2009 29th IEEE International Conference on Distributed Computing Systems

Fragmentation Design for Efficient Query Execution over Sensitive Distributed Databases

Valentina Ciriani*, Sabrina De Capitani di Vimercati*, Sara Foresti*,
Sushil Jajodia †, Stefano Paraboschi‡, and Pierangela Samarati*
*DTI - University of Milan, 26013 Crema - Italy
Email: {ciriani,decapita,foresti,samarati}@dti.unmi.it
†CSIS - George Mason University, Fairfax, VA 22030-4444
Email: jajodia@gmu.edu
‡DIIMM - University of Bergamo, 24044 Dalmine - Italy
Email: parabosc@unibg.it

Abstract

The balance between privacy and utility is a classical problem with an increasing impact on the design of modern information systems. On the one side it is crucial to ensure that sensitive information is properly protected; on the other side, the impact of protection on the workload must be limited as query efficiency and system performance remain a primary requirement. We address this privacy/efficiency balance proposing an approach that, starting from a flexible definition of confidentiality constraints on a relational schema, applies encryption on information in a parsimonious way and mostly relies on fragmentation to protect sensitive associations among attributes. Fragmentation is guided by workload considerations so to minimize the cost of executing queries over fragments. We discuss the minimization problem when fragmenting data and provide a heuristic approach to its solution.

1. Introduction

A medical organization manages a collection of data recording the medical histories of a community of patients. Researchers can then access these data and effectively and efficiently discover behavioral and social patterns that exhibit correlation with specific pathologies, with a direct positive impact on medical research. The downside is that a compromise of the server can disclose patients' information and violate their privacy. The owner of an e-commerce Web site must store the complete description of the financial data about transactions executed on the site. The Web site offers a wider choice and lower prices than a brick-and-mortar store, producing an immediate benefit to consumers and a considerable positive economic impact. The downside is that a compromise of the Web server may bring customers' data into the black market, where they can be used in fraudulent transactions. The two scenarios demonstrate that, while information and communication technology can provide important benefits, they inevitably introduce risks of exposing private information to improper disclosure. The proposal in this paper aims at reducing the risks introduced by the management of sensitive information.

The crucial observation behind our approach is that users of the system may normally need to access the data in a way that does not introduce risks. For instance, medical researchers may typically need to access generic and not-identifying patient data when performing their research. The owner of the Web site mostly accesses the financial data about the transactions managed by the Web site with no need to reference the personal data of the customer. On the other hand, medical researchers may sometimes need to evaluate parameters that may lead to the specific identity of the patient, and the Web site owner may need to retrieve the complete credit card data when a dispute arises. In addition, regulations are forcing requirements on the management of personal information that often explicitly demand the use of encryption for the protection of sensitive data.

A promising approach to protect sensitive data or sensitive associations among data stored at external parties is represented by the combined use of fragmentation and encryption [4]. Fragmentation and encryption provide protection of data in storage, or when disseminated, ensuring no sensitive information is disclosed neither directly (i.e., present in the database) nor indirectly (i.e., derivable from other information in the database). With this design, the data can be outsourced and stored on an untrusted server, typically obtaining lower costs, greater availability, and more efficient distributed access. This scenario resembles the "database-as-a-service" (DAS) paradigm [3], [6] and indeed the techniques presented in the paper can be considered an adaptation of this paradigm to a context where only part of the information stored into the database is confidential and where the confidentiality of associations among values is protected by storing them in separate fragments. The advantage of having only part of the data encrypted is that all the queries that do not require to reconstruct the confidential

1063-6927/09 $25.00 © 2009 IEEE
DOI 10.1109/ICDCS.2009.52

information will be managed more efficiently and securely. This approach represents for the real-world database and security administrators a more interesting solution compared to the canonical full-encryption DAS scenario, where the use of the secret key for each access creates a significant vulnerability.

The combined use of fragmentation and encryption to protect confidentiality has been initially proposed in [1], [4]. However, the proposal in [1] assumes information to be stored on two separate servers and protection relies on the hypothesis that the servers cannot communicate. This assumption is clearly too strong in any practical situation. The proposal in [4] removes this assumption but simply introduces the problem of computing fragmentations. This paper presents an approach for the design of a fragmentation (Section 2) that looks carefully at the performance issues, particularly important in distributed scenarios, and takes into account the profile of the query load on the server (Section 3). We introduce a heuristic algorithm (Section 4) for producing, given a set of confidentiality constraints to be satisfied, a fragmentation design exhibiting good performance. The experimental results (Section 5) support the quality of the solutions produced by the heuristic.

2. Basic concepts

We consider a scenario where, consistently with other proposals (e.g., [1], [9]), the data to be protected are represented with a single relation r over a relation schema $R(a_1, \ldots, a_n)$. When clear from the context, we will use *R* to denote either the relation schema *R* or the set of attributes in *R*. Privacy requirements are represented by *confidentiality constraints*.

Definition 2.1 (Confidentiality constraint): Given a set $\mathcal{A}$ of attributes, a *confidentiality constraint c* over $\mathcal{A}$ is:

1) a singleton set $\{a\} \subset \mathcal{A}$, stating that the values of the attribute are sensitive *(attribute visibility)*; or
2) a subset of attributes in $\mathcal{A}$, stating that the association between values of the given attributes is sensitive *(association visibility)*.

While simple, the confidentiality constraint construct is quite powerful, supporting the definition of different privacy requirements that may need to be expressed.

Example 2.1: Figure 1 shows relation PATIENT and confidentiality constraints over it. Here, c_0 states that the list of SSN of patients is sensitive; c_1 and c_2 state that the associations between Name and Occup, and between Name and Sickness, respectively, are sensitive; c_3 states that the association among Occup, ZIP, and Sickness is sensitive (the rationale is that Occup and ZIP are a quasi-identifier [9] and therefore may allow to derive information on names).

Since the satisfaction of a constraint c_i implies the satisfaction of any constraint c_j such that $c_i \subseteq c_j$, we consider a *well defined* set of constraints $\mathcal{C} = \{c_1, \ldots, c_m\}$, i.e., $\forall c_i, c_j \in \mathcal{C}, i \neq j,\ c_i \not\subset c_j$.

PATIENT

SSN	Name	Occup	Sickness	ZIP
123-45-6789	A. Smith	Nurse	Latex al.	94140
987-65-4321	B. Jones	Nurse	Latex al.	94141
246-89-1357	C. Taylor	Clerk	Latex al.	94140
135-79-2468	D. Brown	Lawyer	Celiac	94139
975-31-8642	E. Cooper	Manager	Pollen al.	94138
864-29-7531	F. White	Designer	Nickel al.	94141

(a)

c_0={SSN}
c_1={Name,Occup}
c_2={Name,Sickness}
c_3={Occup,Sickness,ZIP}

(b)

Figure 1. A plaintext relation (a) and confidentiality constraints (b)

Our approach to satisfy confidentiality constraints combines *encryption* and *fragmentation* techniques. Encryption consists in encrypting all the values of an attribute, thus making them unintelligible to unauthorized users. Fragmentation consists in partitioning attributes in *R* in subsets such that only attributes in the same fragment are visible together. While singleton constraints can only be satisfied via encryption, all the other constraints can be enforced either by encrypting at least one of the attributes involved in the constraint or by splitting the attributes in such a way that their association cannot be reconstructed. Since the availability of attributes in the clear makes query execution efficient, we solve non singleton constraints always via fragmentation.

In this paper, we specifically address the fragmentation problem and therefore focus only on the association (non singleton) constraints $\mathcal{C}_f \subseteq \mathcal{C}$ and on the corresponding set $\mathcal{A}_f$ of attributes to be fragmented, defined as $\mathcal{A}_f = \{a \mid a \in R$ AND $\{a\} \notin \mathcal{C}\}$. The term *fragment* is then used to denote a subset of a set of attributes and a *fragmentation* is a set of fragments defined as follows.

Definition 2.2 (Fragmentation): Given a relation schema *R*, a set $\mathcal{C}$ of well defined constraints, and a set $\mathcal{A}_f \subseteq R$ of attributes to be fragmented, a *fragmentation* of *R* on $\mathcal{A}_f$ is a set of fragments $\mathcal{F}=\{F_1, \ldots, F_r\}$ such that:

1) $\forall F \in \mathcal{F},\ F \subseteq \mathcal{A}_f$;
2) $\forall a \in \mathcal{A}_f,\ \exists\ F \in \mathcal{F}$: $a \in F$;
3) $\forall F_i, F_j \in \mathcal{F}, i \neq j : F_i \cap F_j = \emptyset$.

Condition 1 ensures that only attributes to be fragmented are considered in the fragmentation. Conditions 2 and 3 ensure that a fragmentation is a partition of set $\mathcal{A}_f$. This guarantees: *maximal visibility* (Condition 2), meaning that any attribute not involved in a singleton constraint appears in the clear in at least one fragment; *un-linkability* (Condition 3), meaning that fragments do not have common attributes that could be exploited for linking.

In the following, we denote with $\mathfrak{F}$ the set of all possible fragmentations, and with F_i^j the i-th fragment in fragmentation $\mathcal{F}_j$ (the superscript will be omitted when the fragmentation is clear from the context).

Enc_F$_1$

<u>**salt**</u>	*enc*	**Name**
s_1	α	A. Smith
s_2	β	B. Jones
s_3	γ	C. Taylor
s_4	δ	D. Brown
s_5	ε	E. Cooper
s_6	ζ	F. White

(a)

Enc_F$_2$

<u>**salt**</u>	*enc*	**Occup**
s_7	η	Nurse
s_8	θ	Nurse
s_9	ι	Clerk
s_{10}	κ	Lawyer
s_{11}	λ	Manager
s_{12}	μ	Designer

(b)

Enc_F$_3$

<u>**salt**</u>	*enc*	**Sickness**	**ZIP**
s_{13}	ν	Latex al.	94140
s_{14}	ξ	Latex al.	94141
s_{15}	π	Latex al.	94140
s_{16}	ρ	Celiac	94139
s_{17}	σ	Pollen al.	94138
s_{18}	τ	Nickel al.	94141

(c)

Figure 2. Physical fragments

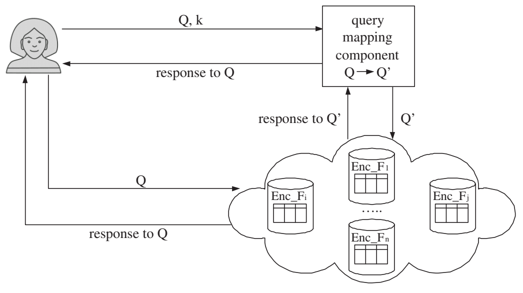


Figure 3. Reference scenario

Each fragment $F_i \in \mathcal{F}$ is physically stored in a relation, denoted *Enc_F*$_i$, called *physical fragment*, defined on the set $\{$<u>*salt*</u>,*enc*,$a_{i_1}, \ldots, a_{i_n}\}$ of attributes, where $a_{i_1}, \ldots, a_{i_n}$ is the set of attributes in F_i and *enc* is the attribute representing the encryption of all attributes in *R* except $a_{i_1}, \ldots, a_{i_n}$ xor-ed with the *salt*, a random value different for each tuple and used for preventing frequency-based attacks over the encrypted values. Physical fragments can be distributed at different servers. Clearly only fragmentations that do not disclose sensitive associations are acceptable, as captured by the definition of *safe fragmentation.*

Definition 2.3 (Safe fragmentation): Given a relation schema *R*, a set $\mathcal{C}$ of well defined constraints over *R*, and a set $\mathcal{A}_f$ of attributes to be fragmented, a fragmentation $\mathcal{F}$ of *R* on $\mathcal{A}_f$ is said to be *safe* iff $\forall F \in \mathcal{F}, \forall c \in \mathcal{C} : c \not\subseteq F$.

A fragmentation is safe if no fragment is a superset of any constraint. We call *unsafe* any fragmentation that is not safe. Figure 2 illustrates a possible set of (physical) fragments for the PATIENT relation in Figure 1(a), which corresponds to the safe fragmentation $\mathcal{F}$={{`Name`},{`Occup`},{`Sickness`, `ZIP`}}.

Figure 3 illustrates the basic scenario we consider. The users that only need to access the cleartext content of a fragment submit the query directly to the server storing the fragment and receive the result with standard approaches. Users authorized to access exact queries involving confidential information submit their query Q, together with the key k needed for decrypting the data, to a trusted query mapping component. This component translates query Q into a query Q' on a physical fragment composing the computed fragmentation. The result is returned to the query mapping component that, if needed, decrypts the tuples and possibly discards spurious tuples. The final plaintext result is then returned to the user.

3. Query cost model

Given a fragmentation $\mathcal{F}$ of *R* on $\mathcal{A}_f$, any query Q can be evaluated on each of the fragments composing $\mathcal{F}$ because the corresponding physical fragments contain all the attributes of *R*, either in encrypted or in clear form. However, the execution cost of a query varies depending on the schema of the physical fragment used for query computation. Overall, with respect to a given query workload, some fragmentations can exhibit a lower cost than others. We are then interested in identifying a safe fragmentation characterized by the lowest cost. To this purpose, we introduce a query cost model for query execution on a fragmentation, which considers a representative set of queries (query workload) as a starting point for the design of a fragmentation.

We describe a query workload $\mathcal{Q}$ as a set $\{Q_1, \ldots, Q_m\}$ of queries, where each query Q_i, $i = 1, \ldots, m$, is characterized by an execution frequency *freq*(Q_i) and is of the form: "SELECT $a_{i_1}, \ldots, a_{i_n}$ FROM *R* WHERE $\bigwedge_{j=1}^{n}$ (a_j IN V_j)" with V_j a set of values in the domain of attribute a_j.

Given a fragment $F_l \in \mathcal{F}$ and a query $Q_i \in \mathcal{Q}$, the cost of executing query Q_i over F_l depends on the set of attributes appearing in F_l and on their selectivity.[1] We estimate the *selectivity* of query Q_i over F_l in terms of the percentage of tuples in F_l that are returned by the execution of query Q_i on F_l, which in turns depends on the selectivity of each single condition in query Q_i. The selectivity of the j-th condition is computed as the ratio of the number of tuples in the fragment such that the value of attribute a_j is a value in V_j, over the number of tuples in F_l ($|R|$). Since we assume that the values of different attributes are distributed independently of each other, the selectivity of $\bigwedge_{j=1}^{n}$ (a_j IN V_j) in query Q_i on fragment F_l, denoted *S*$(Q_i$,$F_l)$, is the product of the selectivity of each single condition. Note that if attribute a_j of the j-th condition does not appear in F_l, the condition cannot be evaluated on F_l and its selectivity is set to 1.

The cost of evaluating query Q_i over fragment F_l, denoted *Cost*$(Q_i$,$F_l)$, is then estimated by the size of the information returned, which is computed by multiplying *S*$(Q_i$,$F_l)$ by the number of tuples in the fragment ($|R|$), and by the size in bytes, denoted *size*(t_l), of the result tuples:

$$Cost(Q_i,F_l) = S(Q_i,F_l) \cdot |R| \cdot size(t_l)$$

Note that *size*(t_l) is obtained by summing the size in bytes of each attribute in the SELECT clause that appears in F_l and the size in bytes of the *enc* attribute of the fragment, if the query requires access to attributes that do not appear in the clear in the fragment and that therefore need to be retrieved

1. We refer to fragments instead of physical fragments because the query cost depends on the attributes appearing in the clear in the physical fragments.

by decrypting attribute *enc*. The final cost of evaluating query Q_i on $\mathcal{F}$ is the minimum among the costs of evaluating the query on each of the fragments in $\mathcal{F}$. In other words, given $\mathcal{F} = \{F_1, \ldots, F_r\}$, the cost of evaluating query Q_i on $\mathcal{F}$ is: $Cost(Q_i,\mathcal{F})$ = Min($Cost(Q_i,F_1)$,...,$Cost(Q_i,F_r)$).

The cost of fragmentation $\mathcal{F}$ with respect to $\mathcal{Q}$ is the sum of the costs $Cost(Q_i,\mathcal{F})$ of each single query Q_i weighted by its frequency, as formally stated as follows.

Definition 3.1 (Fragmentation cost): Given a relation schema *R*, a set $\mathcal{A}_f$ of attributes to be fragmented, a fragmentation $\mathcal{F}$ of *R* on $\mathcal{A}_f$, and a query workload $\mathcal{Q}=\{Q_1, \ldots, Q_m\}$ for R, the cost of $\mathcal{F}$ with respect to $\mathcal{Q}$ is computed as: $Cost(\mathcal{Q},\mathcal{F}) = \sum_{i=1}^{m} freq(Q_i) \cdot Cost(Q_i,\mathcal{F})$

Example 3.1: Consider the fragmentation in Figure 2, where $|R|$ = 6, and query Q: "SELECT $*$ FROM Patient WHERE Sickness='Latex al.' AND Occup='Nurse'". The selectivity of Q on the fragments is: $S(Q,F_1)$=1, since Sickness and Occup do not belong to F_1; $S(Q,F_2)$=2/6, since Occup belongs to F_2 and there are 2 nurses; $S(Q,F_3)$=3/6, since Sickness belongs to F_3 and 3 patients suffer from Latex allergy. Suppose that the size in bytes of the tuples resulting from the evaluation of Q on F_1, F_2, and F_3 is always 1, then $Cost(Q,\mathcal{F})$=Min$(6, 2, 3)$=$Cost(Q,F_2)$=2.

We are interested in finding a safe fragmentation $\mathcal{F}$ that *minimizes* the cost associated with a given query workload.

Problem 3.1 (Minimal fragmentation): Given a relation schema *R*, a set $\mathcal{C}$ of well defined constraints over *R*, a set $\mathcal{A}_f$ of attributes to be fragmented, and a query workload $\mathcal{Q}=\{Q_1, \ldots, Q_m\}$ for *R*, find a *fragmentation* $\mathcal{F}$ of *R* on $\mathcal{A}_f$ such that:

1) $\mathcal{F}$ is a safe fragmentation of *R* on $\mathcal{A}_f$ (Definition 2.3); and
2) $\nexists \mathcal{F}'$ satisfying Condition 1 such that $Cost(\mathcal{Q},\mathcal{F}') < Cost(\mathcal{Q},\mathcal{F})$.

The minimal fragmentation problem is NP-hard since the hitting set problem can be reduced to it in polynomial time by defining an adequate query workload. In the following, we present a heuristic approach for solving it.

4. Fragmentation design

We first characterize the space of possible fragmentations and the relationship among them. We then describe the algorithm for solving Problem 3.1.

4.1. Fragmentation lattice

Since each fragmentation is a partitioning on the set $\mathcal{A}_f$ of attributes, the following partial order relationship can then be defined on fragmentations.

Definition 4.1 (Dominance): Given a relation schema *R*, a set $\mathcal{C}$ of well defined constraints over *R*, a set $\mathcal{A}_f \subseteq$ *R* of attributes to be fragmented, and two fragmentations $\mathcal{F}_i=\{F_1^i, \ldots, F_n^i\}$ and $\mathcal{F}_j=\{F_1^j, \ldots, F_m^j\}$ for *R* on $\mathcal{A}_f$, $\mathcal{F}_j$ *dominates* $\mathcal{F}_i$, denoted $\mathcal{F}_j \succeq \mathcal{F}_i$, iff $\forall a \in \mathcal{A}_f$, the fragment F_k^j containing *a* is a subset of the fragment F_l^i containing *a*.

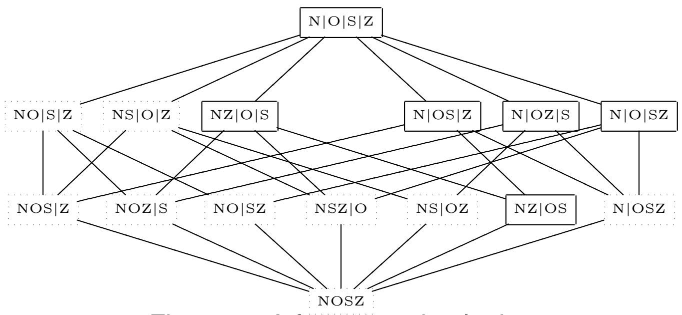


Figure 4. A fragmentation lattice

By Definition 2.2, the fragments composing a fragmentation cannot have common attributes, therefore any solution $\mathcal{F}_i$ directly dominated by $\mathcal{F}_j$ is obtained by merging two fragments in $\mathcal{F}_j$. For instance, fragmentation $\mathcal{F}_1$={{Name},{Occup, ZIP},{Sickness}} directly dominates $\mathcal{F}_2$={{Name, Occup, ZIP},{Sickness}}, which is obtained by merging {Name} and {Occup, ZIP}.

Definition 4.2 (Fragmentation lattice): Given a relation schema *R*, a set $\mathcal{C}$ of well defined constraints over *R*, and a set $\mathcal{A}_f \subseteq R$ of attributes to be fragmented, the *fragmentation lattice* is a pair $(\mathfrak{F}, \succeq)$, where $\mathfrak{F}$ is the set of all fragmentations of *R* on $\mathcal{A}_f$ and $\succeq$ is the dominance relationship among fragmentations as defined in Definition 4.1.

The top element $\mathcal{F}_\top$ of the lattice represents a fragmentation where each attribute in $\mathcal{A}_f$ appears in different fragments. The bottom element $\mathcal{F}_\perp$ of the lattice represents a fragmentation composed of a single fragment containing all attributes in $\mathcal{A}_f$. Figure 4 illustrates the fragmentation lattice for the example in Figure 1. Here, attributes in $\mathcal{A}_f$={Name, Occup, Sickness, ZIP} are represented with their initials (i.e., N, O, S, and Z) and fragments are divided by a vertical line. Also, safe fragmentations (Definition 2.3) are framed by solid boxes and unsafe fragmentations by dotted boxes.

An interesting property of the fragmentation lattice is that given an unsafe fragmentation $\mathcal{F}_j$, any fragmentation $\mathcal{F}_i$ such that $\mathcal{F}_j \succeq \mathcal{F}_i$ is unsafe. Also, the cost of computing queries is *monotonic* with respect to the dominance relationship $\succeq$.[2]

Theorem 4.1: Given a fragmentation lattice $(\mathfrak{F}, \succeq)$, $\forall \mathcal{F}_i, \mathcal{F}_j \in \mathfrak{F}$, $\mathcal{F}_j \succeq \mathcal{F}_i$, $\mathcal{F}_j$ unsafe $\Rightarrow$ $\mathcal{F}_i$ unsafe.

Theorem 4.2 (Monotonicity): Given a relation schema *R*, a set $\mathcal{C}$ of well defined constraints over *R*, a set $\mathcal{A}_f \subseteq R$ of attributes to be fragmented, and a query workload $\mathcal{Q}$ for *R*, $\forall \mathcal{F}_i, \mathcal{F}_j \in \mathfrak{F}$: $\mathcal{F}_j \succeq \mathcal{F}_i \Rightarrow Cost(\mathcal{Q},\mathcal{F}_j) \geq Cost(\mathcal{Q},\mathcal{F}_i)$.

Our modeling of the problem and our solution can be used with any cost function that satisfies the monotonicity

2. Proofs of theorems are omitted from the paper for space constraints.

property, which is a reasonable assumption as query cost should not increase if the number of plaintext attributes in a fragment increases. From Theorem 4.1 and Theorem 4.2, it follows that each path in the lattice is characterized by a *locally minimal fragmentation*, which is the safe fragmentation whose descendants in the path are all unsafe. We propose a heuristic algorithm that partially visits the lattice, following a top-down strategy, to compute a locally minimal fragmentation that, as proved by experimental results, has a cost near to the minimum.

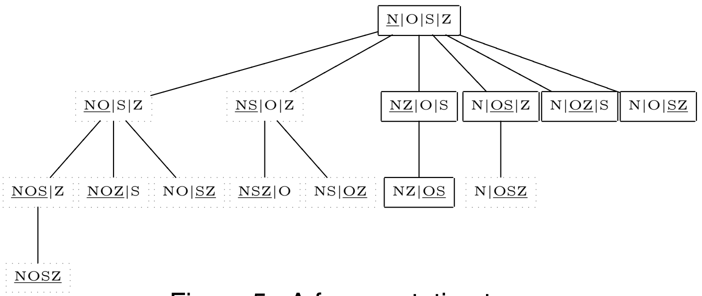


Figure 5. A fragmentation tree

4.2. Fragmentation tree

Our heuristic algorithm is based on the definition of a tree spanning the fragmentation lattice, to visit each fragmentation at most once.

Definition 4.3 (Fragmentation tree): Given a fragmentation lattice $(\mathfrak{F}, \succeq)$, a *fragmentation tree* of the lattice is a spanning tree of $(\mathfrak{F}, \succeq)$ rooted in $\mathcal{F}_\top$.

We now describe a method for building a fragmentation tree over a given fragmentation lattice. For convenience, and without loss of generality, we assume set $\mathcal{A}_f$ to be totally ordered, according to a relationship, denoted $<_A$, and assume that in each fragment F attributes are maintained ordered, from the smallest, denoted F*.first*, to the greatest, denoted F*.last*. We then translate the order relationship among attributes into an order relationship among fragments within a fragmentation, by considering fragments to be ordered according to the order dictated by their smallest (*.first*) attribute. Since, within a fragmentation, each attribute appears in exactly one fragment, the fragments in each fragmentation are totally ordered. Each fragmentation $\mathcal{F}$ is then a sequence $\mathcal{F} = [F_1, \ldots, F_r]$ of fragments, where $\forall i, j = 1, \ldots, r : i < j$, F_i*.first* $<_A$ F_j*.first*. In this case, we say that fragment F_i *precedes* fragment F_j in fragmentation $\mathcal{F}$. Given two fragments F_i, F_j with $i < j$, we say that F_i *fully precedes* F_j iff all attributes in F_i are smaller than all attributes in F_j, that is, F_i*.last* $<_A$ F_j*.first*. Note that the *full precedence* is only a partial ordering.

To avoid computing a fragmentation twice, we associate with each fragmentation $\mathcal{F} = [F_1, \ldots, F_r]$ a *marker* F_i that is the non-singleton fragment such that $\forall j > i$, F_j is a singleton fragment. For the root, the marker is its first fragment. Intuitively, the marker associated with a fragmentation denotes the starting point for fragments to be combined to obtain children of the fragmentation (as a combination with any fragment preceding it will produce duplicate fragmentations). We then define an order-based cover for the lattice as follows.

Definition 4.4 (Order-based cover): Given a fragmentation lattice $(\mathfrak{F}, \succeq)$, an *order-based cover* of the lattice, denoted $\mathcal{T}(V, E)$, is an oriented graph, where $V = \mathfrak{F}$, and $\forall \mathcal{F}_p, \mathcal{F}_c \in V$, $(\mathcal{F}_p,\mathcal{F}_c) \in E$ iff, being F^p_m the marker of $\mathcal{F}_p$, there exists i, j with $m \leq i$ and F^p_i fully preceding F^p_j, such that:

1) $\forall l < j, l \neq i$, $F^c_l = F^p_l$;
2) $F^c_i = F^p_i F^p_j$;
3) $\forall l \geq j$, $F^c_l = F^p_{l+1}$.

An order-based cover for $(\mathfrak{F}, \succeq)$ is a graph with the same vertices as $(\mathfrak{F}, \succeq)$. Each edge $(\mathcal{F}_p,\mathcal{F}_c)$ in the graph represents a dominance relationship $\mathcal{F}_p \succeq \mathcal{F}_c$, where the fragments F^p_i and F^p_j merged to obtain $\mathcal{F}_c$ follow the marker F^p_m and are such that all the attributes in F^p_i precede all the attributes in F^p_j. As an example, consider the order-based cover in Figure 5, where $<_A$ is the lexicographic order, built on the fragmentation lattice in Figure 4. The underlined fragments are the markers. Given fragmentations $\mathcal{F}_p$=[N|O|S|Z] and $\mathcal{F}_c$=[N|OS|Z], edge $(\mathcal{F}_p,\mathcal{F}_c)$ belongs to $\mathcal{T}$ since for $i = 2$ and $j = 3$ we have that F^c_1=F^p_1=N; F^c_2=$F^p_2 F^p_3$=OS; and F^c_3=F^p_{3+1}=Z. The order-based cover so defined corresponds to a fragmentation tree for the lattice, as stated by the following theorem.

Theorem 4.3: The order-based cover $\mathcal{T}$ of a lattice $(\mathfrak{F}, \succeq)$ is a fragmentation tree for $(\mathfrak{F}, \succeq)$ with root $\mathcal{F}_\top$.

A straightforward solution to Problem 3.1 consists in visiting the fragmentation tree $\mathcal{T}$ built over the set $\mathcal{A}_f$ of attributes to be fragmented. This solution, while ensuring a non-redundant evaluation of fragmentations, remains exponential in the number of attributes. While this may not be an issue for small schemas, it may make the algorithm not applicable for complex schemas. We then propose a heuristic algorithm working in polynomial time.

4.3. Heuristic Search

Figure 6 shows our heuristic algorithm for computing a minimal fragmentation. The algorithm takes as input the set $\mathcal{A}_f$ of attributes to be fragmented, the set $\mathcal{C}_f$ of well defined non-singleton constraints, the set Q of queries along with their frequencies, two parameters *d* and *ps* limiting the fragmentations computed by the algorithm, and returns as output a locally minimal fragmentation *Min*.

Basically, the algorithm performs a complete visit on subtrees of depth *d*, covering only a subset of the vertices in the lattice, as shown in Figure 7. The fragmentation lattice

```
INPUT
$\mathcal{A}_f = \{a_1, \ldots, a_n\}$ /* attributes to be fragmented */
$\mathcal{C}_f = \{c_1, \ldots, c_m\}$ /* well defined non-singleton constraints */
$\mathcal{Q} = \{Q_1, \ldots, Q_q\}$ /* queries in the system */
$freq(Q_1), \ldots, freq(Q_q)$ /* relative frequencies of queries */
$d$ /* depth of the subtree completely explored */
$ps$ /* number of subtrees explored at each recursive call */
OUTPUT
$Min$ /* resulting fragmentation */

MAIN
for each $a_i \in \mathcal{A}_f$ do $F_i^{\top} := \{a_i\}$ /* root of the search tree $\mathcal{F}_{\top}$ */
$marker[\mathcal{F}_{\top}] := 1$ /* next fragment to be merged */
$Min := \mathcal{F}_{\top}$ /* current minimal fragmentation */
$MinCost := Cost(\mathcal{Q}, Min)$
$nextqueue :=$ NULL /* priority queue of promising solutions */
$currentqueue :=$ NULL /* queue containing the best $ps$ solutions */
/* compute the best $ps$ solutions within $d$ levels from $\mathcal{F}_{\top}$ */
insert($nextqueue, \langle Min, MinCost \rangle$)
while $nextqueue \neq$ NULL do
    $i := 1$
    while ($i \leq ps$)AND($nextqueue \neq$ NULL) do
        $i := i+1$
        enqueue($currentqueue$, extractmin($nextqueue$))
    $nextqueue :=$ NULL
    while $currentqueue \neq$ NULL do
        $\mathcal{F} :=$ dequeue($currentqueue$)
        $marker[\mathcal{F}] := 1$
        Bsm($\mathcal{F}, d$)

BSM($\mathcal{F}_p, dist$) /* Bounded Search Min */
$localmin := true$ /* minimal safe fragmentation */
for $i = marker[\mathcal{F}_p] \ldots (|\mathcal{F}_p|-1)$ do
   for $j := (i+1) \ldots |\mathcal{F}_p|$ do
      if $F_i^p.last <_A F_j^p.first$ then /* $F_i^p$ fully precedes $F_j^p$ */
        for $l = 1 \ldots |\mathcal{F}_p|$ do
           case:
             ($l < j$ AND $l \neq i$): $F_l^c := F_l^p$
             ($l > j$):             $F_{l-1}^c := F_l^p$
             ($l = i$):             $F_l^c := F_i^p F_j^p$
        $marker[\mathcal{F}_c] := i$
        if SatCon($F_i^c$) then
          $localmin := false$
          if $dist = 1$ then insert($nextqueue, \langle \mathcal{F}_c, Cost(\mathcal{Q}, \mathcal{F}_c) \rangle$)
          else Bsm($\mathcal{F}_c, dist-1$) /* recursive call */
if $localmin$ then
  $cost := Cost(\mathcal{Q}, \mathcal{F}_p)$
  if $cost < MinCost$ then
    $MinCost := cost$
    $Min := \mathcal{F}_p$

SATCON($F$)
for each $c \in \mathcal{C}_f$ do if $c \subseteq F$ then return($false$)
return($true$)
```

Figure 6. Heuristic search algorithm

is then logically partitioned into $\lceil \frac{n}{d} \rceil$ bands, where n is the cardinality of $\mathcal{A}_f$, each containing d levels of vertices. The first subtree of depth d is built considering as root node the top element $\mathcal{F}_{\top}$ of the lattice. At level $i \cdot d$ the algorithm starts ps visits, building a subtree rooted at each of the ps best solutions found at such a level. These visits artificially stop at level $(i+1) \cdot d$, where the best *ps* solutions are chosen as the root for the next in-depth visits of the solution space. To implement this strategy, the algorithm uses two queues: *currentqueue*, containing the best *ps* fragmentations at level $(i-1) \cdot d$ that represent the roots of the subtrees to be visited; and *nextqueue*, containing, in increasing cost order, the safe fragmentations at level $i \cdot d$ computed by the visits of the subtrees rooted at the solutions in *currentqueue*.

In addition, the algorithm uses variables: *marker*[$\mathcal{F}$], representing the position of the marker within fragmentation

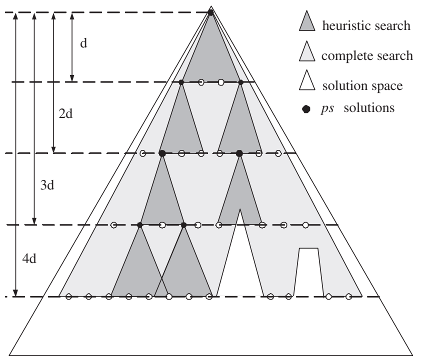


Figure 7. Depiction of the search spaces

$\mathcal{F}$; *Min*, representing the current minimal fragmentation; *MinCost*, representing the cost of *Min*.

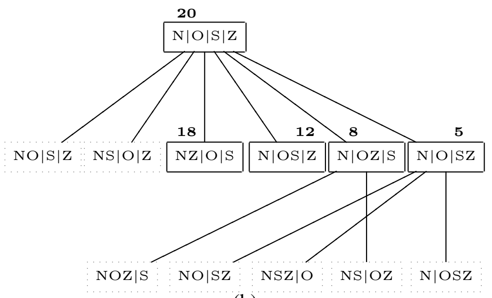
The algorithm first initializes $marker[\top]$ to 1, *Min* to $\mathcal{F}_{\top}$, *MinCost* to the cost of $\mathcal{F}_{\top}$, *currentqueue* and *nextqueue* to NULL, and inserts *Min* into *nextqueue*. At each iteration of the outermost **while** loop, the algorithm copies the first *ps* solutions in *nextqueue* into *currentqueue*, and re-initializes *nextqueue* to NULL. For each $\mathcal{F}$ in *currentqueue*, the algorithm moves the marker of $\mathcal{F}$ to the first fragment and calls procedure **Bsm** (Bounded Search Min) on $\mathcal{F}$. The re-initialization of the marker implies that, for the root fragmentation $\mathcal{F}$ of each subtree, all the fragmentations that represent a child of $\mathcal{F}$ in the lattice are re-evaluated. We note that this strategy could compute more than once the same fragmentation. However, the maximum number of times that a fragmentation can be generated is *ps*. The algorithm terminates when, at the end of an iteration of the outermost **while** loop, variable *nextqueue* is NULL.

Procedure **Bsm** receives as input a fragmentation $\mathcal{F}_p$ and parameter *dist* and iteratively computes the children of $\mathcal{F}_p$ according to Definition 4.4. For each $\mathcal{F}_c$, child of $\mathcal{F}_p$, the procedure verifies whether $\mathcal{F}_c$ satisfies all the constraints (i.e., function **SatCon** returns *true*). If $\mathcal{F}_c$ satisfies the constraints and *dist* is equal to 1, then $\mathcal{F}_c$ is one of the solutions at level $i \cdot d$ and is inserted in *nextqueue*. Otherwise, if $\mathcal{F}_c$ is a safe fragmentation and *dist* is not equal to 1, procedure **Bsm** is recursively called on fragmentation $\mathcal{F}_c$ and distance *dist*-1. Note that for efficiency reasons, the procedure exploits the monotonicity of the cost function (Theorem 4.2) and computes the cost of fragmentation $\mathcal{F}_p$ only if $\mathcal{F}_p$ does not have safe children. In this case, if *Cost*($\mathcal{Q}$,$\mathcal{F}_p$) is less than the current minimal cost *MinCost*, *Min* is set to $\mathcal{F}_p$ and *MinCost* to *Cost*($\mathcal{Q}$,$\mathcal{F}_p$). When all the recursive calls to **Bsm** terminate, *nextqueue* contains the safe fragmentations at level $i \cdot d$, which will be the roots of subtrees visited in the next iteration of the outermost **while** loop of the algorithm.

The computational time of the proposed heuristic is $O(\frac{ps}{d} \cdot n^{2d+2} \cdot |\mathcal{C}_f|)$, since the number of partitions in the fragmentation tree (and therefore of iterations of the **while**

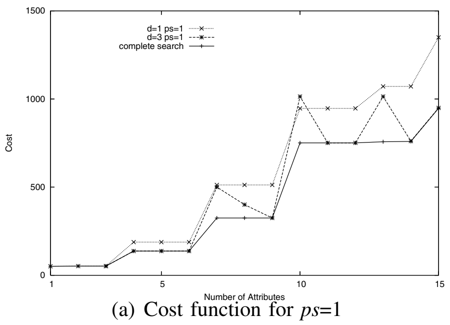
BSM($\mathcal{F}_p$,*dist*)	F^p_i	F^p_j	$\mathcal{F}_c$	**SatCon**(F^c_i)	**BSM**($\mathcal{F}_c$,*dist*)	*Cost*($\mathcal{Q}$,$\mathcal{F}_c$)	*Min*	*nextqueue*
$\underline{\text{N}}$|O|S|Z,1	N	O	$\underline{\text{NO}}$|S|Z	*false*	–			
		S	$\underline{\text{NS}}$|O|Z	*false*	–			
		Z	$\underline{\text{NZ}}$|O|S	*true*	–	18		$\underline{\text{NZ}}$|O|S,18
	O	S	N|$\underline{\text{OS}}$|Z	*true*	–	12		N|$\underline{\text{OS}}$|Z,12
		Z	N|$\underline{\text{OZ}}$|S	*true*	–	8		N|$\underline{\text{OZ}}$|S,8
	S	Z	N|O|$\underline{\text{SZ}}$	*true*	–	5		N|O|$\underline{\text{SZ}}$,5
$\underline{\text{N}}$|O|SZ,1	N	O	$\underline{\text{NO}}$|SZ	*false*	–			
		SZ	$\underline{\text{NSZ}}$|O	*false*	–			
	O	SZ	N|$\underline{\text{OSZ}}$	*false*	–	5	N|O|$\underline{\text{SZ}}$	
$\underline{\text{N}}$|OZ|S,1	N	OZ	$\underline{\text{NOZ}}$|S	*false*	–			
		S	$\underline{\text{NS}}$|OZ	*false*	–			
	OZ	S	–	–	–	8		

(a)

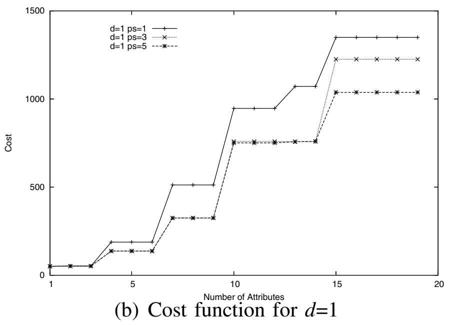


(b)

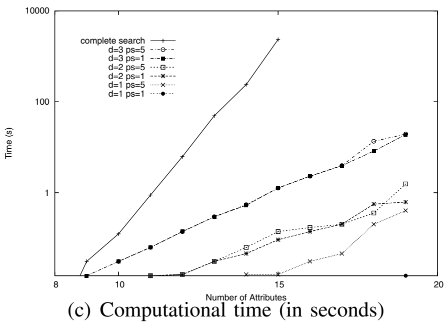
Figure 8. An execution of the heuristic algorithm



(a) Cost function for *ps*=1



(b) Cost function for *d*=1



(c) Computational time (in seconds)

Figure 9. Summary of the experimental results

loop) is $O(\frac{n}{d})$, in each partition the number of subtrees visited (i.e., the number of times **Bsm** is called) is $O(ps)$, and for each partition the heuristic visits $O(n^{2d})$ fragmentations, each of which contains n attributes and is compared with all the constraints in $\mathcal{C}_f$.

Example 4.1: Figure 8 illustrates the execution, step by step, of function **Bsm** applied to the example introduced in Section 2, assuming $d = 1$ and $ps = 2$. The columns of the table in Figure 8(a) represent the call to **Bsm** with its parameter $\mathcal{F}_p$; the fragments F^p_i and F^p_j merged; the resulting fragmentation $\mathcal{F}_c$; the value of **SatCon** on F^c_i; the possible recursive call to **Bsm**($\mathcal{F}_c$,*dist*); the cost of $\mathcal{F}_p$, when computed; the updates to Min; and *nextqueue*. Figure 8(b) illustrates the portion of the fragmentation tree visited by the algorithm. At the beginning variable *Min* is initialized to [$\underline{\text{N}}$|O|S|Z], which is the fragmentation representing the root of the tree, the cost *MinCost* is initialized to 20, and *nextqueue* is initially empty. First, function **Bsm** is called on [$\underline{\text{N}}$|O|S|Z], with *dist*= 1. Since *dist* is 1, the fragmentations generated from [$\underline{\text{N}}$|O|S|Z] and satisfying constraints do not cause a recursive call to **Bsm**, but they are inserted in *nextqueue* after the evaluation of their cost. Then, **Bsm** is called on the first two fragmentations extracted from *nextqueue*, that is, [N|O|$\underline{\text{SZ}}$] and [N|$\underline{\text{OZ}}$|S]. The final fragmentation computed by the heuristic algorithm is [N|O|SZ], which has cost 5 and is represented by the relations in Figure 2.

5. Experimental results

We implemented our algorithm as well as a complete search algorithm for solving Problem 3.1 and compared their results and performances. We considered a health management context and the following configuration: a relational schema of 19 attributes, 12 confidentiality constraints, and 14 queries. The experiments start with a configuration composed of 2 attributes and progressively adds an attribute to the schema. At every step, the experiments consider all and only the constraints and queries that refer to the attributes included in the schema up to that point.

The results of the experiments in terms of query cost are represented in Figures 9(a)(b). Figure 9(a) compares the cost of the solution obtained by a complete search algorithm with the cost of the solution produced by the algorithm in Figure 6 in two configurations: ($d = 1$, $ps = 1$) and ($d = 3$, $ps = 1$). The graph shows that even the simplest configuration ($d = 1$, $ps = 1$) guarantees good-quality fragmentations. Figure 9(b) shows the cost of the solutions produced by the heuristic with $d = 1$ and varying ps (i.e., 1, 3, and 5). It is sufficient to use $ps = 5$ to obtain near-optimum fragmentations.

Figure 9(c) presents the time required for a complete search over the lattice and for our heuristic and shows that, as expected, the complete search algorithm becomes soon unfeasible (a configuration with 15 attributes requires more than 30 minutes). On the other hand, the time required by the heuristics increases exponentially with the increase in *d* and linearly with the increase in *ps*, always showing a limited time for configuration (*d*=1, *ps*=1). It is therefore possible to apply a dynamic approach in solving Problem 3.1, which starts with the most efficient heuristic (*d*=1, *ps*=1) and progressively increases *d* according to the available resources. Parameter *ps* becomes particularly interesting for the implementation of the heuristics on a multi-core architecture, where each core can manage the exploration of one of the alternatives. Overall, the quality of the solutions produced by the heuristic algorithm shows that, even if the problem is computationally hard, it can be managed in real systems.

6. Related work

The outsourced data paradigm has been widely studied [3], [6], [10]. All these works are based on the assumption that data are entirely encrypted and aim at designing techniques for efficient query evaluation. In [1] the authors first introduce a technique for storing plaintext data, while enforcing privacy constraints. Data are split over two non-communicating servers and resorts to encryption any time two fragments are not sufficient for enforcing the constraints. In [4] the authors overcome the assumption that the servers storing data cannot communicate with each other and propose a model that splits the data over different fragments. The main difference between the work in [4] and the work proposed in this paper is that we take into consideration query evaluation cost during the fragmentation process and address the problem of computing a fragmentation that minimizes such a query cost.

The problem of computing a vertical fragmentation that maximizes query efficiency has been addressed in classical distributed databases (e.g., [8]), without taking into account confidentiality constraints. Therefore, these solutions are not applicable to our problem.

Other related proposals can be found in [2], [5], [7] that share with our problem the common goal of enforcing confidentiality constraints on data. However, they are concerned with retrieving a data classification (according to a multilevel mandatory policy) that ensures sensitive information is not disclosed: the consideration of fragmentation and encryption makes the problem completely different.

7. Conclusions

Future information systems are going to show a continuous increase in the level of integration among heterogeneous sources, accessibility from outside actors, and exposure to adversarial behaviors. In this scenario, it is crucial to develop solutions for ensuring that information is properly protected. The formalization of confidentiality constraints, together with the design of an approach for its integration within a database-centered information processing environment, enables the actual enforcement of different privacy requirements that may need to be respected in data storage and dissemination. It therefore promises to become a powerful addition to the collection of tools of the security designer.

Acknowledgements

This work was supported in part by the EU, within the 7FP project, under grant agreement 216483 “PrimeLife”. The work of Sushil Jajodia was partially supported by National Science Foundation under grants CT-0716567, CT-0627493, and IIS-0430402 and by Air Force Office of Scientific Research under grant FA9550-07-1-0527.

References

[1] G. Aggarwal, M. Bawa, P. Ganesan, H. Garcia-Molina, K. Kenthapadi, R. Motwani, U. Srivastava, D. Thomas, and Y. Xu. Two can keep a secret: a distributed architecture for secure database services. In *Proc. of CIDR*, CA, Jan. 2005.

[2] J. Biskup, D.W. Embley, and J. Lochner. Reducing inference control to access control for normalized database schemas. *Information Processing Letters*, 106(1):8–12, 2008.

[3] A. Ceselli, E. Damiani, S. De Capitani di Vimercati, S. Jajodia, S. Paraboschi, and P. Samarati. Modeling and assessing inference exposure in encrypted databases. *ACM TISSEC*, 8(1):119–152, Feb. 2005.

[4] V. Ciriani, S. De Capitani di Vimercati, S. Foresti, S. Jajodia, S. Paraboschi, and P. Samarati. Fragmentation and encryption to enforce privacy in data storage. In *Proc. of ESORICS07*, Dresden, Germany, September 2007.

[5] S. Dawson, S. De Capitani di Vimercati, P. Lincoln, and P. Samarati. Maximizing sharing of protected information. *JCSS*, 64(3):496–541, May 2002.

[6] H. Hacigümüs, B. Iyer, S. Mehrotra, and C. Li. Executing SQL over encrypted data in the database-service-provider model. In *Proc. of SIGMOD 2002*, Madison, WI, June 2002.

[7] S. Jajodia and C. Meadows. Inference problems in multilevel secure database management systems. In *Information Security - An Integrated Collection of Essays*. IEEE CS Press, 1995.

[8] S. Navathe and M. Ra. Vertical partitioning for database design: a graphical algorithm. In *Proc. of SIGMOD 1989*, Portland, OR, June 1989.

[9] P. Samarati. Protecting respondent’s identities in microdata release. *IEEE TKDE*, 13(6):1010–1017, 2001.

[10] H. Wang and L.V.S. Lakshmanan. Efficient secure query evaluation over encrypted XML databases. In *Proc. of the 32nd VLDB Conference*, Seoul, Korea, September 2006.