Privacy-Preserving Database Systems

Elisa Bertino, Ji-Won Byun, and Ninghui Li

Department of Computer Science and Cerias,
Purdue University,
656 Oval Drive, West Lafayette, IN 47907
{bertino, byunj, ninghui}@cs.purdue.edu

Abstract. Privacy is today an important concern for both users and enterprises. Therefore, intense research is today being carried out on various aspects of privacy-preserving data management systems. In this paper, we focus on database management systems (DBMS) able to enforce privacy promises encoded in privacy languages such as P3P. In particular, in the paper, we first present an overview of the P3P language and outlines some of its critical aspects. We then outline the main requirements for a privacy-preserving DBMS and we discuss solutions related to the management of privacy-related meta-data, focusing on special category of meta-data information, that is, purpose information. Purpose information represents an important component of privacy statements and thus their effective management is crucial. We then discuss current solutions to to fine-grained access control in the context of relational database systems and identify relevant issues.

1 Introduction

Data represent today an important asset. We see an increasing number of organizations that collect data, very often concerning individuals, and use them for various purposes, ranging from scientific research, as in the case of medical data, to demographic trend analysis and marketing purposes. Organizations may also give access to the data they own or even release such data to third parties. The number of increased data sets that are thus available poses serious threats against the privacy of individuals and organizations. Because privacy is today an important concern, several research efforts have been devoted to address issues related to the development of privacy-preserving data management techniques.

A first important class of techniques deals with privacy-preservation when data are to be released to third parties. In this case, data once are released are not any longer under the control of the organizations owning them. Therefore, the organizations owners of the data are not able to control the way data are used. The most common approach to address the privacy of released data is to modify the data by removing all information that can directly link data items with individuals; such a process is referred to as data anonymization. It is important to note that simply removing identity information, like names or social-security-numbers, from the released data may not be enough to anonymize the data. There are many examples showing that even when such information is removed from the released data, the remaining data combined with other information sources may still link the information to the individuals it refers to. To overcome this problem, approaches based on generalization techniques have been proposed, the most well known of which is based on the notion of k-anonymity [27,28].

A second class of techniques deals specifically with privacy-preservation in the context of data mining. Data mining techniques are today very effective. Thus even though a database is sanitized by removing private information, the use of data mining techniques may allow one to recover the removed information. Several approaches have been proposed, some of which are specialized for specific data mining techniques, for example tools for association rule mining or classification systems, whereas other are independent from the specific data mining technique. In general all approaches are based on modifying or perturbing the data in some way; for example, techniques specialized for privacy preserving mining of association rules modify the data so to reduce the confidence of sensitive association rules. A problem common to most of those techniques is represented by the quality of the resulting database; if data undergo too many modifications, they may not be any longer useful. To address this problem, techniques have been developed to estimate the errors introduced by the modifications; such estimate can be used to drive the data modification process. A different technique in this context is based on data sampling [7]. The idea is to release a subset of the data, chosen in such a way that any inference made from the data has a low degree of confidence. Finally, still in the area of data mining, techniques have been developed, mainly based on commutative encryption techniques, the goal of which is to support distributed data mining processes on encrypted data [8]. In particular, the addressed problem deals with situations in which the data to be mined is contained at multiple sites, but the sites are unable to release the data. The solutions involve algorithms that share some information to calculate correct results, where the shared information can be shown not to disclose private data.

Finally, some efforts have been reported dealing with database management systems (DBMS) specifically tailored to support privacy policies, like the policies that can be expressed by using the well known P3P standard [29]. In particular, Agrawal et al. [1] have recently introduced the concept of Hippocratic databases, incorporating privacy protection in relational database systems. In their paper, Agrawal et al. introduce the fundamental principles underlying Hippocratic databases and then propose a reference architecture. An important feature of such architecture is that it uses some privacy metadata, consisting of privacy policies and privacy authorizations stored in privacy-policies tables and privacy-authorizations table respectively. There is strong need for development of privacy-preserving DBMS driven by the demand organizations have of complying with various privacy laws and requirements and of increasing user trusts [19]. The development of database technology entails however addressing many challenging issues, ranging from modeling to architectures, and may lead to the next-generation of DBMS.

In this paper we focus on the development of privacy-preserving DBMS because it poses several challenges with respect to both theory and architectures.

It is important to notice, however, that privacy-preserving DBMS may be combined with tools for data anonymization and privacy-preserving data mining in order to provide comprehensive platforms for supporting flexible and articulated privacy-preserving information management.

The remainder of this paper is organized as follows. In Section 2, we provide an overview of P3P; this standard, even though has several limitations [19], is an important reference in the current privacy practices. Thus, it represens one of the starting points for development of privacy-preserving DBMS. In Section 3 we then discuss relevant requirements towards the development of privacy-preserving DBMS; in particular, we elaborate on some of the requirements arising from the support of P3P policies. We then survey some existing solutions and elaborate on some of those requirements in Section 4. We review some techniques of fine-grained access control and discuss some key challenges in Section 5, and we conclude the paper in Section 6.

2 Platform for Privacy Preferences (P3P)

The W3C's Platform for Privacy Preferences Project (P3P) [29] is one major effort to improve today's online privacy practices. P3P enables websites to encode their data-collection and data-use practices in a machine-readable XML format, known as P3P policies [12]. The W3C has also designed APPEL (A P3P Preference Exchange Language) [16], which allows users to specify their privacy preferences. Ideally, through the use of P3P and APPEL, a user's agent should be able to check a website's privacy policy against the user's privacy preferences, and automatically determine when the user's private information can be disclosed. In short, P3P and APPEL are designed to enable users to play an active role in controlling their private information. In this section, we provide a brief overview of P3P and APPEL and discuss some related issues.

2.1 Overview of P3P

Each P3P policy is specified by one `POLICY` element that includes the following major elements.

– One `ENTITY` element: identifies the legal entity making the representation of privacy practices contained in the policy.
– One `ACCESS` element: indicates whether the site allows users to access the various kind of information collected about them.
– One `DISPUTES-GROUP` element: contains one or more `DISPUTES` elements that describe dispute resolution procedures to be followed when disputes arise about a service's privacy practices.
– Zero or more `EXTENSION` elements: contain a website's self-defined extensions to the P3P specification.
– And one or more `STATEMENT` elements: describe data collection, use and storage. A `STATEMENT` element specifies the data (e.g. user's name) and the data categories (e.g. user's demographic data) being collected by the site, as well as the purposes, recipients and retention of that data.

```
<STATEMENT>
 <PURPOSE><admin required="opt-in"/></PURPOSE>
 <RECIPIENT><public/></RECIPIENT>
 <RETENTION><indefinitely/></RETENTION>
 <DATA-GROUP>
  <DATA ref="#user.home-info.postal"></DATA>
 </DATA-GROUP>
</STATEMENT>
```

```
stmt(
 purpose: {admin(opt-in)}
 recipient: {public}
 retention: {indefinitely}
 data: {#user.home-info.postal}
)
```

Fig. 1. An Example P3P Statement. The XML representation appears on the left side and a more succinct representation on the right side.

There are two kinds of P3P statements. The first kind contains the NON-IDENTIFIABLE element, which is used to indicate that either no information will be collected or information will be anonymized during collection. The second kind does not contain the NON-IDENTIFIABLE element; this is the commonly used one. For now, we will focus on the latter. A brief discussion of statements with NON-IDENTIFIABLE element is given later in this section.

Figure 1 provides an example of a P3P statement. Each such statement contains the following:

– One PURPOSE element, which describes for which purpose(s) the information will be used. It contains one or more pre-defined values such as current, admin, individual-analysis and historical. A purpose value can have an optional attribute 'required', which takes one of the following values: opt-in, opt-out, and always. The value 'opt-in' means that data may be used for this purpose only when the user affirmatively requests this use. The value 'opt-out' means that data may be used for this purpose unless the user requests that it not be used in this way. The value 'always' means that users cannot opt-in or opt-out of this use of their data. Therefore; in terms of strength of data usage, 'always' > 'opt-out' > 'opt-in'. In Figure 1, PURPOSE is admin and the attribute 'required' takes the value opt-in.
– One RECIPIENT element, which describes with whom the collected information will be shared. It contains one or more pre-defined values such as ours, delivery and public. A recipient value can have an optional attribute 'required', which is similar to that of a PURPOSE element. In Figure 1, RECIPIENT is public.
– One RETENTION element, which describes for how long the collected information will be kept. It contains exactly one of the following pre-defined values: no-retention, stated-purpose, legal-requirement, business-practices and indefinitely. In Figure 1, the RETENTION value is indefinite.
– One or more DATA-GROUP elements, which specify what information will be collected and used. Each DATA-GROUP element contains one or more DATA elements. Each DATA element has two attributes. The mandatory attribute 'ref' identifies the data being collected. For example, '#user.home-info.telecom. telephone' identifies a user's home telephone number. The 'optional' attribute indicates whether or not the data collection is optional. A DATA

element may also contain a `CATEGORIES` element, which describes the kind of information this data item is, e.g., financial, demographic and health. In Figure 1, `DATA` is postal info.
- Zero or one `CONSEQUENCE` element, which contains human-readable contents that can be shown to users to explain the data usage practice's ramifications and why the usage is useful.

2.2 Issues in P3P

Since proposed, P3P has received broad attention from both industry and the research community, and has been gradually adopted by companies. On the other hand, the full deployment of P3P in enterprise information systems has raised many challenging questions. For example, P3P represents an enterprise's promise to users about its privacy practice. How can we ensure that an organization and its customers have a common understanding of these promises? P3P promises must be fulfilled in the services provided by enterprises. How can a company guarantee that its P3P policy is correctly enforced in those applications? We discuss some of the issues regarding the former question (privacy policy specification) in this section. The issues regarding the latter question (privacy policy enforcement) are discussed in the subsequent section.

The Lack of Formal Semantics in P3P. One major problem that hinders P3P adoption is that a P3P policy may be interpreted and represented differently by different user agents. Companies are thus reluctant to provide P3P policies on their websites, fearing that the policies may be misrepresented [11,25]. Quoting from CitiGroup's position paper [25], "The same P3P policy could be represented to users in ways that may be counter to each other as well as to the intent of the site." "... This results in legal and media risk for companies implementing P3P that needs to be addressed and resolved if P3P is to fulfill a very important need."

For instance, consider the statement in Figure 1. In the statement, the three components (purpose, recipient and retention) all refer to the same data item '#user.home-info.postal'; however, for the statement to have a precise meaning, one must also determine how these components interact. We consider two interpretations. In the first interpretation, all three components are related, i.e., the purpose, the recipient and the retention are about *one* data usage. In Figure 1, the postal information will be used for the admin purpose (technical support of the website and its computer system); the information will be shared with the public and will be stored indefinitely. For this statement, this interpretation seems counterintuitive, because there is no need to share the data with the public for the admin purpose. Furthermore, it is not clear whether this data usage is required or optional, since the 'required' attribute has the 'opt-in' value for purpose but the default 'always' value for recipient. The explanation for this statement, provided by one of the P3P architects [10], is that the data item '#user.home-info.postal' will always be collected and shared with the public. Additionally, if the user chooses to opt-in, their postal information will be used for the admin purpose. In other words, whether the individual's postal information will be shared with the public does not depend upon whether or not the information is used for the admin purpose.

This leads us to the second interpretation, in which purpose, recipient and retention are considered orthogonal. In this interpretation, a P3P statement specifies three relations: the purposes for which a data item will be used, the recipients with whom a data item will be shared, and how long the data item will be stored. Even though these relations are specified in the same statement, they are not necessarily about a single data usage. Given this *data-centric* interpretation, the following three P3P policies will have the same meaning in the sense that all relations contain a data component:

Example 1. Three P3P policies that have the same meaning.

Policy 1:
```
stmt( data: {#user.home-info.telecom,
             #user.bdate(optional)},
      purpose: {individual-analysis,
                telemarketing(opt-in)},
      recipient: {ours},
      retention: {stated-purpose} )
```
Policy 2:
```
stmt( data: {#user.home-info.telecom,
             #user.bdate(optional)},
      purpose: {individual-analysis},
      recipient: {ours},
      retention: {stated-purpose})
stmt( data: {#user.home-info.telecom,
             #user.bdate(optional)},
      purpose: {telemarketing(opt-in)},
      recipient: {ours},
      retention: {stated-purpose} )
```
Policy 3:
```
stmt( data: {#user.home-info.telecom},
      purpose: {individual-analysis,
                telemarketing(opt-in)},
      recipient: {ours},
      retention: {stated-purpose} )
stmt( data: {#user.bdate(optional)},
      purpose: {individual-analysis,
                telemarketing(opt-in)},
      recipient: {ours},
      retention: {stated-purpose} )
```

Part of this problem is caused by overlooking the need for a semantics in the initial design of P3P, leaving too much freedom for P3P policies to be misinterpreted and misrepresented by user agents. The fact that the same meaning may be encoded in several different ways makes it very difficult to correctly express privacy preferences in a syntax-based preference language such as APPEL. One representation can be accepted by a preference, but another representation could be rejected by the same preference.

Potential Semantic Inconsistencies in P3P Policies. In general, any combinations of the values for purpose, recipient and retention are allowed in P3P. However, in a practical setting, semantic dependencies arise naturally between these values, making some of the combinations invalid. A P3P policy using invalid combinations is thus semantically inconsistent. This problem has been recognized [9,24], and P3P's designers are beginning to address some of these conflicts [9]. Nonetheless, many places where potential conflicts may occur have not been previously identified. We now identify some additional classes of potential semantic inconsistencies in P3P.

– *A P3P policy may be inconsistent because multiple retention values apply to one data item.*
  P3P allows one data item to appear in multiple statements, which introduces a semantic problem. Recall that in each P3P statement, only one retention value can be specified, even though multiple purposes and recipients can be used. The rationale behind this is that retention values are mutually exclusive, i.e., two retention values conflict with each other. For instance, *no-retention* means that "Information is not retained for more than a brief period of time necessary to make use of it during the course of a single online interaction" [12]. And *indefinitely* means that "Information is retained for an indeterminate period of time" [12]. One data item cannot have both retention values. However, allowing one data item to appear in multiple statements makes it possible for multiple retention values to apply to one data item.
– *A statement may have conflicting purposes and retention values.*
  Consider a statement in a P3P policy that collects users' postal information for the purpose *historical* with retention *no-retention*. Clearly, if the postal information is going to be "... archived or stored for the purpose of preserving social history ...", as described by the *historical* purpose, it will conflict with *no-retention*, which requires that the collected information "... MUST NOT be logged, archived or otherwise stored" [12].
– *A statement may have conflicting purposes and recipients.*
  Consider a statement that includes all the purpose values (e.g., history, admin, telemarketing, individual-analysis, etc.) but only the recipient value *delivery* (delivery services). This does not make sense as one would expect that at least *ours* should be included in the recipients.
– *A statement may have conflicting purposes and data items.*
  Certain purposes imply the collection and usage of some data items. This has been recognized by the P3P designers and reflected in the guidelines for designing P3P user agents [9]. For example, suppose a statement contains purpose contact but does not collect any information from the categories physical and online. Then the statement is inconsistent because, in order to contact a user, "the initiator of the contact would possess a data element identifying the individual . . . . This would presuppose elements contained by one of the above categories" [9].

All semantic inconsistency instances must be identified and specified in the P3P specification. Completion of this work requires a detailed analysis of the vocabulary, ideally by the individuals who design and use these vocabularies.

Dealing with P3P Statements Having the `NON-IDENTIFIABLE` Element. A `STATEMENT` element in a P3P policy may optionally contain the `NON-IDENTIFIABLE` element, which "signifies that either no data is collected (including Web logs), or that the organization collecting the data will anonymize the data referenced in the enclosing `STATEMENT`" [12]. We call such statements non-identifiable statements.

From the above description, we see that the `NON-IDENTIFIABLE` element is used for two unrelated purposes in P3P. We argue that using the `NON-IDENTIFIABLE` element to signify that no data is collected is inappropriate. Intuitively, if a statement with the `NON-IDENTIFIABLE` element contains the `DATA-GROUP` element, it means that the data collected in this statement is anonymized. If such a statement does not have the `DATA-GROUP` element, it means that no data is collected. However, this statement is meaningless when the policy contains other statements that collect and use data. In general, the fact that a policy does not collect any data should not be specified at the level of a `STATEMENT` element; instead, it should be specified at the level of a `POLICY` statement. For instance, it seems more appropriate to use a separate sub-element (or an attribute) for the `POLICY` element to denote that a policy collects no data.

Another issue that arises from having the `NON-IDENTIFIABLE` element is that a data item may appear both in normal statements and in non-identifiable statements. In this situation, it is not clear whether the data item is anonymized upon collection. According to Cranor [10], it may be possible that: "a company keeps two different unlinkable databases and the data is anonymized in one but not the other."

The most straightforward way seems to annotate an anonymized data item so that it is different from a normal data item, e.g., one may use '#user.home-info' to denote a normal data item and '#@.user.home-info' to denote an anonymized version of the same data.

2.3 An Overview of APPEL

Privacy preferences are expressed as a *ruleset* in APPEL. A ruleset is an ordered set of rules. An APPEL evaluator evaluates a ruleset against a P3P policy.[1] A rule includes the following two parts:

– A behavior, which specifies the action to be taken if the rule fires. It can be *request*, implying that a P3P policy conforms to preferences specified in the rule body and should be accepted. We call this an *accept* rule. It can be

[1] The APPEL specification allows arbitrary XML elements not related to P3P to be included together with the P3P policy for evaluation. Since the users may not even know about them, it is not clear how they write preferences dealing with them. In this paper, we assume that only a P3P policy is evaluated against a ruleset.

block, implying that a P3P policy violates the user's privacy preferences and should be rejected. We call this a *reject* rule. It can also be *limit*, which can be interpreted as accept with warning.
- A number of expressions, which follow the XML structure in P3P policies. An expression may contain subexpressions. An expression is evaluated to TRUE or FALSE by matching (recursively) against a target XML element. A rule *fires* if the expressions in the rule evaluate to TRUE.

Every APPEL expression has a *connective* attribute that defines the logical operators between its subexpressions and subelements of the target XML element to be matched against. A connective can be: *or*, *and*, *non-or*, *non-and*, *or-exact* and *and-exact*. The default connective is *and*, which means that all subexpressions must match against some subelements in the target XML element, but the target element may contain subelements that do not match any subexpression. The connective *and-exact* further requires that any subelement in the target match one subexpression. The *or* connective means that at least one subexpression matches a subelement of the target. The *or-exact* connective further requires that all subelements in the target matches some subexpressions.

When evaluating a ruleset against a P3P policy, each rule in a ruleset is evaluated in the order in which it appears. Once a rule evaluates to true, the corresponding behavior is returned.

2.4 Issues in EPPAL

Many authors have noted that APPEL is complex and problematic [2,13,14,30]. In this section, we analyze APPEL's pitfalls and the rationales for some of the design decisions embedded in APPEL. The main objective for our analysis is to ensure we design a preference language that avoids the APPEL's pitfalls but preserves the desirable functionalities in APPEL. The following subsections examine the pitfalls and limitations of APPEL.

Semantic Inconsistencies of APPEL. Because of APPEL's syntax-based design, two P3P policies that have the same semantic meanings but are expressed in syntactically different ways may be treated differently by one APPEL ruleset. This deficiency in APPEL has been identified before [15]. We now show an example APPEL rule.

```
<appel:RULE behavior="block">
 <p3p:POLICY>
  <p3p:STATEMENT>
   <p3p:DATA-GROUP>
    <p3p:DATA ref=
     "#user.home-info.telecom"/>
    <p3p:DATA ref="#user.bdate"/>
   </p3p:DATA-GROUP>
  </p3p:STATEMENT>
 </p3p:POLICY>
</appel:RULE>
```

This is a reject rule, since the behavior (on line 1) is “block”. The body of this rule has one expression (lines 2-10) for matching a P3P policy. This expression contains one subexpression (lines 3-9), which in turns contain one subexpression (lines 4-8). The outmost expression (lines 2-10) uses the default **and** directive; therefore, it matches a P3P policy only if the policy contains at least one statement that matches the enclosed expression (lines 3-9). Overall, this APPEL rule says that a P3P policy will be rejected if it contains a `STATEMENT` element that mentions both the user’s birthday and the user’s home telephone number.

This rule rejects Policies 1 and 2 in Example 2, but not Policy 3. In Policy 3, the two data items are mentioned in different statements and no statement mentions both data items. This is clearly undesirable, as the three statements have the same semantics. This problem is a direct consequence of that fact that APPEL is designed to query the representation of a P3P policy, rather the semantics of the policy.

The same problem exists in XPref [2], since it is also syntax-based.

The Subtlety of APPEL’s Connectives. The meaning of an APPEL rule depends very much on the connective used in the expressions. However, the connectives are difficult to understand and use. The APPEL designers made mistakes using them in the first example in the APPEL specification [16]. Consider the following example taken from [16].

Example 2. The user does not mind revealing click-stream and user agent information to sites that collect no other information. However, she insists that the service provides some form of assurance.

The APPEL rule used in [16] for the above example is as follows:

```
01 <appel:RULE behavior="request"
02   description="clickstream okay">
03 <p3p:POLICY>
04  <p3p:STATEMENT>
05   <p3p:DATA-GROUP
06     appel:connective="or-exact">
07    <p3p:DATA
08     ref="#dynamic.http.useragent"/>
09    <p3p:DATA
10     ref="#dynamic.clickstream.server"/>
11   </p3p:DATA-GROUP>
12  </p3p:STATEMENT>
13  <p3p:DISPUTES-GROUP>
14   <p3p:DISPUTES service="*"/>
15  </p3p:DISPUTES-GROUP>
16 </p3p:POLICY>
17 </appel:RULE>
```

The above APPEL rule is an accept rule; its body has one outmost expression (lines 3-16) to match a P3P policy. The expression contains two subexpressions, matching different elements in a policy. The expression denoted by the p3p:POLICY element (lines 3–16) does not have the 'connective' attribute; therefore, the default **and** connective is used, which means that as long as the two included expressions, i.e., p3p:STATEMENT (lines 4-12) and p3p:DISPUTES-GROUP (lines 13-15), match some parts in the P3P policy, the rule accepts the policy. The expression denoted by p3p:DATA-GROUP uses the **or-exact** connective, it matches a DATA-GROUP element if the DATA elements contained in the element is a non-empty subset of {#dynamic.http.useragent, #dynamic.clickstream.server}.

Overall, this rule means that a P3P policy will be accepted if it contains a STATEMENT element that mentions only the two specified data items and a DISPUTES-GROUP element.

Observe that this rule does not express the preference, as it does not take into consideration the fact that a P3P policy can have multiple statements. Subsequently, a policy that only mentions "#dynamic.http.useragent" in the first statement can be accepted by the rule, even if the next statement collects and uses other user data.

One may try to fix this problem by using the **and-exact** connective on line 2, which means that each element contained in the P3P policy must match one of the expressions within the APPEL rule. For such a rule to work as intended, the p3p:POLICY expression (lines 3–16) must contain two additional sub-expressions: p3p:ENTITY and p3p:ACCESS. Otherwise, no P3P policy will be accepted because of the existence of ENTITY and ACCESS elements. However, this fix still does not work. A P3P policy may optionally contain an EXTENSION element. Even when such a policy collects only '#dynamic.http.useragent' and '#dynamic.clickstream.server', it will not be accepted by the above rule, due to the semantics of **and-exact**. On the other hand, including a sub-expression p3p:EXTENTION cannot fix the problem, as a policy without extensions will not be accepted in this case.

Another approach to fix the problem is to use the **or-exact** connective on line 3 and to include subexpressions for p3p:ENTITY, p3p:ACCESS and p3p:EXTENSION. Recall that the **or-exact** connective means that all elements in the policy must match some subexpressions, but not every subexpression is required to match some element in the policy. However, this means that the p3p:DISPUTES-GROUP element also becomes optional. A P3P policy that does not have the DISPUTES-GROUP element will also be accepted. This is not the preference described in Example 2.

In fact, as far as we can see, there is no way to correctly specify the preference in Example 2 in APPEL. One source of difficulty is that one has to intermingle statements and other aspects (e.g., dispute procedures) of a P3P policy in a preference rule. This is caused by APPEL's syntactic nature and the fact that statements and dispute procedures are all immediate children of a POLICY element. If conditions about data usages and other aspects of P3P policies may be specified separately, it is possible to specify the conditions on data usage in Example 2 using the **or-exact** connective.

3 Requirements Towards the Development of Privacy-Preserving DBMS

Languages for specification of privacy promises, such as P3P, represent only one of the components in a comprehensive solution to privacy [3]. It is crucial that once data are collected, privacy promises be enforced by the information systems managing them. Because in today information systems, data are in most cases managed by DBMS, the development of DBMS properly equipped for the enforcement of privacy promises and of other privacy policies is crucial. Here we discuss a set of requirements towards the development of such DBMS. Some of those requirements, as the support for purpose meta-data and privacy obligations, derive directly from P3P. Other requirements are not directly related to P3P; however, they are crucial for the development of DBMS able to support a wide range of privacy policies, going beyond the ones strictly related to P3P. In the discussion, we use the terms subject to denote the active entities, trying to gain accesses to the data, and subjects to denote the passive entities, that are to be protected.

Support for Rich Privacy Related Meta-data. An important characteristic of P3P is that very often privacy promises, that is, statements specifying the use of the data by the party collecting them, include the specification of the intended use of the data by the collecting party as well as other information. Examples of this additional information are how long the data will be kept and possible actions that are to be executed whenever a subject accesses the data. Supporting this additional information calls for the need of privacy-specific meta-data that should be associated with the data, stored in the database together with the data, and send with the data whenever the data flow to other parties in the system. Metadata should be associated with the data according to a range of possible granularities. For example in a relational database, one should be able to associate specific metadata with an entire table, with a single tuple, or even with a column within a single tuple. Such flexibility should not however affect the performance; thus we need to develop highly efficient techniques for managing these metadata in particular when dealing with query executions. Query executions may need to take into account the contents of such metadata in order to filter out from the data to be returned, the data that cannot be accessed because of privacy constraints.

Support for Expressive Attribute-Based Descriptions of Subjects. We see an increasing trend towards the development of access control models that relies on information concerning subjects. Examples of such models are represented by trust negotiation systems [4,18], that use credentials certifying relevant properties of subjects. Such access control models are crucial in the context of privacy because they provide a high-level mechanism able to support a very detailed specification of the conditions that subjects must verify in order to access data. As such, fine-grained privacy-preserving access control policies can be supported. They also make it easy to formulate and maintain privacy policies and verify their correctness. Moreover, such high-level models can provide better support for interoperability because they can, for example, easily integrate with SAML assertions. However, current database technology is very poor in the representation of subjects. At the best current DBMS provide support for roles in the context of the well-known role-based access control (RBAC) model [3]. However, apart from this, DBMS do not provide the possibility of specifying application-dependent user profiles for use in access control and privacy enforcement. It can be argued that such profiles should perhaps be built on top of the DBMS or even be supported externally. However, in such a case, it is not clear how efficient access control and privacy enforcement could be supported. It also important to notice that RBAC does not support subject attributes. Extensions of RBAC models supporting such a feature should be devised.

Support for Obligations. Obligations specify privacy-related actions that are to be executed upon data accesses for certain purposes. There is a large variety of actions that can be undertaken, including modifications to the data, deletion of the data, notifications of data access to the individual to whom the data are related or to other individuals, insertion of records into privacy logs. These obligations should be possibly executed, or at least initiated by, the DBMS because their execution is tightly coupled with data accesses. An important issue here is the development of expressive languages supporting the specification obligations, and analysis tools to verify the correctness and consistency of obligations. A viable technology to support obligations is represented by trigger mechanisms, currently available in all commercial DBMS. The main question is however whether current trigger languages are adequate to support the specification of obligations.

Fine-Grained Access Control to Data. The availability of a fine-grained access control mechanism is an important requirement of a comprehensive solution to privacy. Conventional view mechanisms, the only available mechanism able to support in some ways a very fine granularity in access control, have several shortcomings. A naive solution to enforce fine-grained authorizations would require specifying a view for each tuple or subset of a tuple that are to be protected. Moreover, because access control policies are often different for different users, the number of views would further increase. Furthermore, applications programs would have to code different interfaces for each user, or group of users, because queries and other data management commands would need to use for each user, or group of users, the correct view. Modifications to access control policies would also require creation of new views with consequent modifications to application programs. Alternative approaches that address some of those issues have been proposed that are based on the idea that queries are written against base tables and then automatically re-written by the system against the view available to the user. These approaches do not require to code different interfaces for different users, and thus address on of the main problems in the use of conventional view mechanisms. However, they introduce other problems, such as inconsistencies between what the user expects to see and what the systems returns; in some cases, they return incorrect results to queries rather than rejecting them as unauthorized. Different solutions thus need to be investigated. These solutions must not only address the specification of fine-grained access control policies but also their efficient implementation in current DBMS.

Privacy-Preserving Information Flow. In many organizations, data flow across different domains. It is thus important that privacy policies related to data "stick" with the data when these data move within an organization or across organizations. This is crucial to assure that if data have been collected under a given privacy promise from an individual, this promise is enforced also when data are passed to parties different from the party that have initially collected them. Information flow has been extensively investigated in the past in the area of multi-level secure databases. An important issue is to revisit such theory and possibly extend it for application in the context of privacy.

Protection from Insider Threats. An important problem that so far has not received much attention is related to the misuse of privileges by legitimate subjects. Most research in the past has been devoted to protection from intrusions by subjects external to the systems, against which technologies like firewalls can provide a certain degree of protection. However, such approaches are not effective against users that are inside the firewalls. A possible technique that can be employed to start addressing such problem is based on the use of subject access profiling techniques. Once the profile of the legitimate accesses has been defined, such profile can be used to detect behavior that is different. Developing such an approach requires however investigating several issues, such as specific machine learning techniques to use, efficiency and scalability. Also, the collection of user profiles may in turn introduce more privacy problems, because users of the system may be sensible to the fact that all their actions are being monitored.

In the rest of this paper, we elaborate on some of those requirements and discuss possible solutions.

4 Purpose Management and Access Control

In this section, we present three privacy-centric access control models from recent literatures. Prior to proceed, we note that privacy protection cannot be easily achieved by traditional access control models. The key difficulty comes from the fact that privacy-oriented access control models are mainly concerned with which data object is used for which purpose(s) (i.e., the intent(s) of data usage), rather than which user is performing which action on which data object as in traditional access control models. Another difficulty of privacy protection is that the comfort level of privacy varies from individual to individual, which requires access control be fine-grained. Thus, the main challenge of privacy protecting access control is to provide access control based on the notion of purpose, incorporating data subjects' preferences if necessary, at the most fine-grained level. In the remainder of this section we discuss two approaches to the management of purpose information and their use in access control.

4.1 Purpose Based Access Control

Our work in [5,6] presents a comprehensive approach to purpose management, which is the fundamental building block on which purpose-based access control can be developed. Our approach is based on intended purposes, which specify the intended usage of data, and access purposes, which specify the purposes for which a given data element is accessed. We also introduce the notion of purpose compliance, which is the basis for verifying that the purpose of a data access with the intended purposes of the data.

Another important issue addressed in this work is the data labeling scheme; that is, how data are associated with intended purposes. We address this issue in the context of relational data model. The main issue here is the granularity of data labeling. We propose four different labeling schemes, each providing a different granularity. We also exploit query modification techniques to support data filtering based on purpose information.

Evidently, how the system determines the purpose of an access request is also crucial as the access decision is made directly based on the access purpose. To address this issue, we present a possible method for determining access purposes. In our approach, users are required to state their access purposes along with the data access requests, and the system validates the stated access purposes by ensuring that the users are indeed allowed to access data for the particular purposes.

Definition of Purposes. In privacy protecting access control models, the notion of purpose plays a central role as the purpose is the basic concept on which access decisions are made. In order to simplify the management, purposes are organized according to a hierarchical structure based on the principles of generalization and specialization, which is appropriate in common business environments. Figure 2 gives an example of purpose tree, where each node is represented with the conceptual name of a purpose.

Intuitively, an access to a specific data item is allowed if the purposes allowed by privacy policies for the data include or imply the purpose for accessing the data. We refer to purposes associated with data and thus regulating data accesses as *Intended Purposes*, and to purposes for accessing data as *Access Purposes.*

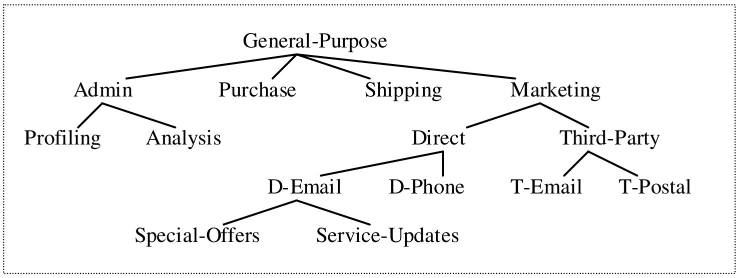

Fig. 2. Purpose Tree (From [5])

Intended purposes can be viewed as brief summaries of privacy policies for data, stating for which purposes data can be accessed. When an access to data is requested, the access purpose is checked against the intended purposes for the data.

In our model intended purposes support both positive and negative privacy policies. An intended purpose consists of two components: *Allowed Intended Purposes* and *Prohibited Intended Purposes.* This structure provides greater flexibility to the access control model. Moreover, by using prohibited intended purposes, one can guarantee that data accesses for particular purposes are never allowed. Conflicts between the allowed intended purposes and the prohibited intended purposes for the same data item are resolved by applying the denial-takes-precedence policy where prohibited intended purposes override allowed intended purposes.

An access purpose is the purpose of a particular data access, which is determined or validated by the system when the data access is requested. Thus, an access decision can be made based on the relationship between the access purpose and the intended purposes of data. That is, an access is granted if the access purpose is entailed by the allowed intended purposes and not entailed by the prohibited intended purposes; in this case we say the access purpose is *compliant to* the intended purpose. The access is denied if any of these two conditions fails; we then say that the access purpose is *not compliant to* the intended purpose.

Data Labeling Model. In order to build an access control model based on the notion of purpose, we must consider a specific data model and based on this model devise a proper labeling scheme. A major question here is at what level of granularity intended purposes are associated with data.

It is clear that in order to make the best use of data while at the same time ensure that data providers feel comfortable, the labeling model should allow the assignments of intended purposes with data at the most fine-grained level. That is, we should be able to assign an intended purpose to each data element in every tuple; e.g., for each attribute and for each data provider. It is also possible that a data provider allows or prohibits access to his/her entire record (i.e., address), not to individual sub-elements (i.e., street, city, state, etc.). This means that it is not always necessary to associate each data element with an intended purpose. However, intended purpose for the address information can vary depending on each individual. To address these concerns, the labeling model should allow the assignment of intended purpose to each tuple of a relation.

However, this most fine-grained approach is not efficient in terms of storage and is not always necessary. For instance, it is possible that there exists some information for which corresponding privacy policies are mandated by enterprises or by laws; i.e., data providers do not have a choice to opt-out from the required intended purposes. In such cases, the data elements in each column in the relation have the identical intended purpose. Thus, in order to avoid any redundant labeling, intended purposes should be assigned to each attribute of a relation using an auxiliary table, e.g., *privacy policy table.* Furthermore, it is possible that the intended purposes of every attribute in a relation be identical.

Such cases occur when information in a relation is meaningful as a whole tuple, but individual elements or tuples do not have any usefulness. In this case, the intended purposes are assigned to the entire relation by using a single entry in the privacy policy table.

In summary, in order to provide privacy protection in a storage efficient way, intended purpose should be assigned to every relation, to every tuple in every relation, to every attribute in every relation, or to every data element in every relation.

Access Control Using Query Modification. Privacy-preserving access control mechanisms must ensure that a query result contains only the data items that are allowed for the access purpose of the query. In other words, the system must check the intended purpose of each data element accessed by the query and filter out its value if the access purpose is not compliant with the intended purpose of the data element. In our approach, this fine-grained access control is achieved using query modification [26]. Our query modification algorithm is outlined in Figure 3. Note that this method is invoked only if the access purpose of the query is verified to be acceptable by the validate function. If the access purpose is not acceptable, then the query is rejected without further being processed.

In Lines 7 and 9 the compliance checks for relations with the relation- or attribute-based labeling schemes are executed statically by the query modification method. On the other hand, the compliance checks for relations with the tuple- or element-based labeling schemes are performed during query processing by the predicates which are added by the query modification algorithm (Lines 15 and 17).

The query modification algorithm checks both the attributes referenced in the projection list and the attributes referenced in predicates (Line 3). As the attributes in the projection list determine what data items will be included in the result relation of a query, it may seem enough to enforce privacy policy based only on the attributes in the projection list. However, the result of a query also depends on the predicates, and not enforcing privacy constraints on the predicates may introduce inference channels. For example, consider the following query:

```
SELECT name
FROM Customer
WHERE income > 100000
FOR Third-Party.
```

Suppose that according to the established privacy policies, *name* can be accessed for the purpose of *Third-Party*, but *income* is prohibited for this purpose. If the privacy constraint is not enforced on the predicates, this query will return a record containing the names of customers whose income is greater than 100,000. This is highly undesirable as this result implicitly conveys information about the customers' income. Note that if the privacy policy is enforced at the predicate level, such inference channels cannot be created.

```
Comp_Check (Number ap, Number aip, Number pip)
Returns Boolean
1.  if (ap & pip) $\neq$ 0 then
2.      return False;
3.  else if (ap & aip) = 0 then
4.      return False;
5.  end if;
6.  return True;

Modifying_Query (Query Q)
Returns a modified privacy-preserving query Q'
1.  Let $R_1$, ..., $R_n$ be the relations referenced by Q
2.  Let P be the predicates in WHERE clause of Q
3.  Let $a_1$, ..., $a_m$ be the attributes referenced in both the projection list and P
4.  Let AP be the access purpose encoding of Q
5.  for each $R_i$ where i = 1, ..., n do
6.    if ($R_i$ is relation-based labeling AND Comp_Check (AP, $R_i$.aip, $R_i$.pip) = False) then
7.      return ILLEGAL-QUERY;
8.    else if $R_i$ is attribute-based labeling then
9.      for each $a_j$ which belongs to $R_i$ do
10.       if Comp_Check (AP, $a_j$.aip, $a_j$.pip) = False then
11.         return ILLEGAL-QUERY;
12.       end if;
13.     end for;
14.   else if $R_i$ is tuple-based labeling then
15.     add ' AND Comp_Check (AP, $R_i$_aip, $R_i$_pip)' to P ;
16.   else if $R_i$ is element-based labeling then
17.     for each $a_j$ which belongs to $R_i$ do
18.       add ' AND Comp_Check (AP, $a_j$_aip, $a_j$_pip)' to P;
19.     end for;
20.   else // $R_i$ is a relation without labeling
21.     do nothing;
22.   end if;
23. end for;
24. return Q with modified P;
```

Fig. 3. Query Modification Algorithm (From [5])

Notice that the provided algorithm filters out a tuple if any of its elements that are accessed is prohibited with respect to the given access purpose. For instance, consider the following query:

```
SELECT name, phone
FROM Customer
FOR Marketing.
```

Suppose there is a customer record of which the *name* is allowed for marketing, but the *phone* is prohibited for this purpose. Then our algorithm excludes the record from the query result. We note that in the environments where partially incomplete information is acceptable, the query modification algorithm can be easily modified to mask prohibited values with null values using the case expression in SQL.

Access Purpose Determination. An access purpose is the reason for accessing a data item, and it must be determined by the system when a data access is requested. Evidently, how the system determines the purpose of an access request is crucial as the access decision is made directly based on the access purpose. There are many possible methods for determining access purposes. First, the users can be required to state their access purpose(s) along with the requests for data access. Even though this method is simple, it requires complete trust on the users and the overall privacy that the system is able to provide entirely relies on the users' trustworthiness. Another possible method is to register each application or stored-procedure with an access purpose. As applications or stored-procedures have limited capabilities and can perform only specific tasks, it can be ensured that data users use them to carry out only certain actions with the associated access purpose. This method, however, cannot be used for complex stored-procedures or applications as they may access various data for multiple purposes. Lastly, the access purposes can be dynamically determined, based on the current context of the system. For example, suppose an employee in the shipping department is requesting to access the address of a customer by using a particular application in a normal business hour. From this context (i.e., the job function, the nature of data to be accessed, the application identification, and the time of the request), the system can reasonably infer that the purpose of the data access must be shipping.

In our work [6], users are required to state their access purposes along with the data access requests, and the system validates the stated access purposes by ensuring that the users are indeed allowed to access data for the particular purposes. To facilitate the validation process, each user is granted authorizations for a set of access purposes, and an authorization for an access purpose permits users to access data with the particular purpose. To ease the management of access purpose authorizations, users are granted authorizations through their roles. This method has a great deployment advantage as many systems are already using RBAC mechanisms for the management of access permissions. This approach is also reasonable as access purposes can be granted to the tasks or functionalities over which roles are defined within an organization. However, using an RBAC mechanism for the management of both access permissions and access purposes may increase the complexity of the role engineering tasks. To address this problem, we introduce a simple extension to RBAC. An important feature of our approach is that by integrating RBAC with attribute-based control, our extension simplifies the role administration and also provides increased flexibility. For more detailed information, users are directed to [6].

4.2 Limiting Disclosure in Hippocratic Databases

LeFevre et al. [17] presented a database architecture for enforcing limited disclosure [2] expressed by privacy polices, which is illustrated by Figure 4. In this section, we briefly review each component of their architecture.

[2] Limited disclosure is one of data privacy principles, which states that data subjects have control over who is allowed to see their personal information and for what purpose [1].

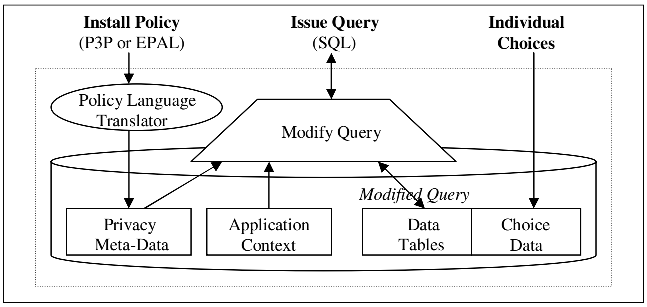

Fig. 4. Implementation architecture overview (From [17])

Privacy Policy Meta-language. A privacy policy, ⟨*data, purpose-recipient pair, condition*⟩ [3], describes to whom the data may be disclosed (i.e., recipients), how the data may be used (i.e., the purposes), and in what specific circumstances the data may be disclosed (i.e., conditions). For instance, a rule ⟨*address, solicitation-charity, optin=yes*⟩ requires that a data subject's address information can be accessed to a charity organization for the solicitation purpose if the subject has explicitly consented to this disclosure. Note that a condition predicate may refer to the data table T as well as any other data tables and the context environment variables (e.g., $USERID). For example, a condition to govern the disclosure of patient data to nurses such that, for treatment, nurses may only see the medical history of patients assigned to the same floor can be expressed as follows:

```
EXISTS ( SELECT NurseID
    FROM Nurses
    WHERE Patients.floor = Nurses.floor
    AND $USERID = Nurses.NurseID)
```

Limited Disclosure Models. For enforcing cell-level limited disclosure, two models are introduced: *table semantics* and *query semantics* [17]. The table semantics model conceptually defines a view of each data table for each purpose-recipient pair, based on privacy policies. Then queries are evaluated against these views. On the other hand, the query semantics model takes the query into account when enforcing disclosure. Despite this subtle difference, the effect of them is the same in that both models mask prohibited values using *null* value. Below we provide the definitions of these two models, taken verbatim from [17].

– (***Table Semantics***) Let T be a table with n data columns, and let K be the set of columns that constitute the primary key of T. For a given purpose-recipient par P_j, the table T, seen as T_{P_j}, is defined as follows:

[3] It is assumed that privacy policies in P3P or EPAL are translated to this form and stored, prior to access control.

$$\{r \mid \exists\, t \in T \wedge \forall\, i,\, 1 \leq i \leq n \\ (r[i] = t[i] \text{ if } \mathit{eval}(t[i,j]) = \text{true},\ ^{4} \\ r[i]= \text{null otherwise}) \\ \wedge\ r[K] \text{ nonenull}\}$$

– (***Query Semantics***) Consider a query Q that is issued on behalf of some purpose-recipient pair P_j and that refers to table T. Query Semantics is enforced as follows:
  1. Every table T in the FROM clause is replaced by T_{P_j}, defined as follows:
     $$\{r \mid \exists\, t \in T \wedge \forall\, i,\, 1 \leq i \leq n \\ (r[i] = t[i] \text{ if } \mathit{eval}(t[i,j]) = \text{true}, \\ r[i]= \text{null otherwise})\}$$
  2. Result tuples that are null in all columns of Q are discarded.

Privacy Policy Storage. The privacy policies (i.e., disclosure rules) are stored in the database as the *privacy meta-data.* These meta-data consists of *policy rules table* and *conditions table.* The policy rules table stores a policy rule as a tuple of ⟨RuleID, PolicyID, Purpose, Recipient, Table, Column, CondID⟩. The condition table stores conditional predicates each of which is identified by a CondID. For instance, a tuple ⟨r1, p1, p, r, t, d, c⟩ in the policy rules table means that the data in d of t is available to the user r for the purpose of p if the predicate identified by c in the conditions table satisfies.

Query Modification. Incoming queries are augmented with case statements [5] to enforce the disclosure rules and conditions specified by the privacy meta-data. For instance, consider the previous example, where nurses may only see the medical history of patients assigned to the same floor for treatment. Suppose a nurse issues a query

```
SELECT history FROM Patients
```

for the purpose of treatment. Then the query is rewritten to enforce the disclosure rules as follows:

```
SELECT
CASE WHEN EXIST
     (SELECT NurseID
      FROM Nurses
      WHERE Patients.floor = Nurses.floor
           AND $USERID = Nurses.NurseID)
THEN history ELSE null END
FROM Patients
```

[4] $\mathit{eval}(t[i,j])$ denotes the boolean result of evaluating the condition that governs the disclosure of data column i in the current row of T to the purpose-recipient pair j.

[5] In [17], a query modification algorithm using outer-joins is also presented.

Observe that the modified query replaces the prohibited history data with *null* values. Note also that the modified query accesses an additional patient data (i.e, "floor"). If this information is governed by another disclosure rule, then such a rule must be also enforced by checking the rule and adding the corresponding condition to the modified query.

5 Generalized Fine-Grained Access Control Models for Database Systems

In recent years, because of privacy requirements, we have seen a growing interest in fine-grained access control in databases; e.g., the VPD in Oracle, parameterized views in DB2 and the Hippocratic databases work. A motivation for fine-grained access control is to move access control from applications to databases, so that they cannot be bypassed and therefore high-assurance privacy can be achieved.

Fine-grained access control policies may be specified according to two different approaches. One approach, that we call view-based approach, uses views or parameterized views. In this approach, what a user is authorized to see is given by a set of views. Unlike in standard SQL, where the user is given access only to the views, the user is given access to the base table; however, the user is limited to access only those parts of the table that is authorized by the views. The other approach uses labeling, where each data element (e.g., a cell or a record) is labeled with information determining whether this element is authorized for a particular user (or a particular query). Purpose based access control discussed in Section 4 is an example of this latter approach. In this section, we review some existing works of the view-based approach and discuss the key challenges posed by fine-grained access control.

5.1 View-Based Approach

In the view-based approach, a user is given a set of views which then represents the parts of the database that are accessible to the user. This differs from the view mechanism in SQL in that users issue their queries against the base tables directly, not using the given views. In this section, we briefly review two works that exemplify the view-based approach.

Access Control in INGRES. Stonebraker and Wong [26] introduced query modification as a part of the access control system in INGRES. The basic idea of query modification is that before being processed, user queries are modified transparently to ensure that users do not see more than what they are authorized to see. In their scheme, an access permission for a user is specified and stored as a view. Thus, each user is associated with a set of views which defines a permitted view of database for the user. When the user issues a query, the query modification algorithm searches for the views that are related to the query; i.e., the views whose attributes contain the attributes addressed by the query [6]. Then the qualifications (i.e., conditions in the WHERE clause) of such views are conjuncted with the qualification of the original query. For instance, consider a table *Employee* with attributes *name*, *department*, *salary*. Suppose a user Smith is allowed to see only information on himself in the *Employee* table. This restrictions is expressed [7] as:

```
SELECT name, department, salary
FROM Employee
WHERE name = ‘Smith’
```

Now suppose Smith wants to find out the salary information on the employees in the *Accounting* department and issues the following query:

```
SELECT name, salary
FROM Employee
WHERE department = ‘Accounting’
```

As the attributes of the given view contain the attributes of the query, the restriction is applied to the query and the query is automatically modified into:

```
SELECT name, salary
FROM Employee
WHERE department = ‘Accounting’ AND name = ‘Smith’
```

Thus, the modified query effectively limits the query result to the information of Smith only. Note that once a query is modified, the modified query can be processed without further access control; that is, the modified query is guaranteed never to violate any access restriction placed on the query issuer.

Motro's Approach. In [21], Motro points out some limitations of the query modification algorithm in INGRES. The main drawback pointed out is that in some cases the algorithm returns less than what the user is actually allowed to see. For instance, consider a relation A with attributes a_1, a_2 and a_3, and assume that a user is given a view permitting her to access the tuples of a_1 and a_2 as long as a condition C is satisfied. Then if the user tries to retrieve the tuples of a_1, a_2, or a_1 and a_2, the access restriction is applied to the query, and only the tuples that satisfy the condition C will be returned. However, when the user tries to retrieve the tuples of all a_1, a_2 and a_3, then the query will be denied as the view given to the user does not contain the all attributes of the query. To address this problem, Motro proposed an alternative technique. Similar to the scheme used in INGRES, views are used to represent statements of access

[6] Note that the attributes of a query include the attributes in both the SELECT clause and the WHERE clause.

[7] While QUEL [20] is used as the query language in the original paper, we use SQL for the convenience of readers.

permissions. However, the main difference is that granting a user permission to access a set of views $V = \{v_1, \ldots, v_m\}$ in Motro's scheme implies that permission is also granted to access any view derived from V. Thus, when the user issues a query Q, which is also a view, the system accepts Q if Q is a view that can be derived from V. In addition, if Q is not a view of V, but any subview of Q is, then the subview is accepted; that is, in the previous example where the user tries to access all a_1, a_2 and a_3, the tuples of a_1 and a_2 that satisfy C will be returned. In [21], this is accomplished as follows. The views that permit access are stored as "meta-relations" in the database. When a query is presented to the database, the query is performed both on the meta-relations and the actual relations. As applying the query on the meta-relations results in a view that is accessible to the user, this result is then applied to the actual query result, yielding the final result. For complete description and examples, readers are referred to [21].

Although the approach of Motro provides more flexibility to access control, his technique also poses some subtle problems. First, his algorithm may result in accepting multiple disjoint subviews of a query. However, it is not trivial how these subviews are presented to users. Another shortcoming of his approach is that the logical structure of actual query result may be different from the expected structure; that is, a query retrieving three attributes may return tuples with only two attributes. This is highly undesirable or unacceptable to most database applications.

5.2 Virtual Private Database (VPD) in Oracle

Since the release of *Oracle8i*, the Virtual Private Database (VPD) has been included as one of major access control components in Oracle database systems. The VPD, defined as "the aggregation of server-enforced, fine-grained access control" [22], provides a way to limit data access at the row level. Surely, it is possible to support row level access control using the view mechanism. However, the view mechanism has several limitations as fine-grained access control mechanism. First, it is not scalable. For instance, consider a table *Employees*, and suppose each employee can only access her own information. In order to enforce such policy using views, administrators have to create a view for each employee and grant each employee a permission to access the personal view. Clearly, this task is not efficient when there are a large number of employees. Also, this approach is cumbersome to support when policies frequently change. Another drawback of using views for access control is that view security becomes useless if users have direct access to base tables.

The VPD overcomes such limitations of views by dynamically modifying user queries. The basic idea of the VPD is as follows. A table (or a view) that needs protection is associated with a policy function which returns various predicates depending on the system context (e.g., current user, current time, etc.). Then when a query is issued against the table, the system dynamically modifies the query by adding the predicate returned by the policy function. Thus, the VPD provides an efficient mechanism to control data access at the row level based on both the system context and the data content.

It is possible to use VPD at the column level; i.e., associate policy functions with a column, not an entire table. In the case of column level VPD, one can also change the behavior of the VPD so that instead of filtering out inaccessible tuples, just the prohibited values are masked with nulls. The VPD also allows one to associate multiple policy functions with the same table. In such case, all policies are enforced with *AND* syntax.

5.3 Truman Model vs. Non-Truman Model

As previously mentioned, query-modification has been widely accepted as an effective technique for implementing access control in relational database systems. With this technique, when a subject issues a query, a modified query is executed against the database. As the modification of queries is transparent to users, this approach is equivalent to providing each user with a view of the complete database restricted by access control considerations and executing her queries only on this view. For this reason, this approach is called the "Truman model" for access control in relational database systems [23].

Although the Truman model can effectively support fine-grained access control, it has a major drawback; there may be significant inconsistencies between what the user expects to see and what the system returns [23]. For instance, consider a relation *Grades* which contains the grades of multiple students, and suppose each student is allowed to access only the grade of herself. Then when a student issues a query `SELECT * FROM Grades`, the Truman model returns only the grade of the student. However, suppose that a student wants to know the average grade and issues a query `SELECT AVG(grade) FROM Grades`. The Truman model effectively filters out the tuples inaccessible to the student. However, the system ends up returning the grade of the student, which is obviously incorrect. However, as such filtering is transparent to the student, the student may falsely believe that her grade is the same as the average grade.

Due to the inadequacy of the Truman model described above, Rizvi et al. [23] argue that query modification is inherently inapplicable as a mechanism for access control in database systems. As an alternative, they propose that every user be associated with a set of authorized views. Then when a user issues a query, the system assesses whether the query can be answered based on the authorized views associated with the user. If the answer is positive, then the query is processed without any modification. If the answer is negative, the query is rejected with a proper notification given to the user. For instance, if the user issues a query that asks for the average grade of all students, and the user's authorized views do not include every student's grade, then the system does not process the query and informs the user that the query cannot be answered.

5.4 Challenges in Fine-Grained Access Control

In general, a query processing algorithm A that enforces fine-grained access control policy takes as input a database D, a policy P, and a query Q, and outputs a result $R = A(D, P, Q)$. We argue that a "correct" query processing algorithm A should have the following three properties.

Soundness. $A(D, P, Q)$ should be "consistent" with $S(D, Q)$, where S is the standard relational query answering procedure. What do we mean by consistent will be formally defined later.

The intuition is as follows. When the policy P allows complete access to D, then $A(D, P, Q)$ should be the same as $S(D, Q)$. When P restricts access to D, $A(D, P, Q)$ may return less information than $S(D, Q)$ does, but it shouldn't return wrong information.

Security. $A(D, P, Q)$ should be using only information allowed by P; in other words, $A(D, P, Q)$ should not depend on any information not allowed by P. To formalize this, we require that for any D', P' such that if the portion of D allowed by P is the same as the portion of D' allowed by P', $A(D, P, Q) = A(D', P', Q)$. Precise definition of "the portion of D allowed by P" will depend on how P is specified, and will be given later.

This security property is inspired by the notion of "indistinguishability" security requirement for encryption schemes [**?**]. It says that if the algorithm A is used for query processing, then no matter what queries one issue, one cannot tell whether the database is in state D, P or in D', P'.

Maximality. While satisfying the above two conditions, A should return as much information as possible. To appreciate the importance of this property, observe that a query processing algorithm that always returns no information would satisfy the sound and secure property; however this is clearly .

We argue that our list of the three properties is intuitive and natural. In particular, it is declarative in that it does not define any procedure for answering queries. We illustrate the importance of having a definition of "correct query processing with fine-grained access control policy" by showing that two approaches previously discussed violate the above properties. For illustration, we consider the following example.

Example 3. We have a relation *Employee*, with four attributes: *id*, *name*, *age*, and *salary*, where *id* is the primary key.

id	**Name**	**Age**	**Income**
1	Alice	22	30000
2	Bob	45	65000
3	Carl	34	40000
4	Diane	28	55000

A policy for a user Alice consists of the following views:

- V_1: Alice can see all information about herself.
- V_2: Alice can see *id*, *name*, and *income* information for anyone whose *income* is less than \$60000.
- V_3: Alice can see *id*, *name*, and *age* for anyone whose *age* is less than 30.

Suppose Alice issues a query:

`SELECT name, age, income FROM Employee` (Q_1)

Using the algorithm in [26], only the tuple about Alice is returned, because neither V_2 nor V_3 includes all three attributes in the query. However, if Alice issues two queries:

`SELECT id, name, income FROM Employee` (Q_2)
`SELECT id, name, age FROM Employee` (Q_3)

and does a natural join, then Alice would also be able to see Diane's information. In other words, Diane's information is indeed allowed by the views; however, the query rewriting algorithm in [26] does not use this information. Thus, the maximal property is violated.

In [17], an access control policy specifies which cells are allowed and which cells are not, and the approach for enforcing such a policy is to effectively replace each cell that is not allowed with the special value null. The soundness property is however violated using the standard SQL approach for handling null values; the standard SQL evaluates any operation involving a null as its operand to false. For instance, suppose that Alice issues a query:

`SELECT name FROM Employee WHERE age < 25` (Q_4).
Observe that Q_4 is equivalent to the following query

```
(SELECT name FROM Employee) EXCEPT
   (SELECT name FROM Employee WHERE age ≥ 25) (Q5)
```

However, using the algorithm in [17], Q_3 returns {Alice} while Q_4 returns {Alice, Bob, Carl}.

Although our correctness criteria seems trivial, to the best of our knowledge, no fine-grained access control algorithm exists that is correct with respect to our definition. Also, devising a "correct" algorithm is not trivial. It seems that this problem of fine-grained access control still remains open.

6 Concluding Remarks

In this paper we have discussed some important requirements towards the development of privacy-preserving DBMS and we have identified initial approaches to address some of these requirements. In particular, we have presented two approaches dealing with purpose meta-data and their use in access control. These approaches represent initial solutions, that need to be extended in various directions. Relevant extensions are represented by efficient storage techniques and the introduction of obligations. We have also discussed current approaches to fine-grained access control and we have outlined the major drawbacks of these approaches. We have also identified three important properties that fine-grained access control models should satisfy. To date, however, no such model exist and its development is an important challenge.

Acknowledgement

The work reported in this paper has been partially supported by NSF under the project "Collaborative Research: A Comprehensive Policy-Driven Framework for Online Privacy Protection: Integrating IT, Human, Legal and Economic Perspectives".

References

1. R. Agrawal, J. Kiernan, R. Srikant, and Y. Xu. Hippocratic databases. In *The 28th International Conference on Very Large Databases (VLDB)*, 2002.
2. R. Agrawal, J. Kiernan, R. Srikant, and Y. Xu. An XPath-based preference language for P3P. In *Proceedings of the Twelfth International World Wide Web Conference (WWW2003)*, pages 629–639. ACM Press, May 2003.
3. A. I. Anton, E. Bertino, N. Li, and T. Yu. A roadmap for comprehensive online privacy policy. Technical Report TR 2004-47, Purdue University, 2004.
4. E. Bertino, E. Ferari, and A. Squicciarini. Trust negotation: Concepts, systems and languages. *IEEE Computing in Science and Engineering*, 6(4):27–34, Jul 2004.
5. J. Byun, E. Bertino, and N. Li. Purpose based access control for privacy protection in relational database systems. Technical Report 2004-52, Purdue University, 2004.
6. J. Byun, E. Bertino, and N. Li. Purpose based access control of complex data for privacy protection. In *Symposium on Access Control Model And Technologies (SACMAT)*, 2005. To appear.
7. C. Clifton. Using sample size to limit exposure to data mining. *Journal of Computer Security*, 8(4):281–308, 2000.
8. C. Clifton and J. Vaidya. Privacy-preserving data mining: Why, how, and when. *IEEE Security and Privacy*, 2(6):19–27, Nov 2004.
9. L. Cranor. P3P user agent guidlines, May 2003. P3P User Agent Task Force Report 23.
10. L. F. Cranor. Personal communication.
11. L. F. Cranor and J. R. Reidenberg. Can user agents acurately represent privacy notices?, Aug. 2002. Discussion draft 1.0.
12. M. M. et al. The Platform for Privacy Preferences 1.0 (P3P1.0) Specification, Apr. 2002. W3C Recommendation.
13. G. Hogben. A technical analysis of problems with P3P v1.0 and possible solutions, Nov. 2002. Position paper for W3C Workshop on the Future of P3P. Available at http://www.w3.org/2002/p3p-ws/pp/jrc.html.
14. G. Hogben. Suggestions for long term changes to P3P, June 2003. Position paper for W3C Workshop on the Long Term Future of P3P. Available at http://www.w3.org/2003/p3p-ws/pp/jrc.pdf.
15. G. Hogben, T. Jackson, and M. Wilikens. A fully compliant research implementation of the P3P standard for privacy protection: Experiences and recommendations. In *Proceedings of the 7th European Symposium on Research in Computer Security (ESORICS 2002)*, volume 2502 of *LNCS*, pages 104–125. Springer, Oct. 2002.
16. M. Langheinrich. A P3P Preference Exchange Language 1.0 (APPEL1.0). W3C Working Draft, Apr. 2002.
17. K. LeFevre, R. Agrawal, V. Ercegovac, R. Ramakrishnan, Y. Xu, and D. DeWitt. Limiting disclosure in hippocratic databases, Aug. 2004. In 30th International Conference on Very Large Data Bases (VLDB), Toronto, Canada.

18. N. Li, J. C. Mitchell, and W. H. Winsborough. Design of a role-based trust management framework. In *Proceedings of the 2002 IEEE Symposium on Security and Privacy*, pages 114–130. IEEE Computer Society Press, May 2002.
19. N. Li, T. Yu, and A. I. Antón. A semantics-based approach to privacy languages. Technical Report TR 2003-28, CERIAS, Nov. 2003.
20. N. McDonald, M. Stonbraker, and E. Wong. Preliminary specification of ingres. Technical Report 435-436, University of California, Berkeley, May 1974.
21. A. Motro. An access authorization model for relational databases based on algebraic manipulation of view definitions. In *The Fifth International Conference on Data Engineering (ICDE)*, pages 339–347, Feb. 1989.
22. Oracle Coperation. *Oracle Database: Security Guide*, December 2003. Available at www.oracle.com.
23. S. Rizvi, A. Mendelzon, S. Sudarshan, and P. Roy. Extending query rewriting techniques for fine-grained access control. In *Proceedings of the 2004 ACM SIGMOD International Conference on Management of Data*, pages 551–562, Paris, France, 2004. ACM Press.
24. M. Schunter, E. V. Herreweghen, and M. Waidner. Expressive privacy promises — how to improve the platform for privacy preferences (P3P). Position paper for W3C Workshop on the Future of P3P. Available at http://www.w3.org/2002/p3p-ws/pp/ibm-zuerich.pdf.
25. D. M. Schutzer. Citigroup P3P position paper. Position paper for W3C Workshop on the Future of P3P. Available at http://www.w3.org/2002/p3p-ws/pp/ibm-zuerich.pdf.
26. M. Stonebraker and E. Wong. Access control in a relational database management system by query modification. In *Proceedings of the 1974 Annual Conference (ACM/CSC-ER)*, pages 180–186. ACM Press, 1974.
27. L. Sweeney. Achieving k-anonymity privacy protection using generalization and suppression. In *International Journal on Uncertainty, Fuzziness and Knowledge-based Systems*, 2002.
28. L. Sweeney. K-anonymity: A model for protecting privacy. In *International Journal on Uncertainty, Fuzziness and Knowledge-based Systems*, 2002.
29. W3C. Platform for privacy preferences (P3P) project. http://www.w3.org/P3P/.
30. R. Wenning. Minutes of the P3P 2.0 workshop, July 2003. Available at http://www.w3.org/2003/p3p-ws/minutes.html.