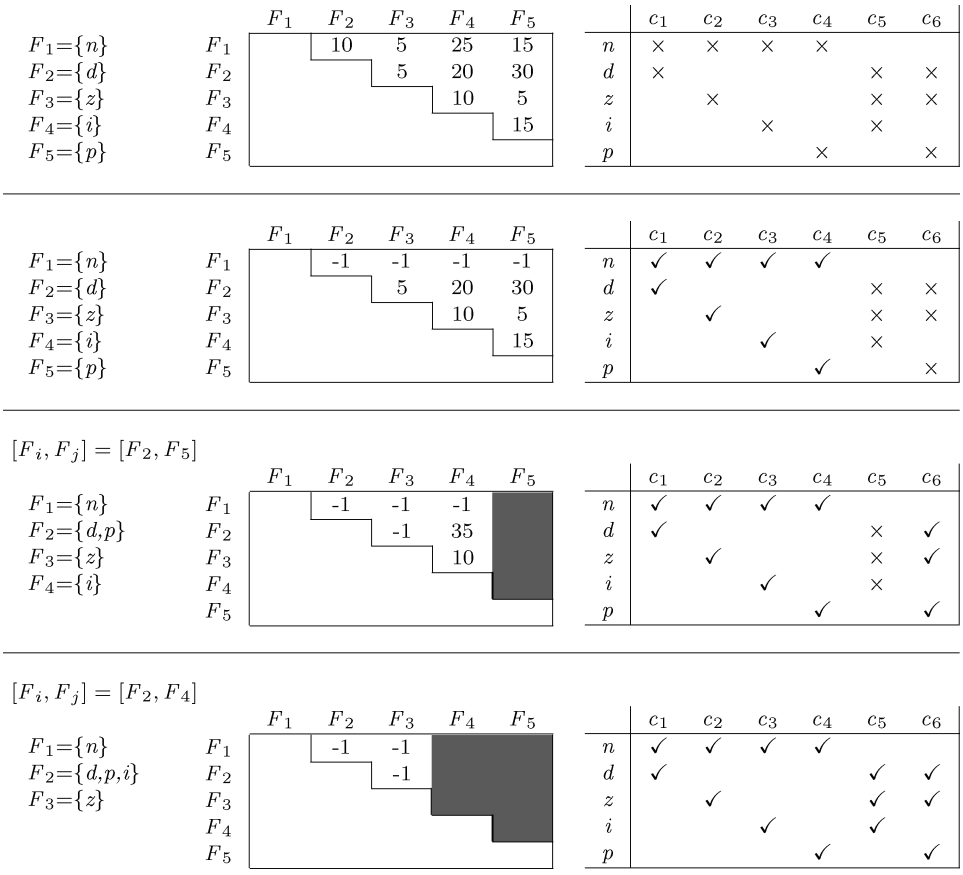

Fig. 10. An example of the execution of Function 7.1.

C_ToSolve of constraints to be solved: if an attribute belongs to constraint c_i in *C_ToSolve*, the corresponding cell is set to $\times$; if c_i is removed from *C_ToSolve*, the cell is set to $\checkmark$. At the beginning, all constraints are not solved and there is a fragment F for each attribute in *A_ToPlace*. First, M is updated by setting to -1 the cells representing constraints involving only two attributes, that is, constraints c_1, c_2, c_3, and c_4, which are then removed from *C_ToSolve*. Function **fragment** chooses the cell in M with the highest affinity, that is, $M[F_2, F_5] = 30$. Consequently, F_5 is merged with F_2 (the 5th column becomes dark grey to denote that fragment F_5 does not exist anymore). Then, values in the affinity matrix are updated: cell $M[F_2, F_3]$ is set to -1, since it represents fragment $\{d, p, z\}$ that violates constraint c_6, which is, therefore, removed from *C_ToSolve*; cell $M[F_2, F_4]$ is set to $M[F_2, F_4] + M[F_4, F_5] = 20 + 15 = 35$. At the next iteration, cell $M[F_2, F_4]$ is chosen and F_4 is merged with F_2 (the 4th column becomes dark grey to denote that fragment F_4 does not exist anymore). Then, cell $M[F_2, F_3]$ is set to -1, since it was -1 in the previous iteration as well. The final solution is $\mathcal{F} = \{\{n\}, \{d, p, i\}, \{z\}\}$, with affinity equal to 65. (Note that the solution computed by function **fragment** in Figure 5, and

represented in Figure 6, has 3 fragments as the solution computed here, but its affinity is 20.)

We note that Function 7.1 can be used to simulate Function 5.7 sorting the attributes in the order with which they are considered by the algorithm in Figure 5 and considering an initial affinity matrix containing 0 as affinity value between each pair of attributes. The ordering of attributes can be simply computed by iteratively calculating the number of unsolved constraints $N\_con[a]$ involving each attribute a, and inserting, as next element of the ordered list, the attribute that maximizes $N\_con[a]$. Since the affinity matrix contains values 0 and -1 only, the order for choosing fragments as the next maximum affinity pair is the same of Function 5.7.

7.2 Correctness and Complexity

Before proving the correctness and complexity of our heuristic, we introduce two lemmas proving the monotonicity property of fragmentation affinity with respect to the dominance relationship $\preceq$ and the correctness of the matrix computation, respectively.

LEMMA 7.3 (MONOTONICITY). *Let R be a relation, M be an affinity matrix for R, $\mathcal{C}$ be a set of well-defined constraints, and $\mathcal{F}$ and $\mathcal{F}'$ be two correct fragmentations for R. If $\mathcal{F} \preceq \mathcal{F}' \Rightarrow$ affinity($\mathcal{F}$) $\leq$ affinity($\mathcal{F}'$).*

PROOF. By definition, given two fragmentations $\mathcal{F} = \{F_1, \ldots, F_n\}$ and $\mathcal{F}' = \{F'_1, \ldots, F'_m\}$ such that $\mathcal{F} \prec \mathcal{F}'$, then $V_{\mathcal{F}}[a] \subseteq V_{\mathcal{F}'}[a], \forall a \in \{a \in R : \{a\} \notin \mathcal{C}\}$. Therefore, for each a such that $V_{\mathcal{F}}[a] = V_{\mathcal{F}'}[a]$, the affinity of the two fragments F and F' containing a in $\mathcal{F}$ and $\mathcal{F}'$, is, respectively, the same. For each a such that $V_{\mathcal{F}}[a] \subset V_{\mathcal{F}'}[a]$, the affinity of the two fragments F and F' containing a in $\mathcal{F}$ and $\mathcal{F}'$, respectively, is such that $aff(F) \leq aff(F')$. In fact, $aff(F') = aff(F) + \sum M[a_i, a_j]$, $\forall a_i \in F'$, $a_j \in F' - F$, with $i < j$. Since $M[a_i, a_j]$ is always a nonnegative value, it holds that if $\mathcal{F} \prec \mathcal{F}'$, then *affinity*($\mathcal{F}$) $\leq$ *affinity*($\mathcal{F}'$).

If $\mathcal{F} = \mathcal{F}'$, it is straightforward to see that *affinity*($\mathcal{F}$) = *affinity*($\mathcal{F}'$). □

LEMMA 7.4. *At the beginning of each iteration of the* **while** *loop of function* **fragment** *in Figure 9, $M[F_i, F_j] = -1 \Leftrightarrow \exists c \in \mathcal{C} : c \subseteq (F_i \cup F_j)$.*

PROOF. At initialization, function **fragment** checks constrains involving exactly two attributes $\{a_x, a_y\}$ and sets to -1 the cell in M corresponding to the pair of fragments $F_x = \{a_x\}$ and $F_y = \{a_y\}$. Also, all these constraints are removed from *C_ToSolve*.

When function **fragment** merges two fragments F_i and F_j $(i < j)$, j is removed from *FragmentIndex*. For each k in *FragmentIndex* but i, cell $M[F_{min(i,k)}, F_{max(i,k)}]$ is set to -1 if either $M[F_{min(i,k)}, F_{max(i,k)}]$ or $M[F_{min(j,k)}, F_{max(j,k)}]$ were -1 before the update. Indeed, if either $F_i \cup F_k$ or $F_j \cup F_k$ violated a constraint before merging F_i with F_j, also $F_i \cup F_k$ violates the same constraint after the update (i.e., $\exists c \in \mathcal{C}$ such that $c \subseteq F_i$ or $c \subseteq F_j$), since F_i is set to $F_i \cup F_j$. Note that constraints removed from *C_ToSolve* are represented by -1 in M. Also, when $F_i \cup F_k$ is checked against constraints in *C_ToSolve*, the algorithm looks for constraints representing a subset of $F_i \cup F_k$. If such constraints exist,

$M[F_{min(i,k)}, F_{max(i,k)}]$ is set to -1 and they are removed from *C_ToSolve*, since value -1 in M represents them. □

THEOREM 7.5 (CORRECTNESS). *Function* **fragment** *in Figure 9 terminates and finds a vector-minimal fragmentation* (*Definition 5.5*).

PROOF. Let R be a relation schema, and $\mathcal{C}$ be a set of well-defined constraints. Consider the sets $\mathcal{C}_F = \{c \in \mathcal{C} : |c| > 1\}$ and $\mathcal{A}_F = \{a \in R: \{a\} \notin \mathcal{C}\}$.

Function **fragment** always terminates. In fact, the **while** loop terminates because at each iteration the number of indexes in *FragmentIndex* decreases by one, and the iterations are performed only if *FragmentIndex* contains at least two indexes.

We now prove that a solution $\mathcal{F}$ computed by function **fragment** over $\mathcal{A}_F$ and $\mathcal{C}_F$ is a vector-minimal fragmentation. According to Definition 5.5 of minimality, a fragmentation $\mathcal{F}$ is *minimal* if and only if (1) it is correct, (2) it maximizes visibility, and (3) $\nexists \mathcal{F}'$: $\mathcal{F} \prec \mathcal{F}'$ that satisfies the first two conditions.

(1) Function **fragment** starts with a simple correct fragmentation ($F_i = \{a_i\}$, for all $a_i \in \mathcal{A}_F$), and it iteratively merges only fragments that form a correct fragment, since the pair of fragments to be merged is extracted as the pair with maximum affinity and the fragments are merged only if their affinity is a positive value. By Lemma 7.4, only fragments whose union does not violate constraints are merged. We can, therefore, conclude that $\mathcal{F}$ correctly enforces $\mathcal{C}$.

(2) Since each attribute in $\mathcal{A}_F$ is initially inserted exactly into one fragment, and when two fragments are merged, only the result of their union is kept in $\mathcal{F}$, the condition of maximal visibility is satisfied.

(3) By contradiction, let $\mathcal{F}'$ be a fragmentation satisfying the constraints in $\mathcal{C}_F$ and maximizing visibility, such that $\mathcal{F} \prec \mathcal{F}'$. Let $V_{\mathcal{F}}$ and $V_{\mathcal{F}'}$ be the fragment vectors associated with $\mathcal{F}$ and $\mathcal{F}'$, respectively.

As already proven in the proof of Theorem 5.9, $\mathcal{F}'$ contains a fragment $V_{\mathcal{F}'}[a_i]$ that is the union of two different fragments, $V_{\mathcal{F}}[a_i]$ and $V_{\mathcal{F}}[a_j]$, of $\mathcal{F}$. We need then to prove that function **fragment** cannot terminate with two different fragments whose union does not violate any constraint.

Let F_h and F_k be the two fragments computed by function **fragment**, corresponding to $V_{\mathcal{F}}[a_i]$ and $V_{\mathcal{F}}[a_j]$, respectively. Assume, without loss of generality, that $h < k$ (since the proof in the case $h > k$ immediately follows by symmetry). By Lemma 7.4, M contains nonnegative values only for pairs of fragments whose union generates a correct fragment; therefore, function **fragment** cannot terminate with fragmentation $\mathcal{F}$, since M still contains a nonnegative value to be considered ($M[F_h, F_k]$). This generates a contradiction.

We can conclude that function **fragment** computes a vector-minimal fragmentation. □

THEOREM 7.6 (COMPLEXITY). *Given a set* $\mathcal{C} = \{c_1, \ldots, c_m\}$ *of constraints and a set* $\mathcal{A} = \{a_1, \ldots a_n\}$ *of attributes the complexity of function* **fragment***($\mathcal{A}$, $\mathcal{C}$) in Figure 9 is* $O(n^3m)$ *in time.*

PROOF. The first **for** and **for each** loops of function **fragment** cost $O(n+m)$. The **while** loop is performed $O(n)$ times, since, at each iteration, an element from *FragmentIndex* is extracted. The **for each** loop nested into the **while** loop updates the cells corresponding to fragments F_i and F_j in the affinity matrix. While j is simply removed from *FragmentIndex*, and the column F_j in the matrix is simply ignored, the update of the cells corresponding to F_i, which are $O(n)$ in number, costs $O(n^2m)$ because all the constraints in *C_ToSolve* containing $F_i \cup F_j$ are considered. Each extraction of the pair of fragments with maximum affinity from M simply scans (in the worst case) the affinity matrix, and its computational cost is $O(n^2)$ in time. The overall time complexity is, therefore, $O(n^3m)$. □

8. QUERY EXECUTION

Fragmentation of a relation R implies that only fragments, which are stored in place of the original relation to satisfy confidentiality constraints, are used for query execution. The fragments can be stored on a single server or on multiple servers. The server (or servers) storing the fragments while needs not be trusted with respect to the confidentiality, since accessing single fragments or encrypted information does not expose to any privacy breach, it is trusted for correctly evaluating queries on fragments (honest but curious).

Users who are not authorized to access the content of the original relation R have only a *partial view* on the data, meaning that they can only access the fragments. A query submitted by a user with a partial view can be presented directly to the server(s) storing the desired fragment. Users who are authorized to access the content of the original relation have a *full view* on the data and can present queries referring to the schema of the original relation. The queries issued by users with full view are then translated into equivalent queries operating on the encrypted and fragmented data stored on the server(s). The translation process is executed by a trusted component, called *query mapping component*, that is invoked every time there is the need to access sensitive information (see Figure 11). In particular, the query mapping component receives a query Q submitted by a user with full view along with the key k possibly needed for decrypting the query result computed by the server, and returns the result of query Q to the user. However, since every physical fragment of R contains all the attributes of R, either in encrypted or in clear form, no more than one fragment needs to be accessed to respond to Q. The query mapping component, therefore, maps the user's query Q onto an equivalent query Q_s, working on a specific fragment. The server executes the received query Q_s on the required fragment and returns the result to the query mapping component. Note that whenever query Q may involve attributes that do not appear in the clear form in the selected fragment, the query mapping component may need to execute an additional query Q_u on the decrypted results of Q_s, which is in charge of enforcing all conditions that cannot be evaluated on the physical fragment or of projecting the attributes reported in the *select* clause of query Q. In this case, the query mapping component decrypts the result received, executes query Q_u on it, and returns the

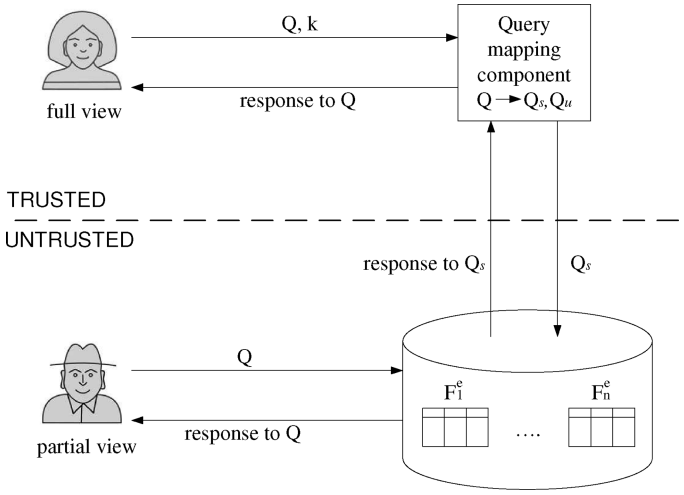

Fig. 11. Interactions among users and server storing the fragments.

result of Q_u to the user. We now describe the query translation process in more details.

We consider *select-from-where* SQL queries of the form Q = "SELECT A_Q FROM R WHERE C", where A_Q is a subset of the attributes of R, and C is a conjunction of basic conditions $c_1, \ldots, c_n$ of the form $(a\ op\ v)$ or $(a_j\ op\ a_k)$, with a, a_j and a_k attributes of R, v constant value, and op comparison operator in $\{=, \neq, >, <, \leq, \geq\}$. Let us then consider the evaluation of query Q on physical fragment $F_i^e(\underline{salt}, enc, a_{i_1}, \ldots, a_{i_n})$, where *salt* is the primary key, *enc* contains the encrypted attributes, and $a_{i_1}, \ldots, a_{i_n}$ are the plaintext attributes (see Section 3). Suppose, for generality, that C contains some conditions that involve attributes stored in the clear form in F_i^e and some others that cannot instead be evaluated on F_i^e. The query mapping component translates the original query Q into a query Q_s operating on the physical fragment and defined as:

SELECT $A_Q \cap \{a_{i_1}, \ldots, a_{i_n}\}$, *salt*, *enc*
FROM F_i^e
WHERE $\bigwedge_{c_j \in C_i^e} c_j$

where C_i^e is the set of basic conditions in C that can be evaluated on physical fragment F_i^e, that is, $C_i^e = \{c_j \mid c_j \in C \wedge attributes(c_j) \in F_i^e\}$, with $attributes(c_j)$ representing the attributes appearing in c_j. Note that the *salt* and *enc* attributes in the SELECT clause of Q_s are specified only if the SELECT or WHERE clauses of the original query Q involve attributes not appearing in clear form in the fragment. The query mapping component then decrypts the tuples received and executes on them a query Q_u defined as:

SELECT A_Q
FROM $Decrypt(Q_s, k)$
WHERE $\bigwedge_{c_j \in \{C - C_i^e\}} c_j$

Original query on R	**Translation over encrypted fragments**
Q := SELECT SSN, Name FROM MedicalData WHERE Illness='obesity' AND Physician='D. Warren'	$Q_{s.3}$:= SELECT salt, *enc* FROM F_3^e WHERE Illness='obesity'AND Physician='D. Warren' Q_u := SELECT SSN, Name FROM *Decrypt*(Q$_{s.3}$, *Key*)
Q′ := SELECT SSN, Name FROM MedicalData WHERE Illness='obesity' AND Physician='D. Warren' AND ZIP='94139'	$Q'_{s.3}$:= SELECT salt, *enc* FROM F_3^e WHERE Illness='obesity'AND Physician='D. Warren' Q'_u := SELECT SSN, Name FROM *Decrypt*(Q$'_{s.3}$, *Key*) WHERE ZIP='94139'

Fig. 12. An example of query translation over a fragment.

where *Decrypt*(Q_s, k) denotes a temporary relation, including the tuples returned by Q_s, and where attribute *enc* has been decrypted through key k. The WHERE clause of Q_u includes all conditions defined on attributes that do not appear in clear form in the physical fragment and that can be only evaluated on the decrypted result. The final result of query Q_u is then returned to the user.

Note that since we are interested in minimizing the query evaluation cost, a query optimizer can be used to select the fragment that allows the execution of more selective queries by the server, thus decreasing the workload of the application and maximizing the efficiency of the execution [Chaudhuri 1998].

Example 8.1. Consider the relation in Figure 2(a) and its fragments in Figure 3.

—Consider a query Q retrieving the Social Security Number and the name of the patients whose illness is obesity and whose physician is D. Warren. Since fragment F_3^e contains both Illness and Physician, it can evaluate both the conditions in the WHERE clause and is chosen for query evaluation. Figure 12 illustrates the translation of Q to queries $Q_{s.3}$ executed by the server on fragment F_3^e (notation $Q_{s.x}$ indicates a query executed by the server on fragment x), and Q_u executed by the application. Query $Q_{s.3}$ returns to the application only the tuples belonging to the final result. The application just needs to decrypt them for projecting the SSN and Name attributes.

—Consider a query Q' retrieving the Social Security Number and the name of the patients whose illness is obesity, whose physician is D. Warren, and whose ZIP is 94139. Fragment F_3^e contains both Illness and Physician, thus allowing the evaluation of two out of three conditions. Fragment F_2 contains only ZIP and allows the evaluation of one out of three conditions. The query mapping component, therefore, translates query Q' into queries $Q'_{s.3}$

executed by the server on fragment F_3^e, and Q'_u executed by the application (see Figure 12). Since ZIP does not appear in clear form in fragment F_3^e, the condition on it needs to be evaluated by the application, which also performs the projection of the SSN and Name attributes after decrypting the result computed by $Q'_{s.3}$.

Note that queries whose WHERE clause contains negated conditions can be easily managed by the query mapping component, since whenever a basic condition c can be evaluated on a physical fragment; its negation (i.e., NOT(c)) can also be evaluated on the same fragment. Queries whose WHERE clauses contain disjunctions need special consideration. As a matter of fact, according to the semantics of the OR, any condition that cannot be evaluated over a fragment but that is in disjunction with other conditions that can be evaluated on the fragment cannot be simply evaluated on the result returned by the server (like done in the case of conjunction). There are, therefore, three possible scenarios: (i) The query conditional part can be reduced to a conjunctive normal form; then the query mapping and evaluation can proceed as illustrated in the conjunctive case described previously. (ii) The query conditional part can be reduced to a disjunctive normal form where all components can be evaluated over different fragments; in this case, the query mapping component will ask the server for the execution of as many queries as the components of the disjunction and will then merge (union) their results. (iii) The query conditional part contains a basic condition (to be evaluated in disjunction with others) that cannot be evaluated on any fragment (as it involves a sensitive attribute or attributes that appear in two different fragments); in this case, the query mapping component will need to retrieve the entire fragment (any fragment will do) and will evaluate the query condition at its site.

9. EXPERIMENTAL RESULTS

The heuristic algorithms presented in Sections 5 and 7 have been implemented as C programs to obtain experimental data and assess the behavior of the algorithms in terms of execution time and quality of the returned solution. Aiming to a comparison of the results computed by our heuristic algorithms to the optimal solutions, we also implemented two algorithms analyzing the complete solution space computing the fragmentation with the minimal number of fragments and the one with maximum affinity. The relation schema we considered in the experiments is composed of 32 attributes and is inspired by a database of medical information. Taking into account possible confidentiality requirements, we expressed up to 30 confidentiality constraints. These constraints are well-defined (see Definition 2.2) and composed of a number of attributes varying from 2 to 4 (we did not consider singleton constraints, as they cannot be solved via fragmentation). The content of the affinity matrix has been produced using a pseudorandom generation function. The experiments have considered configurations with an increasing number of attributes, from 3 to 32, taking into account, for every configuration, only the constraints completely fitting in the selected attributes. The number of constraints for a configuration with n attributes ranges between $n - 3$ and $n + 1$.

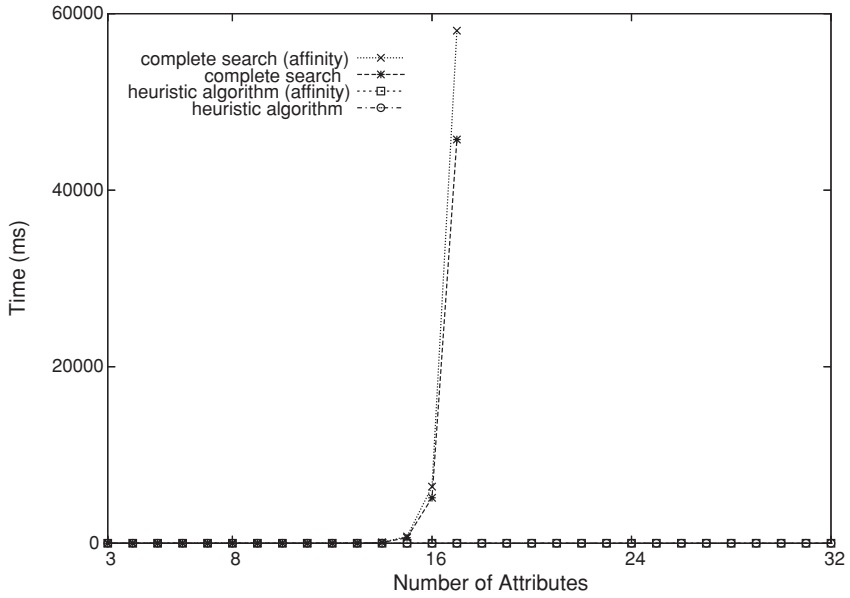

Fig. 13. Computational time of the algorithms.

Figure 13 compares the time required for the execution of the complete search algorithms with the heuristic algorithms presented in this article. Consistently with the fact that both the problem of minimizing the number of fragments and the problem of maximizing affinity while satisfying confidentiality constraints are NP-hard, the two complete search strategies require exponential time in the number of attributes. The complete search then becomes unfeasible even for a relatively small number of attributes; with the availability of large computational resources it would still not be possible to consider large configurations (in our experiments, we were able only to run the complete search for schemas with less than 17 attributes). By contrast, the time required for the execution of both the heuristic algorithms always remains low (it is close to 0). This guarantees that both the heuristic algorithms proposed are applicable to large relational schemas.

Obviously, a successful heuristic presents a good behavior if it combines time efficiency with a demonstrated ability to produce good solutions. We, therefore, compared the solutions computed by the execution of each of the two heuristic algorithms with those returned by the corresponding complete search algorithm.

Figure 14 presents the number of fragments obtained by the execution of the heuristic algorithm computing a vector-minimal fragmentation (Function 5.7) compared with the minimal number of fragments in a solution computed by the complete search for a solution satisfying all the considered constraints. As the graph shows, in all the cases that allow the comparison, our heuristic has always identified an optimal solution.

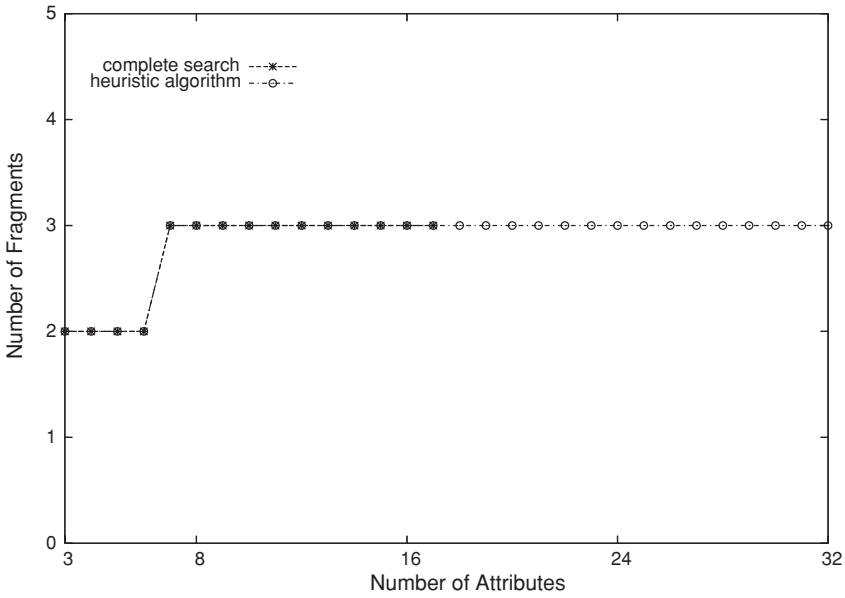

Fig. 14. Number of fragments of the solution computed by the complete search and Function 5.7.

Figure 15 instead compares the affinity of the fragmentation computed through our heuristic algorithm (Function 7.1) with the optimal affinity produced by the complete search strategy. As the graph shows, for all the cases that allow the comparison, the affinity of the solution computed by the heuristic algorithm is close to the optimal value: the average of the difference is 4.7% and the maximum percentage difference is around 14.1%.

Together with the effort that has produced the implementation of the two heuristic algorithms, a parallel development has focused on the integration within a relational database engine of mechanisms supporting the management of partially encrypted relational schemas (Section 8). Like other similar efforts developing novel database services, we chose the PostgreSQL DBMS, typically considered the most complete and sophisticated open-source database engine. The integration of the support for partially encrypted relational tables has required the modification of a few internal components of the PostgreSQL architecture, integrating within the database engine the support for the processing of encryption/decryption functions and adding a few catalog structures. This effort, apart from demonstrating that it is possible to smoothly integrate this data design option within current database systems, has clearly shown the viability of the integration between cryptography and relational structures. The impact on performance of query execution mostly derives in the prototype from the unavailability of the cleartext representation of attributes appearing in selection predicates. Decryption functions were shown to typically exhibit a slight impact on query processing, whose cost is clearly dominated by the costs of other query processing steps.

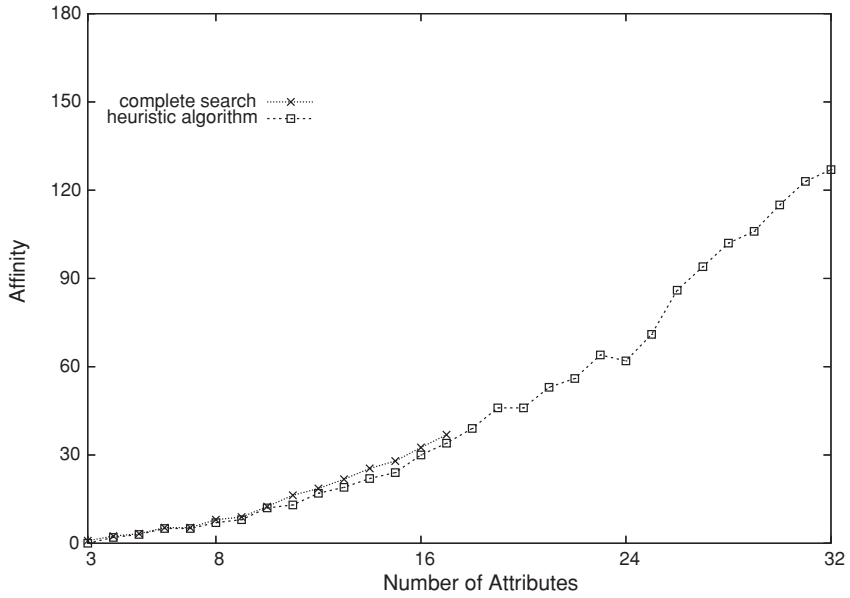

Fig. 15. Affinity of the solution produced by the complete search and Function 7.1.

10. RELATED WORK

A significant amount of research has recently been dedicated to the study of the outsourced data paradigm. Most of this research has assumed the data to be entirely encrypted, focusing on the design of techniques for the efficient execution of queries (Database As a Service paradigm). One of the first proposals toward the solution of this problem is presented in Hacigümüs et al. [2002a, 2002b], where the authors propose storing additional indexing information together with the encrypted database. Such indexes can be used by the DBMS to select the data to be returned in response to a query. In Damiani et al. [2003], the authors propose a hash-based index technique for equality queries, together with a B+ tree technique applicable to range queries. In Wang and Lakshmanan [2006], the authors propose an indexing method that, exploiting B-trees, supports both equality and range queries, while reducing inference exposure thanks to an almost flat distribution of the frequencies of index values. In Ceselli et al. [2005] and Damiani et al. [2003], the authors present different approaches for evaluating the inference exposure for encrypted data enriched with indexing information, showing that even a limited number of indexes can greatly facilitate the task for an attacker wishing to violate the confidentiality provided by encryption.

The first proposal suggesting the storage of plaintext data, while enforcing a series of privacy constraints is presented in Aggarwal et al. [2005]. The main difference with the work proposed in this article is that in Aggarwal et al. [2005], the authors suppose data to be stored on two remote servers, belonging to two different service providers, which never exchange information. This choice also forces to design a fragmentation schema with at most two separate

fragments. The approach presented in our article removes all these restrictions and appears more adequate to the requirements of real scenarios. Our approach may force the use of a greater amount of storage, but in typical environments this presents a smaller cost than that required for the management and execution of queries on remote database servers managed by fully independent third parties.

Other proposals related to our work can be found in Biskup et al. [2008] and Biskup and Lochner [2007], where the authors exploit functional dependencies to the aim of correctly enforcing access control policies. In Biskup and Lochner [2007], the authors propose a policy-based classification of databases that, combined with restriction of the query language, preserves the confidentiality of sensitive information. The classification of a database is based on the concept of classification instance, which is a set of tuples representing the combinations of values that need to be protected. On the basis of the classification instance, it is always possible to identify the set of allowed queries, that is, the queries whose evaluation return tuples that do not correspond to the combinations represented in the classification instance. In Biskup et al. [2008], the authors define a mechanism for defining constraints that reduce the problem of protecting the data from inferences to the enforcement of access control in relational databases.

Our work may bring some resemblance with the work of classifying information while maximizing visibility [Dawson et al. 2002]. However, while the two lines of work share the goal of ensuring protection and minimizing security measures enforcement, the consideration of fragmentation and encryption on the one side and security labeling on the other make the problems considerably different.

The problem of fragmenting relational databases while maximizing query efficiency has been addressed by others in the literature and some approaches have been proposed [Navathe et al. 1984; Navathe and Ra 1989]. However, these approaches are not applicable to our problem since they are only aimed at performance optimization and do not take into consideration protection requirements.

11. CONCLUSIONS

We presented a model and a corresponding concrete approach for the definition and management of privacy requirements in data collections. Our work provides a direct response to the emerging demand by individuals as well as privacy regulators.

Besides the technical contribution, our work can represent a step toward the effective enforcement, as well as the establishment, of privacy regulations. Technical limitations are in fact claimed as one of the main reasons why privacy cannot be achieved and, consequently, regulations not be put into enforcement. Research along the line presented here can then help in providing the building blocks for a more precise specification of privacy needs and regulations, as well as their actual enforcement, together with the benefit of a clearer and more direct integration of privacy requirements within existing ICT infrastructures.

Several open issues still remain to be addressed such as: the protection of fragmented data when the information stored in the fragments may change over time, or the adoption of other strategies for guaranteeing that the published data do not reveal information on the sensitive associations (confidentiality constraints).

REFERENCES

AGGARWAL, G., BAWA, M., GANESAN, P., GARCIA-MOLINA, H., KENTHAPADI, K., MOTWANI, R., SRIVASTAVA, U., THOMAS, D., AND XU, Y. 2005. Two can keep a secret: A distributed architecture for secure database services. In *Proceedings of the Conference on Innovative Data Systems Research*. http://www.cidrdb.org/.

BISKUP, J., EMBLEY, D., AND LOCHNER, J. 2008. Reducing inference control to access control for normalized database schemas. *Inform. Process. Lett. 106*, 1, 8–12.

BISKUP, J. AND LOCHNER, J. 2007. Enforcing confidentiality in relational databases by reducing inference control to access control. In *Proceedings of the 10th International Conference on Information Security*.

CA SB 1386. 2002. California senate bill SB 1386.

CESELLI, A., DAMIANI, E., DE CAPITANI DI VIMERCATI, S., JAJODIA, S., PARABOSCHI, S., AND SAMARATI, P. 2005. Modeling and assessing inference exposure in encrypted databases. *ACM Trans. Inform. Syst. Secur. 8*, 1, 119–152.

CHAUDHURI, S. 1998. An overview of query optimization in relational systems. In *Proceedings of the 17th SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*. ACM, New York.

CIRIANI, V., DE CAPITANI DI VIMERCATI, S., FORESTI, S., JAJODIA, S., PARABOSCHI, S., AND SAMARATI, P. 2007a. Fragmentation and encryption to enforce privacy in data storage. In *Proceedings of the 12th European Symposium on Research in Computer Security*. Springer, Berlin.

CIRIANI, V., DE CAPITANI DI VIMERCATI, S., FORESTI, S., AND SAMARATI, P. 2007b. *k*-Anonymity. In *Secure Data Management in Decentralized Systems*, T. Yu and S. Jajodia, Eds. Springer-Verlag, Berlin.

DAMIANI, E., DE CAPITANI DI VIMERCATI, S., JAJODIA, S., PARABOSCHI, S., AND SAMARATI, P. 2003. Balancing confidentiality and efficiency in untrusted relational DBMSs. In *Proceedings of the 10th Conference on Computer and Communications Security*. ACM, New York.

DAWSON, S., DE CAPITANI DI VIMERCATI, S., LINCOLN, P., AND SAMARATI, P. 2002. Maximizing sharing of protected information. *J. Comput. Syst. Sci. 64*, 3, 496–541.

GAREY, M. R. AND JOHNSON, D. S. 1979. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, New York.

HACIGÜMÜS, H., IYER, B., AND MEHROTRA, S. 2002a. Providing database as a service. In *Proceedings of the 18th International Conference on Data Engineering*. IEEE, Los Alamitos, CA.

HACIGÜMÜS, H., IYER, B., MEHROTRA, S., AND LI, C. 2002b. Executing SQL over encrypted data in the database-service-provider model. In *Proceedings of the SIGMOD International Conference on Management of Data*. ACM, New York.

HOFMEISTER, T. AND LEFMANN, H. 1998. Approximating maximum independent sets in uniform hypergraphs. In *Proceeding of the 23rd International Symposium on Mathematical Foundations of Computer Science*. Springer, Berlin.

KRIVELEVICH, M. AND SUDAKOV, B. 2003. Approximate coloring of uniform hypergraphs. *J. Algorithms 49*, 1, 2–12.

NAVATHE, S., CERI, S., WIEDERHOLD, G., AND DOU, J. 1984. Vertical partitioning algorithms for database design. *ACM Trans. Datab. Syst. 9*, 4, 680–710.

NAVATHE, S. AND RA, M. 1989. Vertical partitioning for database design: A graphical algorithm. In *Proceedings of the SIGMOD International Conference on Management of Data*. ACM, New York.

ÖZSU, M. T. AND VALDURIEZ, P. 1999. *Principles of Distributed Database Systems* 2nd Ed. Prentice-Hall, Upper Saddle River, NJ.

PERSONAL DATA PROTECTION CODE. 2003. Legislative Decree no. 196.
PICDSS. 2006. Payment Card Industry (PCI) data security standard. https://www.pcisecuritystandards.org/pdfs/pci dss v1-1.pdf.
SAMARATI, P. 2001. Protecting respondents' identities in microdata release. *IEEE Trans. Knowl. Data Eng. 13*, 6, 1010–1017.
SCHNEIER, B. 1996. *Applied Cryptography* 2nd Ed. John Wiley & Sons, New York.
TROUESSIN, G., FABRE, J. C., AND DESWARTE, Y. 1991. Reliable processing of confidential information. In *Proceedings of the 7th International Information Security Conference*. Springer, Berlin.
WANG, H. AND LAKSHMANAN, L. V. S. 2006. Efficient secure query evaluation over encrypted XML databases. In *Proceedings of the 32nd International Conference on Very Large Data bases*. ACM, New York.

Received June 2008; accepted July 2009