Available online at www.sciencedirect.com
SCIENCE DIRECT®
Data & Knowledge Engineering 55 (2005) 159–188
DATA & KNOWLEDGE ENGINEERING
www.elsevier.com/locate/datak

Privacy constraint processing in a privacy-enhanced database management system

Bhavani Thuraisingham [1]

University of Texas at Dallas, Department of Computer Science, Richardson, TX, USA

Available online 31 March 2005

Abstract

This paper views the privacy problem as a form of inference problem. It first provides an overview of the privacy problem and then introduces the notion of privacy constraints. Next it describes architecture for a privacy-enhanced database management system and discusses algorithms for privacy constraint processing. A note on privacy constraints processing and release control are given next. Finally some directions for future research on privacy are stated.

Keywords: DBMS; Database; Data management; Privacy; Data-mining; Inference; Architecture

1. Introduction

Privacy is about ensuring that an individual's data and information are not divulged to certain others without the consent of the individual. Privacy control is a set of techniques that ensure that

E-mail addresses: bhavani.thuraisingham@utdallas.edu, thura@mitre.org
URL: www.cs.utdallas.edu/people/thuraisingham.html, www.dr-bhavani.org
[1] Tel.: +1 972 883 4738; fax: +1 972 883 2349.

0169-023X/$ - see front matter © 2005 Published by Elsevier B.V.
doi:10.1016/j.datak.2005.03.001

an individual's privacy is not violated. The privacy problem is the problem of unauthorized release of data that is private. Privacy has been discussed a great deal in the past especially with respect to the disclosure of medical and financial data. Social scientists and policy makers have been discussing privacy for many decades. However it is only recently that privacy has received a lot of attention. This is mainly because of the concern that governments may now use data mining and data analysis tools to extract data and information about individuals for national security purposes (see [19]) and subsequently violate privacy.

In this paper we attempt to provide solutions to the privacy problem. The privacy problem is the problem of determining whether individual privacy can be violated given a set of privacy constraints and a database. We propose the approach of privacy constraint processing to address privacy concerns. The work proposed in this paper follows along the lines of our work on the inference problem over the last several years. Back in 1987 we introduced the notion of security constraints and we discussed an architecture of a database management system augmented with inference engines to handle the inference problem. Our initial work on the inference problem was published in [14] and elaborated in [15]. This work spawned several activities on the inference problem and security constraint processing (see also [16,18]). Similarly in this paper we propose an architecture for what we call a privacy enhanced database system. In this architecture we augment a database system with what we call a privacy controller. The privacy controller processes privacy constraints.

The organization of this paper is as follows. In Section 2 we define a privacy enhanced database management system. In Section 3 we describe the various types of privacy constraints that we have considered. In Section 4 we provide a high level overview of our approach to handling the privacy constraints. In Section 5 we discuss the design of the query processor. In our design, a privacy enhanced database system is augmented with an inference engine (which we can also call a privacy engine). The inference engine has the capability of processing all of the privacy constraints in such a way that certain privacy violations via inference cannot occur. In Section 6 we describe the design of a database update processor, which is responsible for processing certain constraints during database updates. That is, appropriate privacy levels to the data are assigned based on the constraints during the update operation. In Section 7 we describe algorithms that could be utilized by the Systems Privacy Officer (SPO) in order to design the schema of the database. These algorithms handle certain privacy constraints. Some extensions to out current work are given in Section 8. These extensions include privacy control in distributed environments and privacy constraint processing for release control. Section 9 gives summary and directions.

We make an important assumption. We assume that users' roles correspond to privacy levels. That is, depending on a user's role, he or she is essentially cleared at various privacy levels such as Public, Private, or Highly Private. Therefore, when we mean a user at level L, we mean a user cleared at privacy level L and can read all data at or below privacy level L. We also assume that the privacy levels from a hierarchy where public $<$ semi-public $<$ semi-private $<$ private $<$ highly-private. Note that IBM Almaden research center has carried out related work in this area under the hypocritical database project (see [2]). Like the approach we have proposed here, IBM's approach is also related to the security constraint processing work we have carried out and discussed in [14–16].

2. Privacy enhanced database management systems

In a privacy enhanced database management system (PE-DBMS) users with different roles access and share a database consisting of data at different privacy levels. A powerful and dynamic approach to assigning privacy levels to data is one which utilizes privacy constraints. Privacy constraints provide an effective and versatile privacy policy. They can be used to assign privacy levels to the data depending on their content and the context in which the data is displayed. They can also be used to dynamically reclassify the data. In other words, the privacy constraints are essential for describing privacy enhanced applications. An architecture for a PE-DBMS is illustrated in Fig. 1.

Handling privacy constraints in privacy enhanced database systems is following along the lines of security constraint processing (see for example, [14,9,15]). The security constraints that we identified included those that classify data based on content, context, aggregation, and time. The work reported in [14,9] suggest ways of handling security constraints during query processing in such a way that certain security violations via inference does not occur. The work reported in [12,7] focuses on handling constraints during database design where suggestions for database design tools are given. They expect that security constraints during database design are handled in such a way that security violations cannot occur. While the previous papers focus on security constraints, we have adapted the approach for privacy constraints in our present paper. We describe the design techniques for processing privacy constraints. We believe that appropriate handling of privacy constraints is essential for developing a useful PE-DBMS.

From an analysis of the various types of privacy constraints, we believe that they are a form of integrity constraints enforced in a PE-DBMS. This is because, in a privacy enhanced database one can regard the privacy level of an entity to be part of the value of that entity. Therefore privacy constraints specify permissible values that an entity can take. Since a privacy constraint can be regarded as a form of integrity constraint, many of the techniques developed for handling integrity constraints in non-privacy enhanced relational database systems by the logic programming researchers could be used for handling privacy constraints in a PE-DBMS. In these techniques, some integrity constraints, which are called derivation rules, are handled during query processing, some integrity constraints, known as integrity rules, are handled during database updates, and some integrity constraints, known as schema rules, are handled during database design

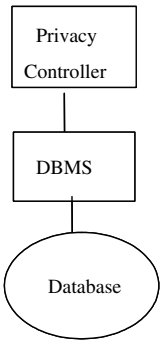

Fig. 1. Privacy enhanced database management system.

(see [6]). Our approach to handling privacy constraints has been influenced by the approach taken to process integrity constraints by the logic programming researchers [10].

Before designing a constraint processor, a question that must be answered is whether a constraint should be processed during query processing, during database updates or during database design. When constraints are handled during query processing, they are treated as a form of derivation rules. That is, they are used to assign privacy levels to the data already in the database before it is released. In other words, new information (e.g., the privacy labels) is deduced from information already in the database. When the privacy constraints are handled during update processing, they are treated as a form of integrity rules. That is, they are constraints that must be satisfied by the data in the privacy enhanced database. When the constraints are handled during database design, then they must be satisfied by the database schema in the same way functional and multivalued dependency constraints must be satisfied by the schema of a relational database.

We believe that it is essential for the query processor to have the capability of processing the privacy constraints. This is because most users usually build their reservoir of knowledge from responses that they receive by querying the database. It is from this reservoir of knowledge that they infer private information. Moreover, no matter how well the database has been designed with respect to privacy, or the data in the database is accurately labeled with privacy labels, users could eventually violate privacy by inference because they are continuously updating their reservoir of knowledge as the world evolves. It is not feasible to have to re-design the database or to re-classify the data continuously. It should however be noted that processing a large number of privacy constraints could have an impact on the performance of the query processing algorithms. Therefore it is desirable to process as many constraints as possible during database updates and during database design. This is because, in general, the database design and the update operation are performed less frequently than the query operation. Therefore, when the database is designed initially, the privacy constraints should be examined and the schema should be generated. Whenever the data is updated, the constraints are examined and the privacy levels are assigned or re-assigned to the affected data. Periodically the System Privacy Officer (SPO) should examine the privacy constraints and re-design the database and/or re-classify the data. If the application is static and if the data in the database is consistent with the privacy constraints, the query processor need not examine the constraints handled by the update processor and the database designer. If there is some change that has occurred in the real world, which makes the database or the schema inconsistent, then the query processor should be triggered so that it can process the relevant constraints during its operation. This way, much of the burden placed on the query processor is alleviated.

In this paper we first define various types of privacy constraints. We then discuss our approach to handling the privacy constraints during query processing, during database updates, and during database design. This is followed by a discussion of three distinct approaches to processing privacy constraints. In the first approach, the privacy constraints are handled during query processing. We describe the design of a query processor, which uses privacy constraints for query modification as well as response processing. In the second approach, certain privacy constraints are handled during database updates. We describe the design of an update processor, which uses privacy constraints to assign privacy levels to the data updated. In the third approach certain privacy constraints are handled during database design. We describe an algorithm, which uses privacy constraints for generating the schema of the privacy enhanced database.

A question that arises with constraint processing is the consistency and completeness of the privacy constraints. We regard constraints to be horn clauses. The difference between a set of horn clauses and a rule base has been well documented. Furthermore, various techniques for ensuring the completeness and consistency of horn clause programs have been developed (see, for example, [10]). These techniques could be utilized for privacy constraints specified as horn clauses also. While more research needs to be done, the work of logic programmers show much promise on ensuring the completeness as well as verifying the consistency of privacy constraints expressed as horn clauses.

There are two aspects to constraint handling. One is constraint generation and the other is constraint enforcement. The constraint generation process analyzes the specification of a privacy enhanced application and generates an initial set of schema and constraints. This is typically performed by an Applications Specialist possibly together with the SPO. Constraint enforcement, on the other hand, is involved with techniques for enforcing the constraints that are produced by the constraint generation task. This is because, for certain applications privacy classification guidelines may not exist. In such a situation it may be difficult for the application specialist to generate the constraints. Our research has focused only on constraint enforcement. We assume that the application specialist (or the SPO) would generate an initial set of privacy constraints. The techniques that we have developed would ensure that these constraints are processed effectively. Since constraint generation is a task that cannot be overlooked eventually, our future work will include the investigation of developing techniques for privacy constraint generation.

Note that much of the work presented in this paper focuses on processing privacy constraints during query time and before the response is released. Recently there has been some work on examining the release information and determining whether the data can be released (see for example, [20]). We briefly examine release control for privacy constraint processing in Section 8. We also discuss extensions to out approach to distributed environments in Section 8.

3. Privacy constraints

Privacy constraints are rules, which assign privacy levels to the data. They can be used either as integrity rules, derivation rules or as schema rules (such as data dependencies). If they are used as integrity rules, then they must be satisfied by the data in the privacy-enhanced database. If they are used as derivation rules, they are applied to the data during query processing. If they are used as data dependencies, they must be satisfied by the schema of the privacy-enhanced database.

We have defined various types of privacy constraints. They include the following:

(i) Constraints that classify a database, relation or an attribute. These constraints are called simple constraints.
(ii) Constraints that classify any part of the database depending on the value of some data. These constraints are called content-based constraints.
(iii) Constraints that classify any part of the database depending on the occurrence of some real-world event. These constraints are called event-based constraints.
(iv) Constraints that classify associations between data (such as tuples, attributes, elements, etc.). These constraints are called association-based constraints.

(v) Constraints that classify any part of the database depending on the information that has been previously released. These constraints are called release-based constraints. We have identified two types of release-based constraints. One is the general release constraint which classifies an entire attribute depending on whether any value of another attribute has been released. The other is the individual release constraint which classifies a value of an attribute depending on whether a value of another attribute has been released.

(vi) Constraints that classify collections of data. These constraints are called aggregate constraints.

(vii) Constraints which specify implications. These are called logical constraints.

(viii) Constraints which have conditions attached to them. These are called constraint with conditions.

(ix) Constraints that classify any part of the database depending on the privacy level of some data. These constraints are called level-based constraints.

(x) Constraints which assign fuzzy values to their classifications. These are called fuzzy constraints.

We will give examples of constraints belonging to each category. In our examples, we assume that the database consists of two relations EMPLOYEE and MEDICAL where EMPLOYEE has attributes E#, ENAME, MANAGER, and M# (with E# as the key), and MEDICAL has attributes M#, RECORDS, and INSURANCE (with M# as the key). Note that M# in EMPLOYEE and M# in MEDICAL take values from the same domain. The constraints may be expressed as some form of logical rules. We have chosen horn clauses to represent the constraints. This way we could eventually take advantage of the techniques that have been developed for logic programs.

Simple constraints: $R(Al, A2, \ldots, An) \rightarrow Level(Ail, Ai2, \ldots, Ait) = Private$.

(Each attribute $Ail, Ai2, \ldots, Ait$ of relation R is Private).

Example: EMPLOYEE(E#, ENAME, MANAGER, M#) → Level(MANAGER) = Private.

Content-based constraints: $R(Al, A2, \ldots, An)$ AND $COND(Value(Bl, B2, \ldots, Bm)) \rightarrow Level(Ail, Ai2, \ldots, Ait) = Private$.

(Each attribute $Ail, Ai2, \ldots, Ait$ of relation R is Private if some specific condition is enforced on the values of some data specified by $Bl, B2, \ldots, Bm$).

Example: EMPLOYEE(E#, ENAME, MANAGER, M#) AND (Value(ENAME) = Washington) → Level(MANAGER) = Private.

Association-based constraints (also called context or together constraints): $\mathbf{R(A1, A2, \ldots, An)} \rightarrow Level(Together(Ail, Ai2, \ldots, Ait)) = Private$.

(The attributes $Ail, Ai2, \ldots, Ait$ of relation R taken together are Private).

Example: EMPLOYEE(E#, NAME, MANAGER, M#) → Level(Together(ENAME, MANAGER)) = Private.

Event-based constraints: $R(Al, A2, \ldots, An)$ AND $Event(E) \rightarrow Level(Ail, Ai2, \ldots, Ait) = Private$ (Each attribute $Ail, Ai2, \ldots, Ait$ of relation R is Private if event E has occurred).

Example: EMPLOYEE(E#, ENAME, MANAGER, M#) AND Event(Change of President) → Level(MANAGER, M#) = Private.

General release-based constraints: $R(Al, A2, \ldots, An)$ AND Release(Ai, Public) → Level(Aj) = Private.

(The attribute Aj of relation R is Private if the attribute Ai has been released at the Public level).

Example: EMPLOYEE(E#, ENAME, MANAGER, M#) AND Release(ENAME, Public) → Level(MANAGER) = Private.

Individual release-based constraints: R(Al, A2, . . ., An) AND Individual-Release(Ai, Public) → Level(Aj) = Private. The individual release-based constraints classify elements of an attribute at a particular level after the corresponding elements of another attribute have been released. They are more difficult to implement than the general release-based constraints. In our design, the individual release-based constraints are handled after the response is assembled while all of the other constraints are handled before the response is generated.

Aggregate constraints: Aggregate constraints classify collections of tuples taken together at a level higher than the individual levels of the tuples in the collection. There could be some semantic association between the tuples. We specify these tuples in the following form:

R(Al, A2, . . ., An) AND Set(S, R) AND Satisfy(S, P) → Level(S) = Private.

This means that if R is a relation and S is a set containing tuples of R and S satisfied some property P. then S is classified at the Private level. Note that P could be any property such as "number of elements is greater than 10."

Logical constraints: Logical constraints are rules which are used to derive new data from the data in the database. The derived data could be classified using one of the other constraints. Logical constraints are of the form: Ai ⇒ Aj; where Ai and Aj are attributes of either a database relation or a real-world relation.

Note that logical constraints are not really privacy constraints. That is, they do not assign privacy levels to the data. They are in fact integrity constraints. In particular, they can be regarded as integrity constraints which are treated as derivation rules.

Constraints with conditions: An example of a constraint with a condition is a constraint-based constraint. Other constraint such as association-based constraints and logical constraint can also be specified with conditions.

Consider the following example: Ai ⇒ Aj if condition C holds.

This constraint can be instantiated as follows:

The INSURANCE of AN EMPLOYEE implies his RECORDS if the INSURANCE IS AAA.

Other constraints: There are several other types of constraints which could be incorporated into our design fairly easily. These include level-based constraints and fuzzy constraints. We describe them below.

Level-based constraints: **R(Al, A2, . . ., An) AND Level(Ai)** = Public → Level(Aj) = Private.

(The attribute Aj of relation R is Private if the attribute Ai is Public).

Example: EMPLOYEE(E#, ENAME, MANAGER, M#) AND Level(ENAME) = Public → Level(MANAGER) = Private.

Fuzzy constraints: Fuzzy constraints are constraints, which use fuzzy values. They can be associated with any of the other types of constraints. An example of a fuzzy constraint which is associated with a content-based constraint is given below.

R(Al, A2, . . ., An) AND COND(Value(Bl, B2, . . ., Bm)) → Level(Ail, Ai2, . . ., Ait) = Private and Fuzzyvalue = r.

(Each attribute Ail, Ai2, . . ., Ait of relation R is Private with a fuzzy value of r if some specific condition is enforced on the values of some data specified by B1, B2, . . ., Bm).

Example: EMPLOYEE(E#, ENAME, MANAGER, M#) AND (Value(ENAME) = Washington) → Level(MANAGER) = Private and Fuzzyvalue = 0.8.

Complex constraints: The examples of constraints that we have given above are enforced on a single relation only.

Note that constraints can also be enforced across relations. We call such constraints complex constraints. An example is given below:

Rl(Al, A2, ..., An) and R2(Bl, B2, ..., Bm) and Rl.Ai = R2.BjLevel(Together(Ak, Bp)) = Private.

This constraint states that pair of values involving the kth attribute of Ri and the pth attribute of R2 are Private provided the corresponding values (i.e. in the same row) of the ith attribute of Ri and the jth attribute of R2 are equal.

This constraint may be instantiated as follows:

EMPLOYEE(E#, ENAME, MANAGER, M#) and MEDICAL(M#, RECORDS, INSURANCE) and EMPLOYEE.M = MEDICAL.M# → Level(Together(ENAME, RECORDS)) = Private.

4. Approach to privacy constraint processing

4.1. Integrated approach

As stated in Section 1, privacy constraints enforce a privacy policy. Therefore it is essential that constraints be manipulated only by an authorized individual. In our approach constraints are maintained by the SPO. That is constraints are protected from ordinary users. We assume that constraints themselves could be classified at privacy levels. However they are stored at system-high. The constraint manager, which is trusted will ensure that a user can read the constraints classified only at or below his level.

Our approach to privacy constraint processing is to handle certain constraints during query processing, certain constraints during database updates and certain constraints during database design. The first step was to decide whether a particular constraint should be processed during the query, update or database design operation. After some consideration, we felt that it was important for the query processor to have the ability to handle all of the privacy constraints. As stated in Section 2, our thesis is that inferences can be most effectively handled, and thus prevented during query processing.

The next step was to decide which of the privacy constraints should be handled during database updates. After some consideration, we felt that except for some types of constraints such as the release and aggregate constraints, the others could be processed during the update operation. However, techniques for handling constraints during database updates could be quite complex as the privacy levels of the data already in the database could be affected by the data being updated. Therefore, initially our algorithms handle only the simple and content-based constraints during database updates.

The constraints that seemed appropriate to handle during the database design operation were those that classified an attribute or collections of attributes taken together. These include the simple and association-based constraints. For example, association-based constraints classify the relationship between attributes. Such relationships are specified by the schema and therefore such constraints could be handled when the schema is specified. Since a logical constraint is a rule

which specifies the implication of an attribute from a set of attributes, it can also be handled during database design.

Note that some constraints can be handled in more than one way. For example, we have the facility to handle the content-based constraints during query processing as well as during database updates. However, it may not be necessary to handle a constraint in more than one place. For example, if the content-based constraints are satisfied during the database update operation, then it may not be necessary to examine them during query processing also. Furthermore, the query operation is performed more frequently than the update operation. Therefore, it is important to minimize the operations performed by the query processor as much as possible to improve performance. However, there must be a way to handle all of the constraints during query processing. This is because, if the real-world is dynamic, then the database data may not satisfy all of the constraints that are enforced as integrity rules, or the schema may not satisfy the constraints that are processed during database design. This means that there must be a trigger which informs the query processor that the privacy enhanced database or the schema is not consistent with the real-world, in which case the query processor can examine the additional constraints.

Below we briefly illustrate the architectures for processing constraints during the query, update, and database design operations. The architecture for query processing is shown in Fig. 2. This architecture can be regarded as a loose coupling between a privacy enhanced relational database management system and a deductive manager. The deductive manager is what we have called the query processor. It has to operate on-line. The architecture for update processing is shown in Fig. 3. This architecture can also be regarded as a loose coupling between a privacy enhanced relational database management system and a deductive manager. The deductive manager is what we have called the update processor. It could be used on-line where the privacy levels of the data are determined during database inserts and updates, or it could be used off-line as a tool that ensures that data entered via bulk data loads and bulk data updates is accurately labeled, if the tool is used off-line, however, it may be difficult to recomputed the levels of the data already in the

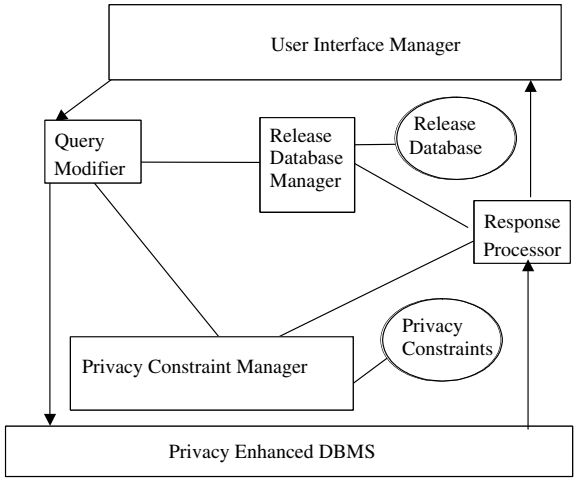

Fig. 2. Query processor.

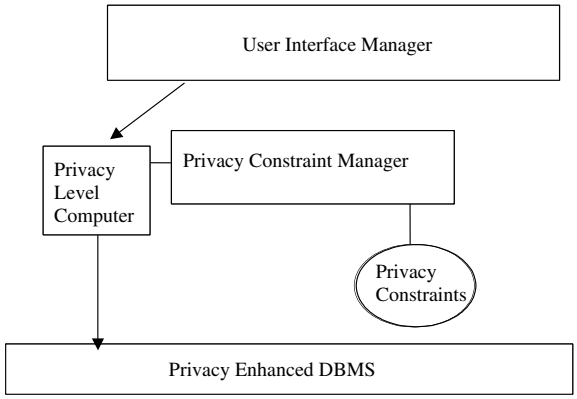

Fig. 3. Update processor.

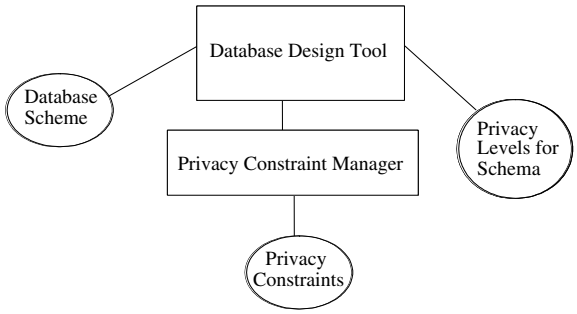

Fig. 4. Database design tool.

database if these levels are affected by the new data that is being inserted. The tool, which handles privacy constraints during database design, illustrated in Fig. 4, can be used by the SPO to design the schema. The input to the tool is the set of privacy constraints that should be handled during database design and the schema. The output of the tool is the modified schema and the constraints.

4.2. Consistency and completeness of the constraints

In Section 2 we described the two tasks involved in constraint handling. They are constraint generation and constraint enforcement. While our main focus is on constraint enforcement, the relationship between the two tasks is illustrated in Fig. 5. That is, the constraint generator takes the specification of the privacy-enhanced application and outputs the initial schema and the constraints that must be enforced. The database design tool takes this output as its input and designs the database. The update processor and the query processor use the constraints and schema produced by the database design tool. Generating the initial set of constraints remains a challenge. We need to examine the use of conceptual structure and semantic data models to see if we can specify the application and subsequently generate the privacy constraints.

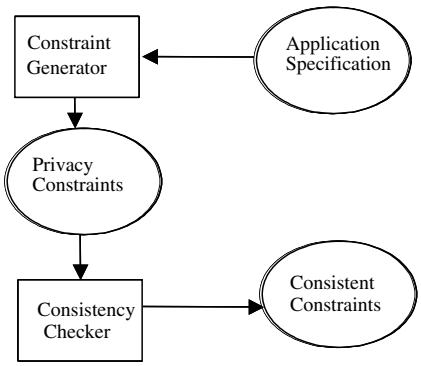

Fig. 5. Constraint generation.

Algorithm A checks for the consistency and completeness of the privacy constraints.

Algorithm A: Consistency and completeness checker.
Input: Set of Privacy Constraints.
Output: Set of consistent and complete privacy constraints.

Step 1: For each relation and attribute compute the security level of the relation and attribute.
Step 2: If there are constraints that assign multiple privacy levels to a relation or attribute then delete/modify the constraint that assigns the lower privacy level; the constraint is modified if that constraint assigns privacy levels to other attributes and relations.
Step 3: If there is some relation or attribute that is not assigned a privacy level by the constraints, then create a constraint that will assign the lowest privacy level to the relation or attribute.
Step 4: The resulting set of constraints is the output.

4.3. Integrated architecture

Although the query processor, update processor, and database design tool are separate modules, they all constitute the solution to constraint processing in privacy enhanced relational databases. That is, these three approaches provide an integrated solution to privacy constraint processing in a privacy enhanced environment.

Fig. 6 illustrates the integrated architecture. In this architecture, the constraints and schema which are produced by the constraint generator are processed further by the database design tool. The modified constraints are given to the Constraint Updater in order to update the constraint database. The schema is given to the Constraint Generator. The constraints in the constraint database are used by the query and update processors. We assume that there is a trusted constraint manager processes, which manages the constraints. In a dynamic environment where the data and the constraints are changing, then the query processor will examine all the relevant constraints and ensure that users do not obtain unauthorized data.

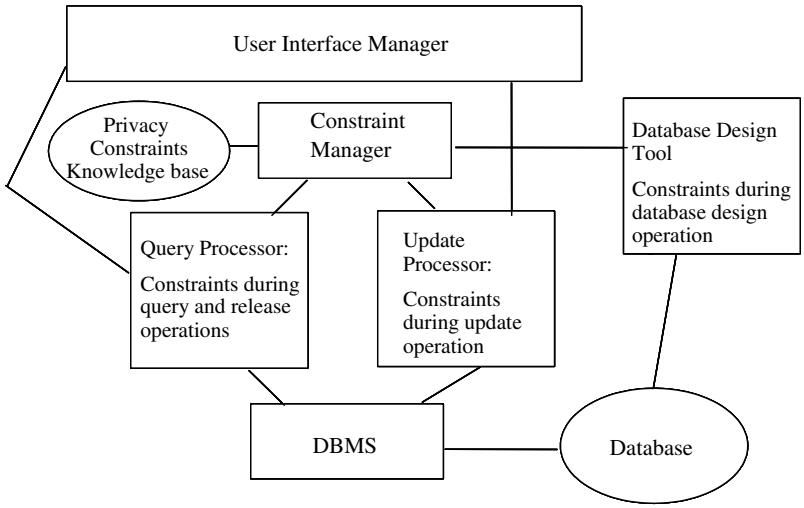

Fig. 6. Integrated architecture.

5. Design of the query processor

We first describe a privacy policy for handling inferences during query processing and then discuss our design approach.

5.1. Privacy policy

A privacy policy for query processing that we propose extends the simple privacy property in [3] to handle inference violations. This policy is stated below (see also [1]).

1. Given a privacy level L, E(L) is the knowledge base associated with L. That is, E(L) will consist of all responses that have been released at privacy level L over a certain time period and the real world information at privacy level L.
2. Let a user U at privacy level L pose a query. Then the response R to the query will be released to this user if the following condition is satisfied: For all privacy levels L* where L* dominates L, if (E(L*) UNION R) $\Rightarrow$ X (for any X) then L* dominates Level(X). Where A $\Rightarrow$ B means B can be inferred from A using any of the inference strategies and Level(X) is the privacy level of X.

We assume that any response that is released into a knowledge base at level L is also released into the knowledge bases at level L* >= L. The policy states that whenever a response is released to a user at level L, it must be ensured that any user at level L* >= L cannot infer information classified at a level L + >= L from the response together with the knowledge that he has already acquired. Note that while we consider only hierarchical levels in specifying the policy, it can be extended to include non-hierarchical levels also.

5.2. Functionality of the query processor

The strength of the query processor depends on the type of inference strategies that it can handle. In our design we consider only a limited set of inference strategies such as inference through association and deduction. In this section, we discuss the techniques that we have used to implement the privacy policy. They are: query modification and response processing. Each technique is described below.

5.2.1. Query modification

Query modification technique has been used in the past to handle discretionary security and views [13]. This technique has been extended to include mandatory security in [5]. In our design of the query processor, this technique is used by the inference engine to modify the query depending on the privacy constraints, the previous responses released, and real world information. When the modified query is posed, the response generated will not violate privacy.

Consider the architecture for query processing illustrated in Fig. 2. The inference engine has access to the knowledge base which includes privacy constraints, previously released responses, and real world information. Conceptually one can think of the database to be part of the knowledge base. We illustrate the query modification technique with examples. The actual implementation of this technique could adapt any of the proposals given in [6] for deductive query processing.

Consider a database which consists of relations EMPLOYEE and MEDICAL where the attributes of EMPLOYEE are E#, ENAME, MANAGER and M# with E# as the key; and the attributes of MEDICAL are At#, RECORDS and INSURANCE with M# as the key. Let the knowledge base consists of the following rules:

1. EMPLOYEE(X, Y, Z, A) and Z = Smith → Level(Y, Private).
2. EMPLOYEE(X, Y, Z, A) and A = 10 → Level(Y, HighlyPrivate).
3. EMPLOYEE(X, Y, Z, A) → Level((Y, Z), Private).
4. EMPLOYEE(X, Y, Z, A) and Release(Z, Public) → Level(Y, Private).
5. EMPLOYEE(X, Y, Z, A) and Release(Y, Public) → Level(Z, Private).
6. NOT(Level(X, Private) or Level(X, HighlyPrivate)) → Level(X, Public).

The first rule is a content-based constraint, which classifies an employee name whose manager is Smith at the Private level. Similarly, the second rule is also a content-based constraint which classifies an employee name whose M# number is 10 at the HighlyPrivate level. The third rule is an association-based constraint which classifies employee names and managers taken together at the Private level. The fourth and fifth rules are additional restrictions that are enforced as a result of the context-based constraint specified in rule 3. The sixth rule states that the default classification level of a data item is Public.

Suppose a Public user requests the employee names in EMPLOYEE. This query is represented as follows: EMPLOYEE(X, Y, Z, D).

Since a employee name is classified at the Private level if either the manager is ''Smith'' or the manager name is already released at the Public level, and it is classified at the HighlyPrivate level if

the M# is "10", assuming that the manager names are not yet released to a Public user, the query is modified to the following:

EMPLOYEE(X, X, Z, D) and NOT (Z = Smith and D = 10).

Note that since query modification is performed in real-time, it will have some impact on the performance of the query processing algorithm. However, several techniques for semantic query optimization have been proposed for intelligent query processing (see, for example, [11]). These techniques could be adapted for query processing in a privacy enhanced environment in order to improve the performance.

5.2.2. Response processing

For many applications, in addition to query modification, some further processing of the response such as response sanitization may need to be performed. We will illustrate this point with examples.

Example: Consider the following release constraints discussed earlier. That is,

(i) all employee names whose corresponding manager names are already released to Public users are Private, and
(ii) all manager names whose corresponding employee names are already released to Public users are Private.

Suppose a Public user requests the employee names first. Depending on the other constraints imposed, let us assume that only certain names are released to the user. Then the employee names released have to be recorded into the knowledge base. Later, suppose a Public user (does not necessarily have to be the same one) asks for manager names. The manager name values (some or all) are then assembled in the response. Before the response is released, the employee names that are already released to the Public user need to be examined. Then the manager name value which corresponds to a employee name value that is already released is suppressed from the response. Note that there has to be a way of correlating the employee names with the managers. This means the primary key values (which is the E#) should also be retrieved with the manager names as well as be stored with the employee names in the release database.

Example: Consider the following aggregate constraint.
A collection of 10 or more tuples in the relation EMPLOYEE is Private.

Suppose a Public user requests the tuples in EMPLOYEE. The response is assembled and then examined to see if it has more then 10 tuples. If so, it is suppressed.

There are some problems associated with maintaining the release information. As more and more relevant release information gets inserted, the knowledge base could grow at a rapid rate. Therefore efficient techniques for processing the knowledge base need to be developed. This would also have an impact on the performance of the query processing algorithms. Therefore, one solution would be to include only certain crucial release information in the knowledge base. The rest of the information can be stored with the audit data, which can then be used by the SPO for analysis.

5.3. Modules of the query processor

We considered an architecture where a privacy enhanced relational database system was augmented with an inference engine. Such an architecture would be useful as the privacy enhanced relational database system would ensure the enforcement of a basic mandatory privacy policy. The inference engine then needs to implement only the policy extensions which are enforced in order to handle inferences.

We assume that the constraints are maintained by the SPO. Constraints themselves could be classified at different levels. However they are stored at system-high. The constraint manager, which is a trusted process, will ensure that a constraint classified at level L can only be read by a user cleared at level L or higher.

The query processor consists of five modules P1 through *P5*. Each module may be considered as a process. A brief overview of the functions of each module is given below. We also identify the trust that must be placed on each process.

Process P1: The User Interface Manager. This process asks for password and privacy level from the user. We rely on the identification and authentication mechanism provided by the operating system to ensure that the user is authenticated correctly. Due to this feature, P1 need not be a trusted process (note that by a trusted process we mean a process that must be verified to operate correctly). It operates at the user's level. P1 accepts a query from the user and performs syntax check. It then sends the query to process P2 and returns the response received from P2 to the user. It then waits in idle state for a request from the user.

Process P2: The Central Privacy Controller. This process first sets up communication with P1. It then waits in idle state for requests from P1. When a request arrives from P1, it logs into the database server at the user's level. It then requests process P3 to return applicable constraints. The query is then modified based on the constraints (if any). The modified query is then sent to the PE-DBMS. The response is then sent to process P4 for further processing. When P4 returns the sanitized response, a request is sent to process P5 to update the release database and the response is given to P1. P2 then returns to idle state. If constraints classified at a higher level are not processed by P3 or if the response from the PE-DBMS is first given to P4 and P5 for sanitization and release database update, then P2 need not be a trusted processes. However, in our design, since P2 could have access to higher privacy level information it must be trusted.

Process P3: The Constraint Gatherer. Process P3 examines not only the privacy constraints classified at or below the user's level, but also higher level constraints. These higher level constraints are examined to ensure that by releasing a response at level L, it is not possible for users cleared at a higher privacy level to infer information to which they are not authorized. P3 builds and maintains the constraint table whenever the constraints are updated. It waits in idle state for requests from P2. When a request arrives, it builds a list of applicable constraints and sends the constraint structure to P2 and then returns to idle state. Since P3 maintains the privacy constraints it is a trusted process.

Process P4: The Sanitizer. This process waits in idle state for a request to arrive from P2. When a request arrives, which consists of the response and the applicable release constraints, it sanitizes the response based on the information that has previously been released. It reads the release database maintained at various levels in order to carry out the sanitization process. It then returns the

sanitized response to P2 and returns to idle state. Since response sanitization is a privacy critical function, P4 must be trusted.

Process P5: The Release Database Updater. This process waits in idle state for requests from P2. When a request arrives, it logs into the database server at all levels from system-high to the user's level and updates the release database at each level depending on the release constraints for that level. Note that this is necessary only if higher level constraints are examined by P3. If not, P5 can log into the PE-DBMS only at the user's level. After each update to the release database, it logs out of the database server at each level. It returns a status message to P2 upon completion and returns to idle state.

5.4. Algorithms for the query processor

Algorithm B describe query processing.

Algorithm B: Query Processing.
Input: Query, Constraints, User's privacy level.
Output: Response.

Step 1: Only consider the constraints that directly or by implication assign privacy levels to the data that the user cannot read related to attributes specified in the query. Let this set of constraints be G.

Step 2: For every attribute specified in the query, find all the constraints in G that apply to the attribute. If the constraint is a simple constraint, then modify the query to not retrieve the value of the attribute. For example if the constraint says that all medical records are private then modify the query not to retrieve medical records. If the constraint is a content constraint, then modify the query where the value of the attribute is not the value in the where clause of the query. For example if the medical records of John are private then modify the query not to retrieve the medical records of John.

Step 3: Next for every attribute in the query check whether the values of the attributes are private (or some level the user cannot read) by deduction. If so, then modify the query as so to not retrieve the values of the attribute. Example, if prescription medication records imply the medical conditions and medical conditions are private, then do not release prescription records.

Step 4: Check for general release constraints in G for attributes in the query. For example, if medical records have been released, then do not release any medical conditions. If a release constraint applies to an attribute, then modify the query so as to not retrieve the values of the attribute.

Step 5: Process the query and retrieve the response.

Step 6: Check for special release constraints and modify the response. E.g., if medical records of John have been released, then do not release further medical conditions of John. Then examine the release constraints further and record that certain attributes have been released.

Step 7: The response computed in step 6 is given to the user.

6. Design of the update processor

We first discuss the privacy policy for database updates and then describe the modules.

6.1. Privacy policy

PE-DBMSs ensure the assignments of a privacy level to data as data is inserted or modified. The privacy level assigned to the data, however, is generally assumed to be the login privacy level of the user entering the data. A more powerful and dynamic approach to assigning privacy levels to data is through the utilization of privacy constraints during update operations.

The privacy policy of the Update Processor is formulated from the simple privacy property in [3] and from a privacy policy provided by the underlying PE-DBMS. This policy is as follows:

1. All users are granted a maximum clearance level with respect to privacy levels. A user may log in at any level that is dominated by his maximum clearance level. Subjects act on behalf of users at the user's login privacy level.
2. Objects are the rows, tables, and databases, and every object is assigned a privacy level upon creation.
3. A subject has read access to an object if the privacy level of the subject dominates the privacy level of the object.
4. A subject has write access to an object if the privacy level of the object dominates the privacy level of the subject.

Statements 3 and 4 of the policy presented above are the simple and *-property of the Bell and La Padula policy.

6.2. Functionality of the update processor

The Update Processor utilizes simple and content-dependent privacy constraints as guidance in determining the privacy level of the data being updated. The use of privacy constraints can thereby protect against users incorrectly labeling data as a result of logging in at the wrong level, against data being incorrectly labeled when it is imported from systems of different modes of operation such as a system high, and against database inconsistencies as a consequence of the privacy label of data in the database being affected by data being entered into the database.

The privacy level of an update request is determined by the Update Processor as follows. The simple and content-dependent privacy constraints associated with the relation being updated and with a privacy label greater than the user login privacy level are retrieved and examined for applicability. If multiple constraints apply, the privacy level is determined by the constraint that specifies the highest classification level, if no constraints apply, the update level is the login privacy level of the user. The Update Processor, therefore, does not determine the privacy level of the data solely from the privacy constraints, but utilizes the constraints as guidance in determining the level of the input data. The following examples illustrate the functionality of the Update Processor.

Consider a database that consists of a relation EMPLOYEE whose attributes are employee number, name, class, date, and MEDICAL, with medical number as its primary key. The

content-based constraint which classifies all EMPLOYEE with name Josephine as private is expressed as:

EMPLOYEE.ename = "Josephine" → Private.

A user at login privacy level say semi-public enters the following data to insert a tuple into the EMPLOYEE relation:

Insert EMPLOYEE values ("SSN 729", "James", "Thomsen", "MR1800"). That is the employee number for James is SSN729, his manager is Thomsen, his medical record number is MR1800.

The Update Processor will receive this insert and retrieve the constraints associated with the EMPLOYEE relation which specify a level greater than the user level, which is semi-public, and whose level is less than or equal to the user level. The content-based constraint stated above is retrieved. Since the data entered for the name field is not "Josephine", the privacy constraint associated with the EMPLOYEE relation will not affect the classification level of the insert, and the Update Processor will determine the insert level to be the user level, which is semi-public.

Suppose a user at login privacy level semi-public then enters the following: Insert EMPLOYEE values ("SSN 730", "Josephine", "Jane", "MR2100"). The Update Processor will again retrieve the content-based constraint associated with the EMPLOYEE relation, which specifies a level greater than the user level and whose level is less than or equal to the user level. Since the data for the name field is "Josephine", the Update Processor will determine the insert level to be private. If, however, the user entered this insert at login privacy level Highlyprivate, the Update Processor would perform the insert at the user level since the user level is higher than the level specified by the privacy constraint.

The update operation of the Update Processor functions similarly to the insert operation. As an example, suppose a user at the semi-public level enters the following: Update EMPLOYEE set ename = "Josephine" where manager = "Thomsen". The Update Processor will retrieve the privacy constraints associated with the EMPLOYEE relation which specify a level greater than the user level and whose level is less than or equal to the user level. The content-dependent constraint stated above will be retrieved, and the Update Processor will determine the update level to be private since the name field is being modified to "Josephine". The tuple with a primary key of "SSN 729" as defined above will then be updated at the private level, and the original tuple will be deleted.

In addition to describing the functionality of the Update Processor, the examples above illustrate the potential signaling channels that exist when operating with the Update Processor. A signaling channel is a form of covert channel, which occurs when the actions of a high privacy-level user or subject interfere with a low privacy-level user or subject in a visible manner. Potential signaling channels occur when data is entered at a level higher than the user level and the user attempts to retrieve the data that he has entered, or when the Update Processor attempts to enter data at a higher level, but cannot since a tuple with the same primary key already exists at this level.

6.3. Utilization of the update processor

A PE-DBMS provides assurance that all objects in a database have a privacy level associated with them and that users are allowed to access only the data which they are cleared. Additionally,

it provides a mechanism for entering privacy enhanced data but relies on the user to login at the level at which the data is to be entered. The Update Processor will provide a mechanism that can operate as a standalone tool with a PE-DBMS to provide assurance that data is accurately labeled as it is entered into the database. This could significantly enhance and simplify the ability of a PE-DBMS to assure that data entered via bulk data loads and bulk data updates are accurately labeled.

Another significant use for an Update Processor is in operation with a Query processor which functions during query processing. The Query processor protects against certain privacy violations via inference that can occur when users issue multiple requests and consequently infer unauthorized knowledge. The Query processor also utilizes privacy constraints as its mechanism for determining the privacy level of data. The privacy constraints are used as derivation rules as they are applied to the data during query processing. Addressing all of the privacy constraint types mentioned above could add a significant burden to the query processor particularly if the number of constraints is high. To enhance the performance of the query processor, the Update Processor can be utilized to address certain constraint types as data is entered into the database, in particular, simple and content-based constraints, alleviating the need for the query processor to handle these constraint types. We assume that the privacy constraints remain relatively static, as reliance on the Update Processor to ensure that data in the database remains consistent would be difficult, particularly in a volatile environment where the constraints change dynamically. An additional concern is that database updates could leave the database in an inconsistent state. The Update Processor, however, is designed to reject updates that cause a rippling effect and thus leave the database in an inconsistent state.

6.4. Modules of the update processor

The Update Processor handles the simple and content-based constraints. The constraints may be represented as a structure or stored in the database itself. The constraints may be assigned different privacy levels, but stored at system-high (which is the highest privacy level). The owner of the constraint structure or relation is the SPO. Therefore only the SPO can manipulate the constraint table. The constraint manager, which is a trusted process, would ensure that only a user classified at level L could read the constraints classified at or below level L.

The Update Processor employs a process structure similar to the Query processor to allow for integration with the Query processor. Process P1 provides a similar user interface to process P1 of the Query processor. Additionally, the functionality of process P2 could be integrated with the functionality of process P2 of the Query processor. However, processes P3 and P4 of the Update Processor must not be confused with processes P3 and P4 of the Query processor. Processes P3 and P4 are specific to the Update Processor. Should the Update Processor be integrated with the Query processor in an implementation, processes P3 and P4 of the Update Processor must remain unique processes.

Process P1: User Interface Manager. The User Interface Manager process, process P1, provides a user interface to the Update Processor prototype. At start-up, P1 prompts the user for a password and privacy level. The level specified by the user is the level at which this process runs. Next, P1 prompts the user for a database request and remains in idle until it receives a request upon receiving a request, P1 logs into the PE-DBMS using the user's userid, password, and clearance.

The request is then sent to the server for a syntax check. The server returns a message indicating the result of the syntax check. If successful, communication is established with process P2, the Update Processor Controller, and the request, along with the login packet that contains the userid, login privacy level, and password of the user, is routed to P2. P1 then remains waiting for a response, which will indicate the success or failure of the transaction from P2. Once a response is received from P2, P1 will display the response to the user, and P1 will again prompt the user for another request If the user chooses to enter a request at a different login privacy level, process P1 will have to be restarted, at which point the user will again be prompted for a password and clearance.

The User Interface Manager operates as the front-end to the Update Processor and does not perform privacy-critical operations. By design, it is isolated from the operations of the higher level processes. As a result, P1 is an untrusted process and, as mentioned above, operates at the user's level.

Process P2: Update Processor Controller. The Update Processor Controller manages the flow of information between the Update Constraint Gatherer (process P3), the Level Upgrader (process P4), and the PE-DBMS in determining the level of the update and in performing the update. Upon start-up, P2 idles, waiting for a request from P1. Upon receiving the login packet and the request, P2 logs into the PE-DBMS utilizing the userid, password, and clearance in the login packet. Thus, P2 runs at the user level. The Update Processor controller then examines the request to determine if it is a select, an insert, an update, or a delete. If the request is an insert or an update, some preliminary processing is performed on the request, and the request along with the login packet is sent to P3. P2 remains idle, waiting for a response, which will contain the insert/update level from P3. If the level determined by P3 is greater than the user level, P2 invokes P4 to perform the insert/update. P2 then idles, waiting for a success or failure response from P4. If the level determined by P3 is the user level, then P2 sends the request to the PE-DBMS to perform the insert/update. The PE-DBMS returns a completion status message to P2, indicating whether the transaction completed or failed. P2 then sends this completion status message to P1 and waits for the next request from P1. The Update Processor Controller provides assurance that the connection to the PE-DBMS is established at the correct level, that the user's request is not modified, and that the level determined by P3 is either the level at which the update is performed or the level sent to P4. As such, the Update Processor Controller is a trusted process.

Process P3: Update Constraint Gatherer. The Update Constraint Gatherer is responsible for determining the privacy level of the data utilizing the applicable privacy constraints. Since P3 must have access to the constraints that are stored in the CONSTRAINT table, which is defined at system high, P3 runs at system high. At start-up, P3 waits for a request from P2. Upon receiving a request, P3 determines the privacy level of the insert or update, utilizing the algorithms described above. P3 then sends this level to P2 and idles, waiting for another request.

Since the Update Constraint Gatherer determines the level at which the insert/update will be performed, assurance must be provided that the applicable constraints are used and that the level determined by this process is accurate. This process, therefore, is a trusted process.

Process P4: Level Upgrader. The Level Upgrader is the process that issues the request to the PE-DBMS at the level determined by P3 when the insert/update level determined by P3 is greater than the user level. (Note: P2 runs at the user level.) At start-up, P4 waits for a request from P2. Upon receiving the level from P2, P4 logs into the PE-DBMS at this level and sends the request to

the server. The response from the server is examined, and the completion status message is sent to P2. P4 then idles, awaiting another request from P2. The Level Upgrader provides assurance that the level at which it requests the PE-DBMS to perform the insert/update is the level received from P2. P4 is therefore a trusted process.

6.5. Algorithm for update processing

Algorithm C describes update processing.

Algorithm C: Update Processing.
Input: Update Request, Constraints.
Output: Privacy level of the data.

Step 1: Only consider the constraints that directly or by implication assign privacy levels to data that the user cannot read related to attributes specified in the query. Let this set of constraints be G.
Step 2: For data to be updated, compute the privacy levels using the relevant constraints in G. For example, if the medical records of John are Private, then John's medical records are assigned private while the other records are public.
Step 3: Enter the data at the appropriate privacy level computed in step 2.
Step 4: Give a status to the user.

7. Handling privacy constraints during database design

7.1. Overview

The main focus of this section is a discussion on how association-based constraints (also called together or context-based constraints), could be handled during database design. We then briefly discuss how simple constraints as well as logical constraints could be handled.

An association-based constraint classifies a collection of attributes taken together at a particular privacy level. What is interesting about the association-based constraint is that it can generate several relationships between the various attributes. For example, if there is a relation EMPLOYEE whose attributes are E#, ENAME, and MANAGER, and if an association-based constraint classifies the ENAME and MANAGER taken together at the Private level, then one of the pairs (E#, ENAME), (E#, MANAGER) should also be classified at the Private level. Otherwise, a Public user can obtain the (E#, ENAME) and the (E#, MANAGER) pairs and infer the Private association (ENAME, MANAGER).

We first describe an algorithm, which processes a given set of association-based constraints and outputs the schema for the privacy-enhanced database. Given a set of association-based constraints and an initial schema, the algorithm will output clusters of attributes and the privacy level of each cluster. We then prove that the attributes within a cluster can be stored securely at the corresponding level. A tool based on this algorithm can help the systems privacy officer (SPO) design the privacy-enhanced database. The algorithm that we have designed does not necessarily

have to be executed during database design only. It can also be executed during query processing. That is, the query processor can examine the attributes in the various clusters generated by the algorithm and then determine which information has to be released to the users. For example, if the algorithm places the attributes Al, A2 in cluster 1 at level L, and the attributes A3, A4 in cluster 2 at level L, then, after an attribute in cluster 1 has been released to a user at level L, none of the attributes in cluster 2 can be released to users at level L.

Since simple constraints can be regarded as a special form of association-based constraints, where only one attribute is classified, we feel that such constraints could also be handled during database design. Another constraint that could be handled during database design is the logical constraint. For example, if attribute A implies an attribute B, and if attribute B is classified at the Private level, then attribute A must be classified at least at the Private level. It should be noted that if any of the constrains have conditions attached to them, then handling them during database design time would be difficult For example, consider the following constraint: ''ENAME and INSURANCE taken together are Private if INSURANCE is from company X''. Such a constraint depends on data values. Therefore, they are best handled during either query or update processing.

The organization of this paper is as follows. In Section 7.2 we describe an algorithm, which determines the privacy levels of the attributes given a set of association-based constraints. A tool could be developed based on this algorithm, which the SPO could use to design the schema. In Section 7.3 we describe how simple constraints could be handled during database design. In Section 7.4 we discuss how logical constraint may be processed. Algorithms are summarized in Section 7.5.

7.2. Handling association-based constraints

In this section we describe an algorithm for handling association-based constraints. The input to this algorithm is a set of association-based constraints and a set of attributes. The output of this algorithm is a set of clusters for each privacy level. Each cluster for a privacy level L will have a collection of attributes that can be safely classified at the level L. That is, if Al, A2, and A4 are attributes in a cluster C at level Private, then the attributes Al, A2, and A3 can be classified together safely at the privacy level Private without violating privacy. The clusters are formed depending on the association-based constraints, which are input to the program. Once the clusters are formed, then the database can be defined according to the functional and multivalued dependencies that are enforced.

Begin (Algorithm)

Let C be the set of privacy constraints and $A1, A2, \ldots, An$ be the set of attributes which are input to the program.

For each privacy level L, do the following:

Begin

Let C[L] be the largest subset of C and $A = \{Al, A2, \ldots, An)$ be the largest subset of $\{Al, A2, \ldots, An)$, such that the elements of subset of C and A are all visible at level L.

Since n is the number of attributes which are visible at level L, clusters $Cl, C2, \ldots, Cn$ will be formed as follows:

Set Cl = C2 = C3 = ··· = Cn = Empty-set.
For each i (1 ⇐ i ⇐ n) do the following:
Begin
Find the first cluster Cj (1 ⇐ j ⇐ n) such that Ai, together with any of the attributes already in Cj, is classified at a level dominated by L by the set of constraints C[L].
Place Ai in the cluster Cj. (Note that since we have defined n clusters, there will definitely be one such Cj.)
End (for each i).
Output all the non-empty clusters along with the security level L.
End (for each security level L).
End (Algorithm).

7.3. A note on simple constraints

Since simple constraints classify individual attributes at a certain privacy level, they could also be handled during database design. Note that when an attribute A in relation R is classified at level L, then all elements which belong to A is also classified at level L. Therefore, we can store A itself at level L.

The algorithm which handles simple constraint is straightforward. Each attribute that is classified by a simple constraint is stored at the level specified in the constraint. Once the algorithm for processing simple constraints is applied and the corresponding schema is obtained, then this schema is given as input to the algorithm handling association-based constraints. The association-based constraints are then applied and the final schema is obtained.

We illustrate the combined algorithm with an example.

Let relation R have attributes Al, A2, A3, and A4. Let the constraints enforced be the following:

Simple constraint: A4 is Private.

Association-based constraint: A2 and A3 together are HighlyPrivate.

Applying the algorithm for handling simple constraints we obtain the following.

Al, A2 and A3 are Public; Al, A2, A3, and A4 are Private.

Next we apply the algorithm for handling association-based constraints. The final output is:

Al and A2 are Public; Al, A2, and A4 are Private; Al, A2, A3, and A4 are HighlyPrivate.

7.4. Handling logical constraints

Logical constraints are rules that can be used to deduce new data from existing data. If a privacy constraint classifies the new data at a level that is higher than that of the existing data, then the existing data must be re-classified. Logical constraints could be straightforward such as Ai ⇒ Aj or they could be more complex such as Al and A2 and A3 and An ⇒ Am. If Aj is classified at the Private level then Ai must be classified at least at the Private level. If Am is classified at the Private level, then at least one of Al, A2, ..., An must be classified at least at the Private level.

In Section 4 we showed how the logical constraints might be handled during query processing. For example consider the constraint Ai ⇒ Aj. If Aj is classified at the Private level, and a Public

user requests for Ai values, the query processor will ensure that the Ai values are not released. That is, although Ai may be explicitly assigned the Public level, since the logical constraint is treated as a derivation rule, it does not cause any inconsistency. That is, during query processing, the privacy level of Ad will be computed to be Private.

For logical constraints which do not have any conditions attached, it appears that they could be handled during database design. That is during design time the logical constraints are examined, and the privacy levels of the attributes specified in the premise of a constraint could be computed. For example, if Aj is classified at the Private level then it must be ensured during design time that Ai is classified at least at the Private level also. The following algorithm will ensure that the privacy levels are computed correctly.

1. Do the following for each logical constraint (Note that we have assumed that the constraints are expressed as horn clauses. That is, there is only one atom in the head of a clause).
2. Check whether there are any simple constraints which classify the attribute appearing in the head of the logical constraint at any level. If not, ignore the constraint.
3. If so, find the highest privacy level L that is specified for this attribute.
4. Check whether any of the attributes appearing as premises of the logical constraint are classified at least at level L. If so, ignore the constraint.
5. If not, classify one of the attributes (say, the first one) at the level L.

The algorithm given above does not ensure that the attributes are not overclassified with respect to privacy. In order to avoid the overclassification problem, modification must be made to step *5*. That is, once an attribute is assigned a privacy level, it is possible for the level to be re-assigned based on other logical constraints that are handled. Our current research includes investigating techniques for successfully assigning privacy levels to the attributes and at the same time avoiding overclassification.

When logical, simple, and association-based constraints are combined, then the first step would be to handle the simple constraints. The next step would be to apply the algorithm given above for the logical constraints. Finally the algorithm given in Section 7.2 is applied for the association-based constraints.

7.5. Algorithm for database design

Algorithm D summarizes the operation of the database design tool.

Algorithm D: Database Design.
Input: Schema, Constraints.
Output: Privacy level of the schema.

Step 1: For each attribute in the schema, examine the simple constraints and compute the security levels of the attributes.
Step 2: For each attribute in the schema, consider the logical constraints as well as the level-based constraints and determine whether the attributes are assigned privacy levels either directly or by implication. If so, assign the correct privacy level to the attribute.

For example, if prescriptions imply medical conditions and medical conditions are private, the prescriptions are also private.

Step 3: For each attribute in the schema, examine the release constraints and determine whether the attribute has to be assigned a privacy level other than the lowest level and if so, assign the correct privacy level to the attribute. For example, if medical conditions are private after the medical records are released and if the medical records are released, then assign level private to the medical conditions.

Step 4: Give a status to the database administrator.

8. Extensions to the integrated architecture

In this section we will discuss extensions to the current integrated architecture. Section 8.1 discusses privacy control in distributed and federated environments. Section 8.2 discusses release control where privacy constraints are processed only after the data is released by the database manager.

8.1. Privacy control is distributed and federated environments

In our approach discussed so far, the privacy controller is centralized. We can also develop a distributed privacy controller as illustrated in Fig. 7. For example, organizations may want to share data and at the same time have autonomy. These organizations could be hospitals in different locations. Each DBMS of an organization is augmented with a distributed processor (DP) responsible for distributed operations. The distributed privacy controller (DPC) is part of the distributed processor.

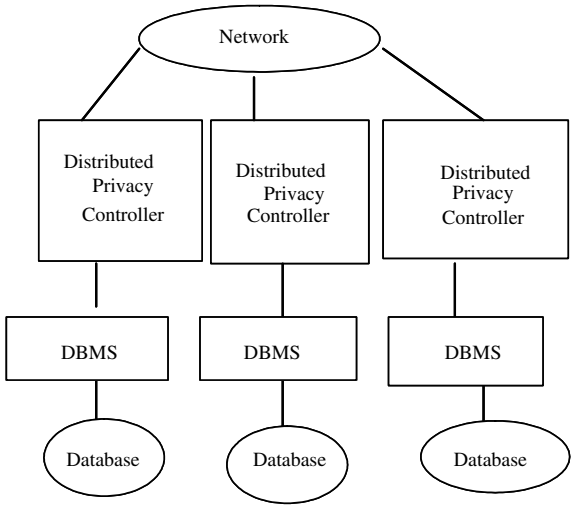

Fig. 7. Distributed privacy controller.

DPC consists of components for distributed query processing, distributed update processing and distributed database design. In [18] we provided a detailed overview of security constraint processing in a distributed environment. We are proposing a similar approach for privacy constraint processing. That is, some constraints are processed during the distributed query operation, some during the distributed update operation and some during the distributed database design operation. The challenge is for the DCPs at different sites to coordinate their activities so that privacy is preserved as much as possible across the sites.

In one of our earlier papers published (see [17]) we discussed security issues for federated database systems. A federated database system enables organizations to share data but at the same time maintain autonomy. We find that the federated database concept is very useful for federal organizations to form federations and share data.

Fig. 8 illustrates an architecture for federated data management and privacy. In this architecture, each agency/organization has its own component data management system. Each component is augmented by a privacy controller, which controls privacy violations at the local level. An export engine further determines the data that can be exported to the federation. The federated data management system manages the federated data. It is also augmented by a privacy controller to ensure privacy at the global level.

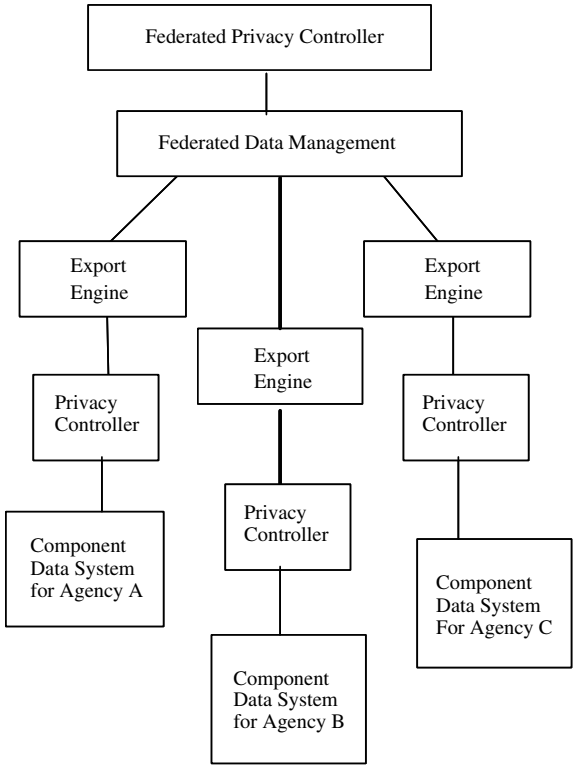

Fig. 8. Federated data management, data sharing and privacy.

We have provided some ideas on distributed and federated data management and privacy control. In a future paper we will elaborate on the ideas and discuss the design of distributed and federated architectures as well as the privacy controllers for such architectures.

8.2. Privacy control processing and release control

Until now we have discussed an integrated architecture mainly for a centralized environment where constraints are examined during query, database update and database design time. That is, in the case of the query operation much of the work is carried out when before the query is sent to the PE-DBMS. Once the query is executed, then we examine certain release constraints to see what has been released before.

Recently there has been some work on processing constraints only after the response is released by the DBMS but before the response is given to the user. These approaches can be found in [20,8]. The idea behind this approach is that it may not be feasible to modify the query especially if there are many constraints. Therefore, rather than doing the work before the query is executed, why not do all of the processing after the response is released by the DBMS?

This approach does have some disadvantages. That is, after the response is released, one has to examine all of the privacy constraints and determine what information to release to the use. Essentially many of the operations carried out by the DBMS will now have to be carried out by the Release Control Manager (RCM). That is, the Release Control Manager will have to carry out selections, projects, joins, etc to obtain the final result. However the advantage with this approach is that if there are many constraints the complex query modification process is avoided.

Note that the DBMS will produce the result at the user's privacy level. The RCM will then examine the result and the constraints and determine whether all of the data could be released. Suppose there is a constraint that stated that RECORDS in MEDICAL are private and the user's level is public. The release data will have all of the information in the relation MEDICAL. RCM will apply the constraint and only give out the medical record number and the insurance attribute values and not the actual record attribute values. Fig. 9 illustrates the RCM. We will elaborate on the various modules and the design in a future paper.

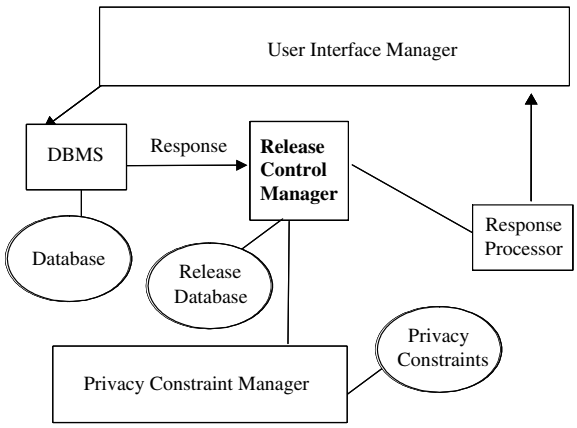

Fig. 9. Release control manager.

9. Summary and directions

In this paper we first defined various types of privacy constraints. Privacy constraints are rules, which assign privacy levels to the data. Then we described an integrated approach to constraint processing. That is, some constraints are handled during query processing, some during database updates, and some during database design. We then described the design of two prototypes, which process constraints during query and update operations. We also described the design of a database design tool. Finally we showed how the query processor, update processor, and the database design tool could be combined so that an integrated solution to constraint processing can be achieved. We also briefly discussed extensions to a distributed architecture as well as discussed an alternative approach to privacy constraint processing during release processing.

Future work includes examining the release control approach further. In addition we need to conduct research on privacy control algorithms for distributed and federated environments. We provided some initial direction in this paper. It is very important that we develop techniques for generating constraints. For example, we may need to examine the use of semantic models and conceptual structures to specify the application and reason about the application and detect privacy violations during design time. We also need to examine the enforcement of more complex constraints. In other words, there is lot of research to be done on privacy constraint processing.

Privacy constraint processing is just one solution to ensuring the privacy of individuals. We also need to explore other solutions such as privacy preserving data mining. In this approach the idea is to mine the data and extract information but at the same time limiting or preventing privacy violations (see for example, [4]). There is a lot of recent work on privacy preserving data mining and this area has received a lot of attention. We need to combine this research with privacy constraint processing.

10. Disclaimer

The views and conclusions expressed in this paper are those of the author and do not reflect the policies of the National Science Foundation or of the MITRE Corporation.

Acknowledgment

I thank the National Science Foundation and the MITRE Corporation for their support of my research on security issues for sensor data management.

References

[1] Air Force Summer Study Report on Multilevel Secure Database Systems, Washington DC, 1983.
[2] R. Agrawal et al., Hypocritical Databases, VLDB, 2002.

[3] D. Bell, L. LaPadula, Secure Computer Systems and Multics Interpretation, Technical Report, The MITRE Corporation, 1975.
[4] C. Clifton, M. Kantarcioglu, J. Vaidya, Defining Privacy for Data Mining, Purdue University, 2002 (see also Next Generation Data Mining Workshop, Baltimore, MD, November).
[5] P. Dwyer, G. Gelatis, B. Thuraisingham, Multilevel security in relational database systems, Computers and Security, June 1987.
[6] H. Gallaire, J. Minker, Logic and Databases, Plenum press, NY.
[7] T. Hinke, Inference and aggregation detection in database management systems, Proceedings of the Security and Privacy Conference, Oakland, CA, April 1988.
[8] S. Jajodia, Release control for XML documents, Proceedings IFIP Conference in Integrity and Control, Lausanne, Switzerland, November 2003.
[9] T. Keefe, B. Thuraisingham, W. Tsai, Secure query processing strategies, IEEE Computer, March 1989.
[10] J. Lloyd, Logic programming, Springer Verlag, Heidelberg, 1987.
[11] J. Minker, Foundations of Deductive Databases and Logic Programming, Morgan Kaufman, 1988.
[12] M. Morgenstern, Security and inference in multilevel database and knowledge base systems, Proceedings of the ACM SIGMOD Conference, San Francisco, CA, June 1987.
[13] M. Stonebraker, E. Wong, Access control in relational database systems through query modification, Proceedings ACM National Conference, 1974.
[14] B. Thuraisingham, Multilevel security for relational database systems augmented by an inference engine, Computers and Security, December 1987.
[15] B. Thuraisingham, Towards the design of a secure data/knowledge base management system, Data and Knowledge Engineering, March 1990.
[16] B. Thuraisingham, W. Ford, M. Collins, Design and implementation of a database inference controller, Data and Knowledge Engineering Journal, December 1993.
[17] B. Thuraisingham, Multilevel security for federated databases, Computers and Security, December 1994.
[18] B. Thuraisingham, W. Ford, Security constraint processing in a multilevel distributed database management system, IEEE Transactions on Knowledge and Data Engineering, April 1995.
[19] B. Thuraisingham, Web Data Mining: Technologies and their Applications in Business Intelligence and Counter-terrorism, CRC Press, FL, 2003.
[20] G. Wiederhold, Release control in database systems, Proceedings IFIP Database Security Conference, Amsterdam, August 2000.

Further reading

[i] B. Thuraisingham, Data Mining: Technologies, Techniques, Tools and Trends, CRC Press, FL, 1998.

Dr. Bhavani Thuraisingham, has recently joined The University of Texas at Dallas as a Professor of Computer Science and Director of the Cyber Security Research Center in the Erik Jonsson School of Engineering. She was the Program Director for Cyber Trust and Data and Applications Security of the National Science Foundation and has been on IPA to NSF of the MITRE Corporation since October 2001. She was part of a team at NSF setting directions for cyber security and data mining for counter-terrorism. She has been with MITRE since January 1989, where she was the Department Head in Data and Information Management of the Information Technology Division and later the Chief Scientist in Data Management in MITRE's Information Technology Directorate. She has conducted research in secure databases for over 19 years and is the recipient of IEEE Computer Society's 1997 Technical Achievement Award and recently IEEE's 2003 Fellow Award for her work in database security. She is also a 2003 Fellow of the American Association for the Advancement of Science. Dr. Thuraisingham has published over 200 refereed conference papers and over 60 journal articles in secure data management and information technology. She serves (or has served) on editorial boards of

journals including IEEE Transactions on Knowledge and Data Engineering, IEEE Transactions on Dependable and Secure Computing, ACM Transactions of Information and Systems Security, Journal of Computer Security and Computer Standards and Interface Journal. She is the inventor of three patents for MITRE on Database Inference Control and has written seven books on data management and data mining for technical managers and has recently completed a textbook on database and application security based on her work the past 19 years. Her research interests are in secure semantic web, sensor information security, and data mining for counter-terrorism.