Fragmentation and Encryption to Enforce Privacy in Data Storage

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

[1] Università degli Studi di Milano, 26013 Crema, Italia
{ciriani,decapita,foresti,samarati}@dti.unimi.it
[2] George Mason University, Fairfax, VA 22030-4444
jajodia@gmu.edu
[3] Università degli Studi di Bergamo, 24044 Dalmine, Italia
parabosc@unibg.it

Abstract. Privacy requirements have an increasing impact on the realization of modern applications. Technical considerations and many significant commercial and legal regulations demand today that privacy guarantees be provided whenever sensitive information is stored, processed, or communicated to external parties. It is therefore crucial to design solutions able to respond to this demand with a clear integration strategy for existing applications and a consideration of the performance impact of the protection measures.

In this paper we address this problem and propose a solution to enforce privacy over data collections by combining data fragmentation with encryption. The idea behind our approach is to use encryption as an underlying (conveniently available) measure for making data unintelligible, while exploiting fragmentation as a way to break sensitive associations between information.

Keywords: Privacy, fragmentation, encryption.

1 Introduction

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

Individuals, privacy advocates, and legislators are today putting more and more attention on the support of privacy over collected information. Regulations are increasingly being established responding to these demands, forcing organizations to provide privacy guarantees over sensitive information when storing, processing or sharing it with others. Most recent regulations (e.g., [2,14]) require

J. Biskup and J. Lopez (Eds.): ESORICS 2007, LNCS 4734, pp. 171–186, 2007.

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) 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" [13] 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.

This demand for encryption is luckily coupled today 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 outsourced scenario [5,8], 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 and therefore encryption is a price to be paid to protect them. This assumption is typically an overkill in many scenarious. As a matter of fact, in many situations 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 paper, 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 [1]. In this 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 [1] suffers from several limitations. The main limitation is that the privacy relies on the complete absence of communication among 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.

In this paper, we propose an approach combining fragmentation and encryption that overcomes the above limitations. Our solution allows storing data on a single server and minimizes the amount of data represented only in encrypted format, therefore allowing for efficient query execution.

We frame our work in the context of relational databases. The reason for this choice is that relational databases 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 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).

Our work assumes that access to data is realized by an application that includes a compact trusted core, which is invoked every time there is the need to access sensitive information (i.e., applying decryption or reconstructing associations by linking fragments). By contrast, the DBMS needs not be trusted, since accessing single fragments or encrypted information does not expose to any privacy breach. This is a considerable advantage over previous proposals, developed, for example, in the data outsourcing scenario [5,8].

The contribution of this paper is threefold. First, we introduce confidentiality constraints as a simple, yet powerful, way to capture privacy requirements. 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. Third, we propose a heuristic algorithm for the concrete identification of a fragmentation solution that satisfies the properties specified.

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 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 the attributes in c should be protected. When the constraint is a singleton set, then the semantics is that the individual attribute must be protected, that is, the list of the attribute values itself is confidential.

While simple, the definition above allows the expression of the different confidentiality requirements that may need to be expressed, such as the following.

- *The values assumed by some attributes are considered sensitive and therefore cannot be stored in the clear*. For instance, phone numbers or email addresses

can be considered sensitive values (even if not associated with any identifying information).
- *The association between values of given attributes is sensitive and therefore should not be released*. For instance, while the list of (names of) patients in a hospital as well as the list of illnesses are by themselves not confidential, the association of patient's names with illnesses is considered sensitive.

Note that constraints specified on the association between attributes can derive from different requirements, as they can correspond to explicit protection of an association (as in the case of names and illnesses above) or to associations that could cause inference on other sensitive information. As an example of the latter, consider a hospital database and 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 [4,15] (i.e., `DoB` and `ZIP` can be used, possibly in association with external information, to help identifying patients and therefore 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 names are not considered sensitive but their association with `Illness` is. Suppose again that `DoB` together with the `ZIP` code can work as a quasi-identifier (then potentially leaking information on names). In this case, an association constraint will be specified protecting the association between `DoB`, `ZIP` code, and `Illness`, implying that the three attributes should never be accessible together in the clear.

In general, we are interested in enforcing a set of *well defined* confidentiality constraints, formally defined as follows.

Definition 2 (Well defined constraints). *A set of confidentiality constraints* $\mathcal{C} = \{c_1,\ldots,c_m\}$ *is said to be* well defined *iff* $\forall c_i, c_j \in \mathcal{C}, i \neq j$, $c_i \not\subset c_j$ *and* $c_j \not\subset c_i$.

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 (or vice versa), the satisfaction of constraint c_i implies the satisfaction of constraint c_j (see Sect. 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 $\mathcal{D}$ a set of domains. A relation schema R is a finite set of attributes $\{a_1,\ldots,a_n\} \subseteq \mathcal{A}$ that are defined on a domain $D_i, i = 1,\ldots,n$. 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 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.

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. 1. An example of plaintext relation (a) and its well defined constraints (b)

For simplicity, and consistently with other proposals [1,15], 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 1. Figure 1 illustrates an example of relation together with some confidentiality constraints on it. The reasons behind the constraints are as follows:

- the list of SSN of patients is considered sensitive (c_0);
- the association of patients' names with any other piece of stored information is considered sensitive ($c_1,\ldots,c_4$);
- DoB and ZIP together can be exploited to infer the name of patients (i.e., they can work as a quasi-identifier), consequently their association with other pieces of information is considered sensitive (c_5,c_6).

Note that also 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). As a matter of fact, 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.

- *Encryption.* 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 [16], we assume that a *salt*, which is a randomly chosen value, is applied on each encryption (similarly to the use of nonces in the protection of messages from replay attacks).
- *Fragmentation.* 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 association among their values is 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, 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 (Fragmentation). *Let R be a relation schema, a* fragmentation *of R is a set of fragments $\mathcal{F}$=$\{F_1,\ldots,F_m\}$, where $F_i \subseteq R$, for $i = 1,\ldots,m$.*

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 in 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. 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 4 (Physical fragment). *Let R be a relation schema, and $\mathcal{F}$=$\{F_1$, $\ldots,F_m\}$ 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(*<u>*salt*</u>*,enc,$a_{i_1},\ldots,a_{i_n}$), where 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 $\otimes$) 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:

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

The algorithm in Fig. 2 shows the construction and population of physical fragments. When the size of the attributes exceeds the size of an encryption block, we assume that encryption of the protected attributes uses a Cipher Block Chaining (CBC) mode [16], with the salt used as the Initialization Vector (IV); in the CBC mode, the clear text of the first block is actually 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 together two or more fragments. In other words, there cannot be attributes that can be

Algorithm 1 (Constraint resolution).

INPUT
A relation r over schema R
$\mathcal{C} = \{c_1, \ldots, c_m\}$ /* well defined constraints */
OUTPUT
A set of physical fragments $\mathcal{F}^e = \{F^e_1, \ldots, F^e_i\}$
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 */
$\mathcal{A}_\mathcal{F} := \{a \in R\text{: } \{a\} \notin \mathcal{C}\}$
$\mathcal{F} :=$ **fragment**$(\mathcal{A}_\mathcal{F}, \mathcal{C}_\mathcal{F})$
/* define physical fragments */
for each $F = \{a_{i_1}, \ldots, a_{i_l}\} \in \mathcal{F}$ **do**
define relation F^e with schema:
$F^e(\underline{salt}, enc, a_{i_1}, \ldots, a_{i_l})$
/* populate physical fragments instances */
for each $t \in r$ **do**
$t^e[salt] :=$ **generatesalt**(F, t)
$t^e[enc] := E_k(t[a_{j_1} \ldots a_{j_p}] \otimes t^e[salt])$ /* $\{a_{j_1} \ldots a_{j_p}\} = R - F$ */
for each $a \in F$ **do** $t^e[a] := t[a]$
insert t^e in f^e

Fig. 2. Algorithm that correctly fragments R

f^e_1

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^e_2

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^e_3

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 relation in Fig. 1(a)

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 attributes appear in clear form in more than one fragment.

The conditions above are formally captured by the following definition.

Definition 5 (Fragmentation correctness). *Let R be a relation schema, $\mathcal{F}$ be a fragmentation of R, and $\mathcal{C}$ a set of well defined constraints over R. $\mathcal{F}$* correctly enforces *$\mathcal{C}$ iff 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. As a matter of fact, as already noted, singleton constraints require the attributes on which they are defined to appear only in encrypted form. Figure 3 illustrates an example of fragmentation of the relation schema in Fig. 1(a) that correctly enforces the well defined constraints in Fig. 1(b).

Original query on R	**Translation over fragment F_3^e**
Q_1 := SELECT SSN, Name FROM MedicalData WHERE Illness='obesity' AND Physician='D. Warren'	Q_1^3 := SELECT salt, enc FROM F_3^e WHERE Illness='obesity'AND Physician='D. Warren' $Q_1^{'}$:= SELECT SSN, Name FROM *Decrypt*(Q_1^3, *Key*)
Q_2 := SELECT SSN, Name FROM MedicalData WHERE Illness='obesity' AND Physician='D. Warren' AND ZIP='94139'	Q_2^3 := SELECT salt, enc FROM F_3^e WHERE Illness='obesity'AND Physician='D. Warren' $Q_2^{'}$:= SELECT SSN, Name FROM *Decrypt*(Q_2^3, *Key*) WHERE ZIP='94139'

Fig. 4. An example of query translation over a fragment

4 Executing Queries on Fragments

Fragmentation of a relation implies that only fragments (which are stored in place of the original relation to satisfy confidentiality constraints) will be available for queries. Note that, 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 a query. However, if the query executed over a fragment involves an attribute that is encrypted, an additional query may need to be executed (after decryption) by the application to evaluate the conditions on the attributes.

We consider generic *select-from-where* SQL queries that present relation R in the *from* clause, specify a conjunction of equality predicates in the *where* clause, and extract a subset of the R's attributes in the *select* clause.

Example 2. Consider the relation in Fig. 1(a) and its fragment F_3^e in Fig. 3(c).

- Consider a query Q_1 retrieving the Social Security Number and the name of the patients whose illness is *obesity* and whose physician is *D. Warren*. Figure 4 illustrates the translation of Q_1 to queries Q_1^3 executed by the DBMS on the fragment, and $Q_1^{'}$ executed by the application. Note that since both `Illness` and `Physician` are represented in the clear in F_3^e, the conditions in the *where* clause can be executed on the fragment itself, thus returning to the application only the tuples belonging to the final result.
- Consider a query Q_2 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*. Figure 4 illustrates the translation of Q_2 to queries Q_2^3 executed

by the DBMS on the fragment, and Q_2' executed by the application. Note that, since ZIP does not appear in the clear in the fragment, the condition on it needs to be evaluated by the application.

The cost of executing a query over a fragment depends on the number of plaintext attributes it contains and on their selectivity. A query optimizer can be used to select the fragment that allows the execution of more selective queries by the DBMS, thus decreasing the workload of the application and maximizing the efficiency of the execution.

5 Minimal Fragmentation

As the examples in Sect 4 have shown, 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 represented formally by the definition of maximal visibility as follows.

Definition 6 (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 *iff* $\forall a \in R$, $\{a\} \notin \mathcal{C}$: $\exists F \in \mathcal{F}$ *such that* $a \in F$.

Note that the combination of maximal visibility together with the second condition of Definition 5 imposes that each attribute that does not appear in a singleton constraint must appear in the clear in exactly one fragment.

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 since it makes the evaluation of a 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 *minimum fragmentation* that is correct and maximizes visibility, while minimizing the number of fragments. This problem is *NP-hard* since it corresponds to the minimum hypergraph coloring problem [7]. It is also interesting to note that, 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 $\epsilon > 0$ [10,17]. We propose therefore a definition of *minimality*, which can be exploited to find an efficient fragmentation through a heuristic (see Sect. 6).

To formally define minimality, we introduce the concept of fragment vector as follows.

Definition 7 (Fragment vector). *Let R be a relation schema, and $\mathcal{F}=\{F_1, \ldots, F_m\}$ be a 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.*

Example 3. Let $\mathcal{F}$ = {{Name},{DoB,ZIP},{Illness,Physician}} be a fragmentation of the relation schema in Fig. 1(a). The fragment vector is the vector $V_\mathcal{F}$ such that:

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

Fragment vectors allow us to define a partial order between fragmentations as follows.

Definition 8 (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 $\mathcal{F} \preceq \mathcal{F}'$, iff $V_\mathcal{F}[a] \subseteq V_{\mathcal{F}'}[a]$, for all $a \in \mathcal{A}$. Consequently, $\mathcal{F} \prec \mathcal{F}'$ iff $\mathcal{F} \preceq \mathcal{F}'$ and $\mathcal{F} \neq \mathcal{F}'$.*

Definition 8 states that solution $\mathcal{F}'$ dominates solution $\mathcal{F}$ if $\mathcal{F}'$ can be computed from $\mathcal{F}$ by merging two (or more) fragments composing $\mathcal{F}$.

Example 4. Let $\mathcal{F}_1$={{Name}, {DoB,ZIP}, {Illness,Physician}} and $\mathcal{F}_2$= {{Name}, {DoB}, {ZIP}, {Illness,Physician}} be two fragmentations of the relation schema in Fig. 1(a). According to Definition 8, $\mathcal{F}_2 \prec \mathcal{F}_1$, since $\mathcal{F}_1$ can be obtained from $\mathcal{F}_2$ by merging fragments {DoB} and {ZIP}.

We can formally define the *minimality* property as follows.

Definition 9 (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* minimal fragmentation *iff all the following conditions are satisfied:*

1. *$\mathcal{F}$ correctly enforces $\mathcal{C}$ (Definition 5);*
2. *$\mathcal{F}$ maximizes visibility (Definition 6);*
3. *$\nexists \mathcal{F}'$ such that $\mathcal{F} \prec \mathcal{F}'$ and $\mathcal{F}'$ satisfies the two conditions above.*

According to this definition of minimality, a fragmentation $\mathcal{F}$ is *minimal* if and only if 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.

Function 1 (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 in 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 in F */
         inserted := true
         break
      endif
   if NOT inserted then /* insert attr in a new fragment */
      add {attr} to F
   endif
until A_ToPlace = ∅
return(F)
```

Fig. 5. Function that finds a minimal fragmentation

Example 5. Consider fragmentations $\mathcal{F}_1$ and $\mathcal{F}_2$ of Example 4, and the set of constraints in Fig. 1(b). Since $\mathcal{F}_2 \prec \mathcal{F}_1$, $\mathcal{F}_2$ is not minimal. By contrast, $\mathcal{F}_1$ is minimal. As a matter of fact, $\mathcal{F}_1$ contains all attributes of relation schema `MedicalData` in Fig. 1(a), but `SSN` (maximize visibility); satisfies all constraints in Fig. 1(b) (correctness); no fragmentation obtained from it by merging any pair of fragments satisfies the constraints.

6 Computing a Minimal Fragmentation

Our heuristic method for computing a minimal fragmentation is based on the **fragment** function illustrated in Fig. 5. This function takes as input a set of attributes *A_ToPlace* to be fragmented, and a set of constraints *C_ToSolve*. It computes a 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 non solved 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** cycle 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 with the highest number of non-solved constraints. The reason for this choice is to bring all constraints to satisfaction in a few number of steps. 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 $A\_ToPlace$ and removes it from $A\_ToPlace$. Then, the function looks for a fragment F in $\mathcal{F}$ in which *attr* can be inserted without violating any constraint including *attr* that has already been solved (indeed, there is no need to check constraints that have not yet been solved). If such a fragment F is found, *attr* is inserted in 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 6. Figure 6 presents the execution, step by step, of function **fragment** applied to the example in Fig. 1. Here, for simplicity, we represent attributes with their initials. The left hand side of Fig. 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 a non solved 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 not solved, 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 non solved constraints. The constraints in $Con[n]$ become now solved, $N\_con[a_i]$ is updated accordingly, and fragment $\{n\}$ is added to $\mathcal{F}$. Function **fragment** proceeds in analogous way by choosing attributes d, z, i, and p. The final solution is represented by the relations in Fig. 3.

The correctness and complexity of our approach are stated by the following theorems, whose complete proofs are omitted here for space constraints.

Theorem 1 (Correctness). *Function* **fragment** *terminates and finds a minimal fragmentation (Definition 9).*

Proof (sketch). The **repeat** loop terminates because $A\_ToPlace$ is finite, and in each iteration an attribute in $A\_ToPlace$ is extracted, and the loop is executed till $A\_ToPlace$ becomes empty. Moreover, all the inner **for** loops always consider a finite set of fragments and constraints. Each attribute *attr* in $A\_ToPlace$ is then inserted exactly in one existing fragment, if no constraint is violated; it is inserted in a new fragment, otherwise (maximal visibility and fragmentation correctness). Moreover, function **fragment** cannot generate two different fragments whose union does not violate any constraint (minimality). In fact, if merging the two

$\mathcal{F}$=∅
$C\_ToSolve$={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 function execution

fragments does not violate any constraint, the function would have inserted all the attributes in the first of the two fragments that was created. ■

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

Proof (sketch). To chose attribute *attr* from *A_ToPlace*, in the worst case the 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 attribute chosen. After the choosing phase, each attribute is inserted in 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 in all the fragments $F\in\mathcal{F}$, and compares $F\cup\{attr\}$ with the constraints. 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 the second phase of function **fragment** is $O(n^2m)$.

Finally, the overall time complexity is therefore $O(n^2m)$. ■

7 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 towards the solution of this problem is presented in [8,9], 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 [5] the authors propose a hash-based index technique for equality queries, together with a B+ tree technique applicable to range queries. In [18] the authors propose an indexing method which, 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 [3,5] 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 [1]. The main difference with the work proposed in this paper is that in [1] 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 paper 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.

Our work may bring some resemblance with the work of classifying information while maximizing visibility [6]. 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 makes 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 [11,12]. However, these approaches are not applicable to our problem since they are only aimed at performance optimization and do not allow taking into consideration protection requirements.

8 Conclusions

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

Besides being a technical contribution, we hope that our work can represent a step towards 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 on the line of ours 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.

Acknowledgements

This work was supported in part by the European Union under contract IST-2002-507591, and by the Italian Ministry of Research, within programs FIRB, under project "RBNE05FKZ2", and PRIN 2006, under project "Basi di dati crittografate" (2006099978). The work of Sushil Jajodia was partially supported by the National Science Foundation under grants CT-0627493, IIS-0242237, and IIS-0430402.

References

1. Aggarwal, G., Bawa, M., Ganesan, P., Garcia-Molina, H., Kenthapadi, K., Motwani, R., Srivastava, U., Thomas, D., Xu, Y.: Two can keep a secret: a distributed architecture for secure database services. In: CIDR 2005. Proc. of the 2nd Conference on Innovative Data Systems Research, Asilomar, California, USA (January 2005)

2. California senate bill SB 1386 (September 2002)
3. Ceselli, A., Damiani, E., De Capitani di Vimercati, S., Jajodia, S., Paraboschi, S., Samarati, P.: Modeling and assessing inference exposure in encrypted databases. ACM Transactions on Information and System Security 8(1), 119–152 (2005)
4. Ciriani, V., De Capitani di Vimercati, S., Foresti, S., Samarati, P.: k-anonymity. In: Yu, T., Jajodia, S. (eds.) Security in Decentralized Data Management, Springer, Heidelberg (2007)
5. Damiani, E., De Capitani di Vimercati, S., Jajodia, S., Paraboschi, S., Samarati, P.: Balancing confidentiality and efficiency in untrusted relational DBMSs. In: CCS03. Proc. of the 10th ACM Conference on Computer and Communications Security, Washington DC, USA, October 2003, ACM Press, New York (2003)
6. Dawson, S., De Capitani di Vimercati, S., Lincoln, P., Samarati, P.: Maximizing sharing of protected information. Journal of Computer and System Sciences 64(3), 496–541 (2002)
7. Garey, M.R., Johnson, D.S.: Computers and intractability: a guide to the theory of *NP*-completeness. W.H. Freeman, New York (1979)
8. Hacigümüs, H., Iyer, B., Mehrotra, S.: Providing database as a service. In: ICDE'02. Proc. of the 18th International Conference on Data Engineering, San Jose, California, USA, IEEE Computer Society, Los Alamitos, California (February 2002)
9. Hacigümüs, H., Iyer, B., Mehrotra, S., Li, C.: Executing SQL over encrypted data in the database-service-provider model. In: Proc. of the 2002 ACM SIGMOD International Conference on Management of Data, Madison, Wisconsin, USA, ACM Press, New York (2002)
10. Krivelevich, M., Sudakov, B.: Approximate coloring of uniform hypergraphs. Journal of Algorithms 49(1), 2–12 (2003)
11. Navathe, S., Ceri, S., Wiederhold, G., Dou, J.: Vertical partitioning algorithms for database design. ACM Transaction on Database Systems 9(4), 680–710 (1984)
12. Navathe, S., Ra, M.: Vertical partitioning for database design: a graphical algorithm. In: Proc. of the 1989 ACM SIGMOD International Conference on Management of Data, Portland, Oregon, USA, ACM Press, New York (June 1989)
13. Payment card industry (PCI) data security standard (September 2006), `https://www.pcisecuritystandards.org/pdfs/pci_dss_v1-1.pdf`
14. Personal data protection code: Legislative Decree no. 196 (June 2003)
15. Samarati, P.: Protecting respondent's privacy in microdata release. IEEE Transactions on Knowledge and Data Engineering 13(6), 1010–1017 (2001)
16. Schneier, B.: Applied Cryptography: protocols, algorithms, and source code in C, 2nd edn. John Wiley & Sons, New York (1996)
17. Hofmeister, T., Lefmann, H.: Approximating Maximum Independent Sets in Uniform Hypergraphs. In: Brim, L., Gruska, J., Zlatuška, J. (eds.) MFCS 1998. LNCS, vol. 1450, Springer, Heidelberg (1998)
18. Wang, H., Lakshmanan, L.V.S.: Efficient secure query evaluation over encrypted XML databases. In: VLDB'06. Proc. of the 32nd International Conference on Very Large Data Bases, Seoul, Korea, ACM Press, New York (September 2006)