

Access Control over Uncertain Data

Vibhor Rastogi
University of Washington

Dan Suciu
University of Washington

Evan Welbourne
University of Washington

ABSTRACT

Access control is the problem of regulating access to secret information based on certain context information. In traditional applications, context information is known exactly, permitting a simple allow/deny semantics. In this paper, we look at access control when the context is itself uncertain. Our motivating application is RFID data management, in which the location of objects and people, and the associations between them is often uncertain to the system, yet access to private data is strictly defined in terms of these locations and associations.

We formalize a natural semantics for access control that allows the release of partial information in the presence of uncertainty and describe an algorithm that uses a provably optimal perturbation function to enforce these semantics. To specify access control policies in practice, we describe UCAL, a new access control language for uncertain data. We then describe an output perturbation algorithm to implement access control policies described by UCAL. We carry out a set of experiments that demonstrate the feasibility of our approach and confirm its superiority over other possible approaches such as thresholding or sampling.

1. INTRODUCTION

Access control consists of restricting the set of actions that a user can perform on a certain object. The basic model, introduced by Lampson [12], consists of a matrix that associates to each user and to each object a set of actions. For read-only data, the matrix is boolean, $M[u, t] = 1$ means that user u is allowed to read an object t (e.g. a tuple in a database) and $M[u, t] = 0$ means that she is denied. Advanced techniques such as authorization views [13, 18] allow the data owner to describe the access control policies using a rich, declarative query language that conditions the access to a piece of data on various context parameters, which at run time are implemented as simple, binary decisions, whether to grant or to deny access to the user. Recent applications, however, often need to manage data that is uncertain, and

where the context that controls the access becomes uncertain too. In this paper, we study the access control problem when the data is uncertain: the entries in the matrix are now continuous values between 0 and 1. Our main motivation comes from applications of RFID data.

2. MOTIVATING APPLICATION

The RFID Ecosystem project at the University of Washington [1] deploys several dozens RFID readers throughout the CSE building, and attaches a few thousands RFID tags to people and objects (laptops, books, mugs, etc). Location data about users and objects is collected at the readers and streamed into a trusted central database. Our goal is to restrict read access to the RFID data stored in the database through a set of context-aware rule-based policies that provide individual users control over the release of their data, allow authorized personnel (e.g. fire fighters) access to information about the people in the building, control to what extent owners can track their own objects while in possession of other users, etc. The access control rules incorporate high level context information about the querier and about the subject of the query. The problem in implementing these access control rules is that the collected RFID data is uncertain: both the context that controls whether a piece of information should be released or not, and the information itself is uncertain and modeled as probabilistic data.

2.1 Data

In this paper, we use the RFID Ecosystem as a running example. After it is collected from the antennas, the RFID location data is stored in the `LocatedAt` table, an instance of which is shown in Table 1. The RFID technology is inherently unreliable: readings are often missed, and occasionally false readings are picked up by neighboring antennas. Methods used to cope with the uncertainty in the data s.a. Hidden Markov Models or Particle Filters [11, 16] clean the data by making it probabilistic. Thus, our RFID database is a probabilistic database. For example the tuple (`Alice`, `Coffee Room`, `5.00`, `0.7`) in Table 1 means that the system has only 0.7 confidence that Alice was in the Coffee Room at 5PM. In addition to `LocatedAt`, the RFID database contains other tables: some are auxiliary tables, s.a. ownership and friendship tables and are deterministic, while others are derived from `LocatedAt` and are therefore probabilistic, like the `Carries` table in Table 2 (c). For every tuple t in the RFID database we denote $\Pr_s(t)$ the probability that the system assigns to t .

2.2 Privacy Policy

Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the VLDB copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Very Large Data Base Endowment. To copy otherwise, or to republish, to post on servers or to redistribute to lists, requires a fee and/or special permission from the publisher, ACM.

VLDB '08, August 24-30, 2008, Auckland, New Zealand
Copyright 2008 VLDB Endowment, ACM 000-0-00000-000-0/00/00.

ID	Location	Timestamp	Probability
Alice	Coffee Room	5:00 PM	0.7
Alice	Atrium	5:00 PM	0.3
Bob	Kitchen	5:00 PM	0.4
Bob	Atrium	5:00 PM	0.6
Alice	Alice's office	5:05 PM	1.0
Bob	Bob's office	5:05 PM	0.7
Bob	Alice's office	5:05 PM	0.3

Table 1: LocatedAt Table

User	Object
Alice	Bag
Alice	Purse
Bob	Keys
Bob	Coffee cup

(a) Owns table: User owns Object

User ₁	User ₂
Bob	Alice
Bob	Delta
Grant	Epsilon
Grant	Frank

(b) Advises table: User₁ is the advisor of User₂

User	Object	Prob.
Alice	Bag	0.2
Alice	Purse	0.4
Bob	Keys	0.6
Bob	Cup	0.1

(c) Carries table: User carries Object

User ₁	User ₂	Time
Alice	Bob	11AM
Alice	Charlie	11AM
Bob	Charlie	11AM
Bob	Charlie	3PM

(d) Met table: User₁ met with User₂

Table 2: Some Auxiliary Tables and Inferred Views

We introduce a simple access control language called UCAL. Each rule in UCAL has the form U : IF *context* GRANT *info* meaning “if *context* is true then release *info* to user”. Here U is a label for the rule, and *context*, *info* are conjunctive queries over the RFID database. The *context* has a distinguished variable, $\$u$, which is bound to the current user. We illustrate by showing in Fig 1(a) three rules in English, followed by their UCAL expression in Fig 1(b). These access control rules are used by the system to determine whether to grant or deny read requests by some user $\$u$. For example, the first rule (PAC) says that if a user $\$u$ is located at 1 at time t , then he should be given access to the tuple $\text{LocatedAt}(v,1,t)$ for every possible value of v .

2.3 Access/Deny Semantics

Consider the semantics of UCAL over *deterministic* databases. Suppose that the user $u = \text{Alice}$ asks: *was Bob in the Hall at 5pm?* Consider the PAC rule in Fig. 1: if the rule’s context $\text{LocatedAt}(\text{Alice}, \text{Hall}, 5\text{pm})$ is true then the system will answer the query for Alice: it will return either *true* or *false* depending on whether the tuple $t = \text{LocatedAt}(\text{Bob}, \text{Hall}, 5\text{pm})$ is in the database or not. On the other hand, if the context is false, then the system will deny Alice’s query. The access/deny semantics extends to non-boolean queries as well: if Alice asks “find all persons that were in the Hall between 4pm and 5pm” the system answers it if Alice was in the Hall at all times between 4pm and 5pm, otherwise it denies the query. Rizvi et al. [18] call this semantics the *non-Truman model*.

2.4 Problem Statement

In RFID data, however, the data is uncertain, and it is

impossible to apply the access/deny semantics to uncertain data. For example, Alice’s location is not known exactly: should the system answer the request about Bob? Alice’s locations between 4pm and 5pm are known only with some degree of confidence: should the system return the set of people that she requested? In this paper we define a new semantics for access control rules on uncertain data, and propose an algorithm for implementing this semantics. Our threat model is that of a *systematic leakage of data*, e.g. when an unauthorized user Alice obtains multiple location information about another user Bob (e.g. traces Bob for an entire day), or obtains location information about many users at a given time (e.g. those who were present at yesterday’s meeting). Occasional leakages, however, are tolerated. This corresponds to real life settings where occasional sightings (on a hallway, through a window) or occasional physical encounters between people (at the coffee room, at the bathroom) are considered acceptable risks to privacy, but systematic surveillance of people or locations is not tolerated. Specifically, we make the following contributions:

- We describe a novel approach to access control over uncertain data, based on perturbation, which we call Conditional Perturbation (CP), in Sect. 3.2.
- We give formal definitions for the privacy and the utility of a CP algorithm (CPA) in Sec. 3.3, and describe a provably optimal CPA in Sec. 3.4.
- We describe a rule based access control language, UCAL, in Sec. 4.
- We give the semantics for UCAL in Sec. 5.1, then describe a CPA for UCAL in Sec. 5.2.
- We validate experimentally the privacy, utility and efficiency of the proposed conditional perturbation approach in Sec. 7.

3. THE APPROACH

Consider the simplest access control scenario, with a single user u and a single tuple t . Here t is a Boolean value: 1 indicates the tuple is in the database, 0 that it is not. There is a single UCAL rule IF c GRANT t , where c is the *context* and is also Boolean. Equivalently, we have an access control matrix with a single entry, $M[u, t] = c$. In response to a read request the system returns t when $M[u, t] = 1$ or returns a special value *deny* when $M[u, t] = 0$.

Now assume that the value $M[u, t]$ is a real number p_c in $[0, 1]$, which represents the system’s confidence that the context c is true. Thus, the context now becomes the real number p_c . When the user requests the tuple, the system needs to return some answer r_t . Our goal is to choose r_t s.t. (a) when $p_c = 1$ then there is full disclosure, i.e. $r_t = t$, and (b) when $p_c = 0$ then $r_t = \text{deny}$. In addition we desire a smooth transition between these two extremes when p_c ranges from 1 to 0. We consider two simple options below and discuss their limitations, then we describe our approach. Often the value t is also probabilistic, and replaced with $p_t \in [0, 1]$; when a user wants to read t , we mean that she seeks the value p_t .

3.1 Two Naive Approaches

Thresholding Choose a parameter $0 \leq b \leq 1$. Define r_t as follows: If $p_c \geq b$, then $r_t = p_t$. If $p_c < b$, then $r_t = \text{deny}$.

PAC: If user v is colocated with another user $\$u$ at location l and time t then the information “user v is located at l at time t ” can be released to $\$u$
Lab: If user v is a graduate advisor of $\$u$, o is an object owned by v , and $\$u$ is on the Sixth Floor at time t then the information “object o is in Lab at time t ” can be released to user $\$u$.
Ownership: If user v carries an object o owned by another user $\$u$ at time t then the information “user v is with object o at time t ” can be released to user $\$u$

(a) Plain text

PAC: IF LocatedAt($\$u, l, t$)
GRANT LocatedAt(v, l, t)
Lab: IF Advises($v, \$u$) \wedge Owns(v, o) \wedge LocatedAt($\$u, \text{Floor6}, t$)
GRANT LocatedAt(o, Lab, t)
Ownership: IF Owns($\$u, o$) \wedge Carries(v, o, t)
GRANT isWith(v, o, t)
AS LocatedAt(v, l, t) \wedge LocatedAt(o, l, t)

(b) UCAL

Figure 1: Access control rules translated from plain text to UCAL

This simple approach essentially transforms an uncertain state into a certain one by choosing an arbitrary threshold. The main problem with this approach is that there is a *phase transition* at $p_c = b$, from providing no information when $p_c < b$ to providing total information when $p_c > b$. This is an important limitation, because it makes choosing the right b critical. If b is too low, then there are systematic leakages, and if it is too high the utility of the system decreases. In practice one may have to calibrate b carefully by comparing the system’s decisions with a ground truth, which is in general very expensive to collect. Moreover, different b ’s are required for different kinds of contexts, because probabilities have different semantics: e.g. in `LocatedAt` a probability of 0.8 may be considered very low, but in `Carries` (which is a derived table) even a probability of 0.4 may be considered high. To make access control less sensitive we prefer a method that does not have a phase transition.

Sampling The next simple approach is justified by the *possible worlds semantics* given for probabilistic databases [6]. In this interpretation $M[u, t]$ is in one of two possible states (worlds), 0 or 1, but it is not known which one; we only know their probabilities, which are $1 - p_c$ and p_c respectively. In this approach we sample randomly one of the possible worlds, then apply the access/deny policy on that world. More precisely, set c randomly to 0 with probability $1 - p_c$ and to 1 with probability p_c . Define r_t as follows: if $c = 1$, then $r_t = p_t$; if $c = 0$, then $r_t = \text{deny}$. Thus, the system sometimes grants the query, and other times it denies it. Unlike thresholding, this method does not have a phase transition. Note that the random bit c needs to be set once per user, not per query, i.e. repeated queries by the same user need to result in the same answer¹. However, there is always a probability that the exact p_t is released even when p_c is small, and this leads to a systematic leakage. Suppose Alice probes the database by choosing n arbitrary tuples t_1, \dots, t_n and querying for each of them if it is in the database. Further suppose that $M[\text{Alice}, t_i] = 0.1$ for $i = 1, n$. Alice is almost certainly not allowed to learn any of these tuples. However, if Alice queries systematically each tuple t_i , then the system will grant her access to about 10% of them: this is a systematic leakage.

3.2 Conditional Perturbation (CP) Approach

Our approach to access control in the presence of uncertainty is based on randomly perturbing the answer with an

¹Otherwise an attacker will learn the exact value of p_t by repeatedly issuing the query and waiting for any response other than `deny`.

amount of random noise that depends on the context p_c . When the user asks for the value of t , the system responds, for example, with $r_t = p_t + \hat{n}$, where \hat{n} is a random noise. The noise, \hat{n} , is chosen as a function of p_c : when $p_c = 1$ then we set $\hat{n} = 0$ and therefore the real value p_t is returned unperturbed; when $p_c = 0$ then we set the noise such that the response r_t becomes a random noise, independent of p_t .

Before showing the technical details on how to generate the noise, we illustrate the method from Alice’s perspective, who queries the database for the presence of the tuple t . First, assume that her context is $p_c = 0.9$ (that is, Alice is almost certainly allowed to see t). Suppose the response is $r_t = p_t + \hat{n} = 0.8$. In addition to r_t , the system also reveals p_c (context probability) to Alice. Thus, Alice knows that the noise is low, and believes p_t is close to r_t : since $r_t = 0.8$, she has high confidence that $t = 1$ (i.e. t is in the database). Similarly, if the response is $r_t = 0.2$, then Alice believes that $t = 0$ is much more likely than $t = 1$. Suppose now that her context is $p_c = 0.1$: Alice is almost certainly not allowed to see t . Here the noise is high and if the response is either $r_t = 0.8$ or $r_t = 0.2$, Alice knows that r_t is drawn from almost uniform noise, hence she learns almost nothing about t . Note that the two extremes $p_c = 1$ and $p_c = 0$ correspond precisely to granting access and denying access respectively: when $p_c = 1$ then $r_t = p_t$ and when $p_c = 0$ then r_t is random noise, independent of p_t , which carries the same information as returning $r_t = \text{deny}$.

This approach also extends to non-boolean queries. Suppose Alice asks for all persons who where in the Hall between 4pm and 5pm. The system will return a large number of tuples t , some that are correct answers, some that are not: for each tuple t in the answer the system will indicate r_t (the response) as well as p_c (the degree to which Alice is entitled to know the answer). Alice can then decide for herself which tuples she believes are in the database, based on their r_t and p_c values. In practice, the system computes² $p_c r_t$ for each tuple and selects the top k tuples and returns them ranked in decreasing order of $p_c r_t$. Unlike sampling, this method does not lead to systematic leakage: continuing the example above, if $M[u, t_i] = 0.1$ for all i , then if Alice queries systematically for the n tuples t_1, \dots, t_n then she receives n answers, each of which is almost a random noise. None of the tuples has leaked.

3.3 Definition of Privacy and Utility

²The basic intuition is that Alice wants to know the correct answers (i.e. tuples for which $t=1$). More over, she is entitled to know about $t = 1$ only if context $c = 1$, and $p_c r_t$ reflects the probability of $(t = 1) \wedge (c = 1)$.

We now define formally the privacy and the utility of a Conditional Perturbation Algorithm (CPA). A CPA has two parts: a randomized algorithm returning a response $r_t = A(p_t)$, and estimation function $EST(r_t) \in [0, 1]$ that, given some value r_t estimates the original value p_t . A is given by its probability density function (pdf), $PDF_A(A(p_t) = r_t)$, thus $\Pr[r_t \in [r_1, r_2]] = \int_{r_1}^{r_2} PDF_A(A(p_t) = r_t) dr_t$.

Privacy Our notion of privacy is based on γ -amplification [10] and differential privacy [8]. The difference is that in our setting the degree of privacy needs to be controlled by a parameter: we call this parameter ρ .

DEFINITION 3.1 (ρ -PRIVACY). *Algorithm A is ρ -private for a given $\rho \in [0, 1]$ if the following holds for all possible probabilities p_t, p'_t of t :*

$$\forall p_t, p'_t, PDF_A(A(p_t) = r_t) \geq \rho PDF_A(A(p'_t) = r_t) \quad (1)$$

Intuitively, ρ -privacy restricts how much the algorithm's answer may differ for p_t and for p'_t . When $\rho = 1$ then the answer must be the same (since Eq (1) needs to hold for p_t, p'_t , as well as p'_t, p_t). Hence, if $\rho = 1$, $A(p_t)$ is independent of p_t : no information about t is released. When $\rho = 0$ then ρ -privacy holds for any algorithm: in particular we may simply return the true value, $A(p_t) = p_t$. For $0 < \rho < 1$, only partial information about t can be released.

For access control, the privacy parameter ρ should depend on p_c . If $p_c = 0$, then we would like A to be 1-private. If $p_c = 1$, we would like A to be 0-private. Intuitively, we would like A to be ρ -private where $\rho = 1 - p_c$.

Utility Our utility requirement is that $EST(r_t)$ be a "good" estimate of p_t . More formally, we have the following definition.

DEFINITION 3.2 (UTILITY). *The utility of an algorithm A for a tuple t with value $p_t \in [0, 1]$ is the expected error $\mathbf{E}_A(|EST(r_t) - p_t|)$.*

Lower the expected error, better is the utility. Thus, best utility is attained if $EST(r_t)$ is exactly equal to p_t .

Discussion of privacy An alternative definition of privacy in the literature uses an information theoretic notion (proposed in [10]) that measures the change in the user's knowledge about t : the user may have some prior knowledge about t , but once she sees the response of the algorithm A her knowledge about t changes. A is private when the prior and the posterior knowledge are nearly same. What changes in our setting is that the allowed difference between the prior and the posterior is controlled by the privacy parameter ρ . These two views of privacy are related, as explained in Sec. 6.

An important point is that user's knowledge about system's knowledge (i.e. user's knowledge of how well system knows t) also plays a part, as shown in the example below:

Example 3.3 Consider an algorithm A that returns the response $r_t = \min(p_c, p_t)$. Suppose that $r_t = 0.1$. Assume that the user has no prior knowledge about t . Then the user learns that $p_t \geq 0.1$. This leads to only a small change in her knowledge about t . Now suppose that the user also knows that the system knows t exactly, i.e. $p_t \in \{0, 1\}$. After seeing r_t , the user concludes that $p_t = 1$, and thus learns the exact value of t resulting in a complete leakage. Note that A is not ρ -private, for any $\rho > 0$, because it is deterministic, hence its PDF is ∞ for one value of the response

r_t . In contrast, for ρ -private algorithms, we show that the change in user's knowledge is bounded as a function of ρ , for a user with arbitrary knowledge about the system's knowledge. This is the main intuition of why we use ρ -privacy.

Motivation for utility As stated earlier, maximum utility is attained if $EST(r_t)$ is always equal to p_t . However, such an algorithm is not private. The following proposition describes the difficulty in achieving both privacy and utility.

PROPOSITION 3.4. *There is no unbiased estimator for a ρ -private algorithm if $\rho > 0$.*

Hence we cannot hope to have a ρ -private algorithm with an unbiased estimator. Instead, we will seek for a ρ -private algorithm that minimizes the expected error. This is what we use for our definition of utility.

3.4 An Optimal CPA

Given the requirement of ρ -privacy, we describe here an algorithm which is in a certain sense optimal. First we need a definition:

DEFINITION 3.5. *Consider a random variable \hat{v} over the reals. Its truncation is the random variable $\text{trunc}(\hat{v})$ conditioned by $\hat{v} \in [0, 1]$. More precisely, if f is the probability density function (pdf) for \hat{v} then the pdf for $\text{trunc}(\hat{v})$ is the function g :*

$$g(x) = \begin{cases} f(x) / \int_0^1 f(t) dt & \text{when } x \in [0, 1] \\ 0 & \text{otherwise} \end{cases}$$

The Algorithm The optimal CPA, $A_{opt}(\rho, p_t)$, is given in Algorithm 1; the estimator is the identity, $EST(r_t) = r_t$. Denote $l = \frac{\sqrt{\rho}}{2(1+\sqrt{\rho})}$. The algorithm starts by mapping $p_t \in [0, 1]$ to $\bar{p}_t \in [l, 1-l]$, then adds a random noise \hat{n} , and truncates the result to $[0, 1]$. Notice that the algorithm does not distinguish between values at the lower end $[0, l]$, nor between values at the upper end $[1-l, 1]$.

Algorithm 1 Conditional Perturbation Algorithm $A_{opt}(\rho, p_t)$

Inputs: $\rho \in [0, 1]$, value $p_t \in [0, 1]$

Output A response r_t that is ρ -private.

- 1: Let $l = \frac{\sqrt{\rho}}{2(1+\sqrt{\rho})}$.
 - 2: Let $\bar{p}_t = \min(\max(p_t, l), 1-l)$; thus $\bar{p}_t \in [l, 1-l]$.
 - 3: Return $r_t = \text{trunc}(\bar{p}_t + \hat{n})$.
-

The noise: \hat{n} depends on ρ . Denoting $l = \frac{\sqrt{\rho}}{2(1+\sqrt{\rho})}$, the noise \hat{n} is given by the following pdf:

$$f(x) = \begin{cases} 2l/\rho, & 0 \leq |x| \leq l \\ 2l, & l < |x| \leq 1 \end{cases}$$

The PDF for noise \hat{n} is illustrated in Figure 2. It is symmetric around 0, hence $\mathbf{E}(\hat{n}) = 0$, and it consists of two steps: a higher one close to 0 and a lower one farther away. The width of the central step is $2l$. When $\rho = 1$ then the heights of the two steps become equal ($2l/\rho = 2l$), and \hat{n} degenerates into uniform noise, shown in Fig. 2 (b). When $\rho = 0$ then the central step has width 0 and height ∞ : this corresponds to no noise at all, $\hat{n} = 0$.

Optimality We prove that A_{opt} is both ρ -private, and in some sense optimal among ρ -private, monotone CPAs. A CPA is monotone if the probability of introducing an error decreases as the error magnitude increases:

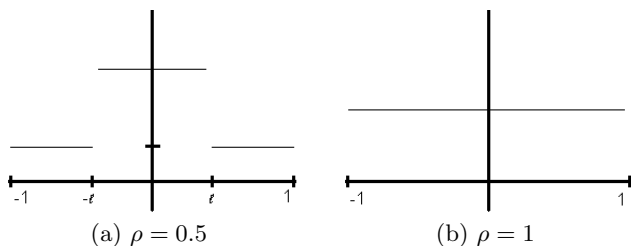


Figure 2: PDF for $\hat{n}(\rho)$

DEFINITION 3.6 (MONOTONICITY). *An algorithm A is monotone if for all p_t and all possible responses r_1, r_2 such that $|r_1 - p_t| \leq |r_2 - p_t|$, $PDF_A(A(p_t) = r_1) \geq PDF_A(A(p_t) = r_2)$.*

THEOREM 3.7. *A_{opt} is monotone and ρ -private. For every $p_t \in [0, 1]$ its expected error is $\mathbf{E}_{A_{opt}}|r_t - p_t| \in [l, 2l]$: moreover, the expected error is $2l$ when $p_t = 0$ or $p_t = 1$, and is l when $p_t = 0.5$.*

The proof is by direct calculation. Note that for $\rho = 0$, $l = 0$ and the algorithm produces no error, while for $\rho = 1$, $l = 1/4$ and the algorithm returns a uniformly distributed random variable, whose expected error is $1/2$ at $p_t = 0$ or $p_t = 1$, and is $1/4$ at $p_t = 0.5$. The following theorem proves that A_{opt} is optimal.

THEOREM 3.8. *Let A be any monotone, ρ -private algorithm. Then for all $p_t \in [0, 1]$ the expected error of A at p_t is at least l . Moreover, there exists $p'_t \in [0, 1]$ such that the expected error of A at p'_t is at least $2l$.*

Proof Sketch We think of ρ -privacy as a constraint for the perturbation function. The utility criteria corresponds to minimizing the error over all monotonic functions that satisfy ρ -privacy. Thus, to get the right perturbation function, we need to solve an optimization problem over functions. To simplify it, we discretize the domain of the candidate functions from the continuous domain of $[0, 1]$ to a discrete domain of n equispaced points in the interval $[0, 1]$. As $n \rightarrow \infty$, the two domains become equivalent. In the discrete domain, we can express the constraint of ρ -privacy and monotonicity as a set of linear constraints. Additionally, minimizing expected error can also be represented as a linear objective function. Thus, in the discrete domain, we can recast the optimization problem as a LP minimization problem. We show that the discretized version of the perturbation function A_{opt} is the optimal solution of this LP problem. This is shown by proving that discretized version of A_{opt} corresponds to a dual feasible solution.

3.5 Discussion

Relating ρ and p_c For access control, the privacy parameter ρ depends on p_c , the probability of context. In sec. 3.3, we used the intuitive function $\rho = 1 - p_c$. In general, any non-increasing monotonic function $\rho : [0, 1] \rightarrow [0, 1]$ can be used. An interesting question is how to choose the ρ function in practice. We want this function to be smooth. For example, if we defined ρ to be a step function:

$$\rho(p_c) = \begin{cases} 1, & 0 \leq p_c \leq b \\ 0, & b < p_c \leq 1 \end{cases}$$

then a CPA becomes equivalent to the threshold approach in Sec. 3.1, which we saw has problems. In practice one may still need to calibrate the ρ function; however, as long as we have a smooth ρ function, with $\rho(p_c) > 0$ for values $p_c < 1$ (unlike the threshold function above) then we have no massive leakage. The advantage of a smooth ρ function is that it is more robust to the (possibly incorrect) choices of the parameters involved. On the other hand, thresholding, due to its discontinuous ρ function, may switch from no disclosure to complete disclosure even for small changes in the threshold parameter b . As discussed earlier, this makes a system using thresholding highly vulnerable to the (possibly incorrect) choices of b . In our application we have found the following two classes of functions to work reasonably well: $\rho(p_c) = \frac{1-p_c}{1+\epsilon p_c}$ and $\rho(p_c) = e^{-\frac{\epsilon p_c}{1-p_c}}$ where $\epsilon \geq 0$ is a parameter. For simplicity, we will use in the rest of the paper the function: $\rho(p_c) = 1 - p_c$.

Non-monotone algorithms Theorem 3.8 only applies to monotone algorithms. To see a counterexample, consider the following naive algorithm $A(p_t) = 0.5$, which returns the constant value 0.5 regardless of p_t . A is ρ -private for any ρ , because it ignores the input. A is not monotone: for example when $p_t = 1$, A returns the response $r_t = 1$ with a probability strictly lower than the probability of returning response $r_t = 0.5$. At $p_t = 0.5$ its expected error is 0. Thus, A shows that Theorem 3.8 does not hold for non-monotone algorithms. However, A is not better than A_{opt} in any practical sense: its expected error at $p_t = 0$ or at $p_t = 1$ is $1/2$, while the expected error of A_{opt} is $\leq 2l$, and l can be arbitrarily small.

4. UCAL

We have discussed the basic principle of access control in the presence of uncertainty in a very simple setting: when the protected data is a single bit, and when the context controlling the access to the data is also one bit. In the following sections we will extend conditional perturbation to a powerful access control language, UCAL.

Syntax A UCAL program consists of a set of rules, each of the form:

$$U: \text{IF } C \text{ GRANT } V$$

Both the *context* C and the *view* V are queries. In this paper we restrict both queries to be unions of conjunctive queries³. An example is illustrated in Figure 1, where all the contexts and all the views are conjunctive queries. We often specify just the body of the conjunctive query, and in this case the head variables are implicitly the variables that appear both in the context and in the view. Alternatively we may name the query and list the head variables explicitly, followed by **AS**, followed by the body. For example the **Ownership** rule in Fig. 1 defines the view:

$$\text{isWith}(v, o, t) \text{ AS LocatedAt}(v, l, t), \text{LocatedAt}(o, l, t)$$

where the head variables are v, o, t , and the non-head variable is l .

Intuitively, the meaning of a UCAL rule is that if the context is true for certain values of the head variables, then the user has access to the data returned by the view, for the same values of the head variables.

Rizvi et al. propose in [18] *authorization views* as a mechanism for fine grained access control to a relational database.

³Equivalently: non-recursive datalog programs.

The semantics is the following: a query q is granted to the user if q can be answered fully from the authorization views, otherwise it is denied. This semantics is called the *non-Truman* model, because reality is never distorted, only denied. By contrast, in a Truman model a query is always answered but returns only those tuples that are certain answers given the authorization view.

UCAL rules generalize authorization views: an authorization view is precisely a UCAL rule consisting only of the view V , i.e. the context is identically **true**.

In the sequel we will give a formal semantics to UCAL rules that (a) extends Rizvi’s semantics for authorization views, and (b) extends the principle of conditional access control in Sec. 3 to arbitrary sets of UCAL rules. We do this by first rewriting UCAL rules into Boolean UCAL rules, then giving semantics to the latter.

5. BOOLEAN UCAL RULES

Consider a query q that is a union of conjunctive queries; we call q a *boolean query* if it has no variables either in the head or in the body⁴. A UCAL rule is a *boolean rule* if both the context and the view are boolean queries. We rewrite every UCAL rule into a set of boolean rules obtained by grounding the context and view queries. A *grounding* of a conjunctive query q is obtained like this: first substitute each head variable with a constant (different choices of these constants result in different groundings); then take the disjunction of all possible substitutions of the other variables with constants. A grounding of a UCAL rule consists of a grounding of its context and of its view, using the same constants for the head variables.

For illustration, consider the query `isWith(v, o, t)` in Sec. 4. One grounding substitutes the head variables as $v = \text{Joe}$, $o = \text{Book73}$ and $t = \text{5PM}$ to get:

$$\begin{aligned} \phi = & \\ & \text{LocatedAt}(\text{Joe}, \text{Rm77}, \text{5PM}), \text{LocatedAt}(\text{Book73}, \text{Rm77}, \text{5PM}) \vee \\ & \text{LocatedAt}(\text{Joe}, \text{Rm78}, \text{5PM}), \text{LocatedAt}(\text{Book73}, \text{Rm78}, \text{5PM}) \vee \\ & \dots \end{aligned}$$

(there is one conjunct for each room in the domain). To obtain a grounding of the `Ownership` rule one has to ground the context with the same constants:

$$\text{IF Owns}(\$u, \text{Book73}), \text{Carries}(\text{Joe}, \text{Book73}, \text{5PM}) \text{ GRANT } \phi$$

The size and number of groundings of a UCAL rule depends on the size of the domain, and this is not intended to be used in practice; we use groundings only to define the semantics. We describe a practical algorithm in Sec. 6.2

Example 5.1 We will use throughout this section the following set of boolean UCAL rules as our running example:

$$\begin{aligned} U_1: & \text{IF } t_1 \text{ GRANT } t_2 t_3 \vee t_3 t_4 \\ U_2: & \text{IF } t_5 \text{ GRANT } t_6 t_7 \\ U_3: & \text{IF } t_8 \text{ GRANT } t_7 \end{aligned}$$

5.1 Semantics of Boolean UCAL Rules

5.1.1 Access/Deny Semantics

We first give the semantics of UCAL over a deterministic database instance I . Denoting Tup the (finite) set of all tuples over a finite domain, we view each tuple as a

⁴We use this definition in this paper although it differs somewhat from common practice where a boolean query means a query without variables in the head.

boolean variable. An instance $I \subseteq Tup$ is a truth assignment to boolean variables, and a boolean query q is a positive boolean formula over variables in Tup . For example, if $I = \{t_2, t_3\}$ and $\phi_1 = t_2 t_3 \vee t_3 t_4$, $\phi_2 = t_3 t_8$ then $\phi_1(I) = 1$ and $\phi_2(I) = 0$. Given a set S of boolean formulas we say that two instances I, I' agree on S if $\forall \phi \in S, \phi(I) = \phi(I')$.

DEFINITION 5.2 (CERTAIN ANSWER). *Let S be a set of boolean formulas, and I an instance. Let ϕ be any given boolean formula. We say that ϕ has a certain answer given S and I if for any instance I' s.t. I, I' agree on S , $\phi(I) = \phi(I')$.*

If ϕ is certain given S and I then we can compute the value $\phi(I)$ by inspecting only the values of $\phi_1(I), \dots, \phi_k(I)$, without looking at the rest of the instance I . For example, referring to ϕ_1, ϕ_2 above, if we know $\phi_1(I) = 1$ and $\phi_2(I) = 0$, then we are certain that $\psi(I) = 0$, where $\psi = t_2 t_8$.

Semantics Consider k boolean UCAL rules `IF c_i GRANT ϕ_i` , for $i = 1, \dots, k$, and let I be an instance. Define:

$$\begin{aligned} C &= \{c_i \mid i = 1, \dots, k\} \\ S_{ok} &= \{\phi_i \mid c_i(I) = 1, i = 1, \dots, k\} \end{aligned}$$

Let A be an algorithm that takes as input a query ψ and returns a response $r = A(\psi, I)$. We define below when A is d-sound (short for deterministic data-sound), i.e. it correctly implements the UCAL rules.

DEFINITION 5.3 (D-SOUNDNESS). *An algorithm A is d-sound with respect to a set of UCAL rules S if for every two instances I, I' that agree on $C \cup S_{ok}$, and for any boolean formula ψ , $A(\psi, I) = A(\psi, I')$.*

Note that the set S_{ok} depends on I , hence it may differ on I and I' . But when I and I' agree on C then S_{ok} is the same for both, hence the notation S_{ok} is well defined.

Consider the following canonical algorithm A_{certain} :

$$A_{\text{certain}}(\psi, I) = \begin{cases} \psi(I) & \text{if } \psi \text{ is certain given } C \cup S_{ok} \text{ and } I \\ \text{deny} & \text{otherwise} \end{cases}$$

One can check that A_{certain} is d-sound. But there exists other d-sound algorithms. For example consider the algorithm that always returns `deny` for every input ψ . This is also d-sound, but arguably not useful at all. A_{certain} turns out to be the most “useful” d-sound algorithm:

PROPOSITION 5.4. (1) A_{certain} is d-sound. (2) For any d-sound algorithm A , if $A_{\text{certain}}(\psi, I) = A_{\text{certain}}(\psi, I')$, then $A(\psi, I) = A(\psi, I')$.

Example 5.5 We examine the effect of the canonical algorithm A_{certain} on the three UCAL rules given in Example 5.1. If the user queries any of the three contexts, t_1, t_5, t_8 , then A_{certain} answers the query. Thus, we may safely assume that the user knows the three contexts. If the user asks a different query ψ , then the A_{certain} ’s response will differ based on the values of these contexts. Suppose t_1, t_5 , and t_8 are all true. Suppose Alice asks the query t_7 : we grant her access, because she is allowed to see t_7 due to the third rule U_3 . Suppose that t_7 is true and Alice asks the query t_6 : we again grant her access, because she is allowed to see $t_6 t_7$ and she is entitled to know t_7 , which is true. Consider now the query t_3 . Here, if the value of the expression $\phi = t_2 t_3 \vee t_3 t_4$ is true, then we return t_3 to Alice (t_3 is certain answer given ϕ); if the expression is false, then we deny the access.

D-soundness extends authorization views in a very strong sense. On one hand, if all the contexts are identically **true**, then the algorithm A_{certain} implements precisely authorization views. On the other hand, given any set of UCAL rules **IF** c_i **GRANT** ϕ_i , for $i \in [k]$, we can associate a set of $2k$ authorization views as follows. The first k views are c_i , for $i \in [k]$; the remaining k views are $c_i \wedge \phi_i$, for $i \in [k]$. The view c_i allows the context to be totally revealed, which corresponds to the semantics of UCAL. Moreover, if c_i is true, then the view $c_i \wedge \phi_i$ becomes logically equivalent to ϕ_i allowing its value to be released. Here too, the semantics of the UCAL rules coincides with that of authorization views in [18].

5.1.2 Probabilistic Semantics

Next, we extend the semantics for UCAL rules when the database instance is probabilistic. A *probabilistic database* PDB is a function $Pr_s : 2^{T^{up}} \rightarrow [0, 1]$ s.t., $\sum_I Pr_s(I) = 1$. For any boolean expression ϕ , its marginal probability is $Pr_s[\phi] = \sum_{I: I \models \phi} Pr_s(I)$. We also call this the *value* of ϕ on the probabilistic database PDB , and use interchangeably the notation $\phi(PDB)$ and $Pr_s[\phi]$. The probability is the system's confidence in the uncertain data, hence the s subscript: the system needs to make grant/deny decisions taking into account these probabilities.

The definition of privacy that we will give below depends on the class \mathcal{C} of probabilistic databases considered. Most of our discussion below applies to any class \mathcal{C} , but we will illustrate three classes in particular: *unrestricted* probabilistic databases (denoted as \mathcal{ALL}), where arbitrary correlations between tuples are allowed, *tuple-independent* probabilistic databases (denoted as \mathcal{IND}), where tuples are independent probabilistic events, and *deterministic* databases⁵ (denoted as \mathcal{DET}).

Let $U = \text{IF } c \text{ GRANT } \phi$ be a UCAL rule and PDB a probabilistic database. To enforce the rule, the system needs to compute the probability of the context formula, $c(PDB)$. We define the *privacy requirement* of U on ϕ to be $\rho(\phi) = 1 - c(PDB)$: this is the amount of privacy that the system needs to ensure for ϕ . When $\rho(\phi) = 0$ then ϕ can be revealed exactly, when $\rho(\phi) = 1$ it needs to be completely private.

Semantics Consider as before k boolean UCAL rules $u_i = \text{IF } c_i \text{ GRANT } \phi_i$, for $i = 1, \dots, k$. Recall that $C = \{c_i \mid i = 1, \dots, k\}$. A perturbation algorithm A receives a user query ϕ and returns a perturbed answer $A(PDB, \phi)$, which we substitute for the true answer $\phi(PDB)$. We will define below when A is u-sound (short for uncertain data-sound), i.e. it implements correctly the privacy requirements of the UCAL rules. Since we allow A to be randomized we need to examine its responses for a sequence of queries: if $\vec{\psi} = \psi_1, \psi_2, \dots, \psi_l$ is a sequence⁶ of queries then $A(PDB, \vec{\psi})$ denotes the sequence of answers $A(PDB, \psi_1), \dots, A(PDB, \psi_l)$.

Let PDB, PDB' be two probabilistic databases that agree on C (i.e. $\forall c \in C, c(PDB) = c(PDB')$). Let $S_{\text{not-ok}} = \{\phi_i \mid \phi_i(PDB) \neq \phi_i(PDB')\}$.

DEFINITION 5.6. *The privacy distance between PDB and PDB' is:*

$$\Delta(PDB, PDB') = \begin{cases} \min\{\rho(\phi_i) \mid \phi_i \in S_{\text{not-ok}}\} & \text{if } S_{\text{not-ok}} \neq \emptyset \\ 1 & \text{otherwise} \end{cases}$$

⁵For all I , $Pr_s[I]$ is either 0 or 1, which further implies $\exists! I. Pr_s[I] = 1$.

⁶In general, $\vec{\psi}$ may contain duplicate queries.

Intuitively, privacy distance $\Delta(PDB, PDB')$ gives how much the algorithm's responses can differ for PDB and PDB' . This is formalized in the definition of u-soundness:

DEFINITION 5.7 (U-SOUNDNESS). *An algorithm A is u-sound with respect to a set of UCAL rules if for all probabilistic databases PDB and PDB' that agree on all contexts C , denoting $\rho = \Delta(PDB, PDB')$, for all query sequences $\vec{\psi}$ and all responses \vec{r} the following holds:*

$$PDF_A(A(PDB, \vec{\psi}) = \vec{r}) \geq \rho \left(PDF_A(A(PDB', \vec{\psi}) = \vec{r}) \right)$$

Intuitively, in the inequality above, if $\rho = 0$ then the algorithm is allowed to answer $\vec{\psi}$ in any way on PDB and PDB' , while if $\rho = 1$ then PDB and PDB' must be indistinguishable from the responses.

We explain now the definition by examining how it achieves our stated goal: to generalize both the semantics of authorization views, and the single-tuple case (given in Sec. 3.3). Consider the case when both PDB and PDB' are deterministic databases. In this case one can check that $\Delta(PDB, PDB') = 1$ if PDB, PDB' agree on S_{ok} , and $\Delta(PDB, PDB') = 0$ otherwise. Thus, u-soundness becomes in this case d-soundness, and therefore it generalizes authorization views. Next, let's turn to the case of a single UCAL rule **IF** c **GRANT** t where both c and t are tuples. Here when two databases PDB, PDB' agree on the context, their distance is $\Delta(PDB, PDB') = 1 - c(PDB)$ if $t(PDB) \neq t(PDB')$ and $\Delta(PDB, PDB') = 1$ if $t(PDB) = t(PDB')$. Hence an algorithm is u-sound iff it is ρ -private for $\rho = 1 - c(PDB)$, and therefore u-soundness generalizes ρ -privacy.

To better understand the power of the u-soundness definition it helps to examine an apparently counterintuitive aspect: the definition seems to ignore the actual query ψ when deciding whether an algorithm is private or not, while we definitely expect in practice to grant different amount of access to different queries ψ . This is indeed the case, as we show below:

Example 5.8 Consider our running example 5.1. Suppose the user asks the query $\psi = t_2t_3 \vee t_3t_4$, which happens to be the view in the first UCAL rule. We will examine how an algorithm needs to behave in order to comply with the u-soundness definition. Given a probabilistic database PDB , the algorithm must return an answer $A(PDB, \psi)$. How much information can A reveal about $\psi(PDB)$? Intuitively we expect this amount to depend on the context in the first rule. To enforce u-soundness the algorithm needs to worry about other databases PDB' that agree with PDB on the three contexts, so consider some other such database PDB' . If $\psi(PDB) = \psi(PDB')$ then the algorithm will return the same answer to ψ in PDB and PDB' (assuming it computes the answer by perturbing the true value of ψ), so let's assume $\psi(PDB) \neq \psi(PDB')$. Then, the set $S_{\text{not-ok}}$ corresponding to the pair PDB, PDB' includes ψ , hence the algorithm must answer ψ with a privacy at least $\rho = \Delta(PDB, PDB') \leq 1 - t_1(PDB)$. On the other hand it is possible to choose a PDB' s.t. $S_{\text{not-ok}}$ contains exactly ψ , and none of the other formulas (e.g. by setting $t_7(PDB) = t_7(PDB')$ and $t_6t_7(PDB) = t_6t_7(PDB')$), and therefore the amount of privacy the algorithm needs to ensure for ψ is exactly $\rho = 1 - t_1(PDB)$. Thus, although the definition of u-soundness does not say this explicitly, the amount of information it allows to be revealed does depend on the query being asked.

5.2 CPA for Boolean UCAL Rules

In the previous section we have given a definition of what it means for an algorithm A to be u-sound for a set of UCAL rules. The definition is not constructive: in order to answer a query ψ on a probabilistic database PDB , $A(PDB, \psi)$, a direct application of the definition requires us to quantify over all other probabilistic databases PDB' , which is impractical. In this section we describe a practical algorithm that is u-sound, based on perturbation.

Given a probabilistic database PDB and boolean formula ψ , the algorithm needs to compute an answer $A(PDB, \psi)$. The first step of the algorithm is to compute a *privacy parameter*, $\rho(\psi)$, for the formula ψ . Intuitively, if ψ is one of the views ϕ_i in the UCAL rules, then $\rho(\psi) = 1 - c_i(PDB)$: we have justified this in Example 5.8. In general, we need to check if ψ is a *certain answer* given the views in the UCAL rules and PDB :

DEFINITION 5.9 (CERTAIN ANSWER). *Let \mathcal{C} be a class of probabilistic databases. Given a set S of boolean formulas, a probabilistic PDB (in \mathcal{C}), we say that some boolean formula ϕ has a certain answer given S and PDB over \mathcal{C} if for any other probabilistic database $PDB' \in \mathcal{C}$ s.t. $\phi_i(PDB) = \phi_i(PDB')$ for $i \in [k]$ we have $\phi(PDB) = \phi(PDB')$.*

For an illustration, let PDB be s.t. $\Pr_s[t_1] = 0.3$ and $\Pr_s[t_2] = 0.4$. Then the formula $\phi = t_1 t_2$ has a certain answer given t_1, t_2 and PDB over all tuple-independent probabilistic database: this is because $\Pr_s[t_1 t_2]$ is uniquely defined as $0.3 * 0.4 = 0.12$. On the other hand, ϕ is not certain given t_1, t_2 and PDB over unrestricted probabilistic databases, because here $\Pr_s[t_1 t_2]$ can range anywhere from 0 to 0.3. For another example, let PDB be s.t. $\Pr_s[t_1] = 0.3$, $\Pr_s[t_2 t_3] = 0.4$ and $\Pr_s[t_1 t_2] = 0$. (Note that PDB cannot be tuple-independent.) Then the formula $\phi = t_1 \vee t_2 t_3$ has a certain answer given $t_1, t_2 t_3, t_1 t_2$ and PDB over all unrestricted probabilistic databases, because $\Pr_s[t_1 \vee t_2 t_3] = \Pr_s[t_1] + \Pr_s[t_2 t_3] - \Pr_s[t_1 t_2 t_3] = 0.3 + 0.4 - 0 = 0.7$.

We can now define the privacy parameter $\rho(\psi)$ for any boolean query ψ . We fix a set of UCAL rules and a probabilistic database PDB . For a subset S of UCAL rules we denote $\text{contexts}(S)$ and $\text{views}(S)$ the set of contexts and the set of views in S . Define $\rho(S) = \max\{\rho(\phi_i) \mid \phi_i \in \text{views}(S)\}$.

DEFINITION 5.10. *Let S be a subset of the UCAL rules. We say that ψ is certain given S, PDB , if ψ is certain given all the views in S and PDB . Then:*

$$\rho(\psi) = \min\{\rho(S) \mid S \subseteq \text{UCAL}, \psi \text{ is certain given } S, PDB\}$$

Example 5.11 Consider the set of UCAL rules defined in Example 5.1. Suppose, we have a tuple-independent PDB specified as: $t_i(PDB) = 0.3$, for $i \in \{1, 2, 3, 4, 5\}$; $t_j(PDB) = 0.9$, for $j \in \{6, 7, 8\}$. Then:

- $\rho(t_7) = 0.1$: This follows from rule U_3 . t_7 is the **view** of U_3 and t_8 is the **context**. Thus, $\rho(t_7) = 1 - t_8(PDB) = 0.1$.
- $\rho(t_6 t_7) = 0.7$: This follows from rule U_2 . $t_6 t_7$ is **view** for U_3 and t_5 the **context**. Thus, $\rho(t_6 t_7) = 1 - t_5(PDB) = 0.7$.
- $\rho(t_6) = 0.7$: This follows from the set of rules $\{U_2, U_3\}$. t_6 has a certain answer given $t_6 t_7$ and t_7 . Thus, $\rho(t_6) = \max(\rho(t_6 t_7), \rho(t_7))$. Thus, $\rho(t_6) = 0.7$.

Computing ρ efficiently As described above, computing $\rho(\phi)$ if ϕ is the view for one of the UCAL rules is simple and efficient. Computing $\rho(\psi)$ for other queries requires determining whether ψ is certain answer. This is decidable, but not necessarily efficient in general:

THEOREM 5.12. *Let \mathcal{C} be a class of probabilistic databases. Denote $\text{CERTAIN}_{\mathcal{C}}$ the decision problem of checking whether a boolean formula ϕ is a certain answer given a set of formulas S and PDB over \mathcal{C} . Then $\text{CERTAIN}_{\text{DET}}$ is coNP-complete [3], $\text{CERTAIN}_{\text{IND}}$ is coNP-hard and $\text{CERTAIN}_{\text{ALC}}$ is in EXPTIME.*

In Sec 6, we discuss some simple cases when certain answers can be checked efficiently. One example occurs frequently in RFID Ecosystem: the issued query ϕ is directly written in terms of views for some UCAL rules. For such queries, ρ can be computed efficiently. In the remaining part of the section, we describe a CPA, which assumes that the $\rho(\phi)$ for each query ϕ has been computed already. It then returns a perturbed response for ϕ based on the privacy requirement $\rho(\phi)$.

CPA The CPA is described in two parts. The first part, Algorithm 2 answers the first k queries. k is a small constant that is given as a parameter to the algorithm. If additional queries are asked, the second part (Algorithm 3) checks individually for each new query whether it can be answered while ensuring u-soundness.

Algorithm 2 Output CPA: Part 1

Inputs: PDB , a query ψ_i from the sequence $\vec{\psi} = \psi_1, \dots, \psi_k$

Output: Response r_i to the query ψ_i

- 1: Let $p_i = \psi_i(PDB)$ be the probability of ψ_i .
 - 2: Let $\rho_i = \rho(\psi_i)$ be the privacy parameter for ψ_i
 - 3: Compute response r_i as $A_{opt}(\rho_i^{1/k}, p_i)$
-

Recall that A_{opt} (Sec. 3.4) is a perturbation function that takes in a privacy parameter ρ_i and a correct response p_i . Then it perturbs p_i by adding noise $\hat{n}(\rho_i)$. Its role in Algorithm 2 is similar. However, instead of using the parameter ρ_i , it uses the parameter $\rho_i^{1/k}$ to allow answering k queries.

Like the algorithm described in [9], Algorithm 2 is an output perturbation algorithm. This means that perturbed probabilities are never stored. Instead the correct response is computed and perturbed only at the end. Algorithm 2 has similar properties as the algorithm in [9]: it can handle arbitrary correlations among the tuples and can only answer a limited number of queries. There are two main differences. Firstly, the privacy requirement here is for boolean formulas ϕ , while in [9] only tuples were considered private. Secondly, the privacy requirement for each formula ϕ is different, and is specified by the parameter $\rho(\phi)$. Thus, it may be possible that revealing the response to a query ϕ , inadvertently compromises privacy of a more private query ψ (ψ is more private than ϕ , if $\rho(\psi) > \rho(\phi)$). Nevertheless, we can show the u-soundness of Algorithm 2 as proved in the following theorem.

THEOREM 5.13. *Algorithm 2 is u-sound.*

Proof Let $\vec{\psi} = \psi_1, \psi_2, \dots, \psi_k$ be any sequence of k queries. Let $\vec{r} = r_1, \dots, r_k$ be the responses returned by the Algorithm 2. Consider any pair of probabilistic databases PDB, PDB' . Let $\rho = \Delta(PDB, PDB')$. For each ψ_i of the given query sequence $\vec{\psi}$, denote $\rho_i = \rho(\psi_i)$. There are two possible cases:

Case 1: $\rho_i < \rho$. In this case, we can show that $\psi_i(PDB) = \psi_i(PDB')$. This is because if $\psi_i(PDB) \neq \psi_i(PDB')$, then it is easy to see that $\Delta(PDB, PDB') \leq \rho_i < \rho$. This is a

contradiction as $\rho = \Delta(PDB, PDB')$. Thus, $\psi_i(PDB) = \psi_i(PDB')$. Next, denoting Algorithm 2 as A , we can show that $PDF_A(A(\psi_i, PDB) = r_i) = PDF_A(A(\psi_i, PDB') = r_i)$ for all possible responses r_i . In other words, the distribution over A 's responses for the query ψ_i is identical for the databases PDB, PDB' . This is because Algorithm 2 generates r_i purely based on the probability of ψ_i , which as shown above is identical for both PDB and PDB' .

Case 2: $\rho_i \geq \rho$. In this case, it may be possible that $\psi_i(PDB) \neq \psi_i(PDB')$. However, as r_i is generated using A_{opt} with parameter $\rho_i^{1/k}$, the following is true:

$$\rho_i^{1/k} \leq \frac{PDF_A(A(\psi_i, PDB) = r_i)}{PDF_A(A(\psi_i, PDB') = r_i)} \leq \frac{1}{\rho_i^{1/k}} \quad (2)$$

Since, $\rho_i \geq \rho$, we have eq(2) implies the following:

$$\rho^{1/k} \leq \frac{PDF_A(A(\psi_i, PDB) = r_i)}{PDF_A(A(\psi_i, PDB') = r_i)} \leq \frac{1}{\rho^{1/k}}$$

Thus, in both case 1 and case 2, the ratio of PDF_A for PDB and PDF_A for PDB' are bounded. Thus,

$$\rho^{(1/k)*k} \leq \frac{PDF_A(A(\vec{\psi}, PDB) = \vec{r})}{PDF_A(A(\vec{\psi}, PDB') = \vec{r})} \leq \frac{1}{\rho^{(1/k)*k}}$$

This proves the u-soundness requirement. \square

As Algorithm 2 is u-sound, we know it is d-sound as well. This can be seen from the boundary cases: For all k , $1^{1/k} = 1$ and $0^{1/k} = 0$. Thus, recalling the properties of A_{opt} (see Sec. 3.4), we see that if $\rho(\psi_i) = 1$, then its response $A_{opt}(1, p_i)$ is completely random noise (corresponding to *deny* case). On the other hand, if $\rho(\psi_i) = 0$, then its response $A_{opt}(0, p_i)$ has no noise at all (corresponding to *allow* case).

Algorithm 3 Output CPA: Part 2

Inputs: PDB , a query ϕ and a sequence of past queries $\vec{\psi}$

Output: Response r to query ϕ

- 1: Let $p_\phi = \phi(PDB)$ be the probability of ϕ
 - 2: Let $\rho = \rho(\phi)$ be the privacy requirement for ϕ
 - 3: Compute $\rho_{check} = \rho^{1/k} \prod_{i: \rho(\psi_i) \geq \rho(\phi)} \rho(\psi_i)^{1/k}$
 - 4: **if** $\rho_{check} \geq \rho$ **then**
 - 5: Privacy for ϕ is protected.
 - 6: **end if**
 - 7: Check similarly the privacy of all other queries in $\vec{\psi}$
 - 8: **if** privacy protected for all queries **then**
 - 9: Return $r = A_{opt}(\rho^{1/k}, p_\phi)$
 - 10: **else**
 - 11: Return $r = A_{opt}(1, p_\phi)$
 - 12: **end if**
-

Additional queries Now we describe Algorithm 3. Its goal is to check whether giving response to a new query ϕ still maintains u-soundness given that a sequence of queries $\vec{\psi}$ have been answered in the past. The basic intuition for Algorithm 3 is that the response given by Algorithm 2 to a given query can never compromise privacy for a more private query. This follows from the case 1 in the proof of theorem 5.13. Thus, to check whether the privacy for a new query ϕ is protected, we just need to consider responses to ϕ and the responses to more private queries. This is done in steps 3 to 6 of Algorithm 3. In addition, we need to check whether the privacy of other queries in $\vec{\psi}$ is preserved after

answering ϕ . This requires checking privacy of only those queries $\psi_i \in \vec{\psi}$ that are less private than ϕ . An important consequence is that, if $\rho(\phi) = 0$, then ϕ will be answered exactly, independent of the sequence of past queries $\vec{\psi}$.

6. EXTENSIONS

6.1 Extensions for the Boolean Case

CPA for Restricted Correlations To handle arbitrary correlations, Algorithm 2 allows only a total of k boolean queries to be answered. In practice, we can answer more queries by relaxing the correlations considered. Specifically, we assume that the tuples in the probabilistic database can be partitioned into sets P_1, P_2, \dots, P_l . Within any partition P_i , there may be arbitrary correlations among the tuples. However, across partitions the tuples are assumed to be independent. We call such databases as partitioned probabilistic databases. This is a common property of many probability distributions [20], and one of the important intuition behind the use of Graphical Models. For such databases, Algorithm 2 can answer k queries about each partition.

One motivation for partitioning comes from RFID data. If we consider the `LocatedAt` table, locations of a single user within a small time interval are obviously correlated. Moreover, a user may be carrying multiple objects. Thus, locations of each of these objects will be correlated. Both these correlations arise in tuples that are read within a narrow time window. If we consider two tuples corresponding to timestamps sufficiently far away, we would expect them to be independent of each other. For example, if the locations of a single user are assumed to satisfy the Markovian assumption, then the locations of two time steps that are sufficiently far away can be assumed to be independent [16].

Thus, in the RFID Ecosystem, each partition corresponds to one correlating event such as a meeting or a coffee break. We assume that each partition corresponds to a time window of at most length L (In practice, L needs to be computed from the data using event detection techniques like those used in [16]). For such partitioned databases, Algorithm 2 can answer k boolean queries for each time window. At the same time, it allows us to handle arbitrary correlations within each time window.

Tuple-independent databases Such databases can be thought of as a limiting case of partitioned probabilistic databases: each partition contains a single tuple. Thus, Algorithm 2 can be used to answer k boolean queries about each tuple of a tuple-independent databases.

Input perturbation In data privacy, there are two main perturbation techniques. There is input perturbation in which data is first perturbed and stored. The response for each query is computed using the perturbed data. On the other hand, there are output perturbation methods that first compute the answer for each query correctly, and then obtain a response by perturbing the correct answer. So far we have discussed just the output perturbation methods, which place restrictions on the number of queries we can answer. We also propose an input perturbation method called sequential CPA. It allows us to answer unlimited number of queries. However, it works only for tuple-independent databases. For details, we refer the full version [15].

Change in users' knowledge We show here the connection between ρ -privacy and the change in user's knowl-

edge. In database privacy [10], user’s prior knowledge about a tuple t is represented using a probability $\text{Prior}(t)$. It represents user’s confidence on t before any information is revealed. After the response r_t is given by the algorithm, the user’s knowledge about t is represented by the probability $\text{Posterior}(t)$. The following theorem shows the connection between ρ -privacy and change in user’s knowledge.

THEOREM 6.1. *Let tuple t be ρ -private according to an algorithm A . Let $\text{Prior}(t)$ and $\text{Posterior}(t)$ be the user’s prior and posterior probability distributions respectively. If $\text{Prior}(t)$ is tuple independent then:*

$$\rho \leq \frac{\text{Posterior}(t)}{\text{Prior}(t)} \leq \frac{1}{\rho}$$

We use $\frac{\text{Posterior}(t)}{\text{Prior}(t)}$ as the privacy metric in our experiments.

6.2 Non-Boolean Case

So far we have considered only the case of boolean UCAL rules and boolean queries. In this section, we briefly explain how to generalize to the non-boolean case. Assume that there are k UCAL rules of the form of: If C_i GRANT V_i , where C_i and V_i are conjunctive queries. Denote the set of rules as S , and the set of views in S as $\text{views}(S)$. We need to answer a conjunctive query Q based on the rules in S .

Computing certain answers As discussed in Sec. 5.2, we first need to compute the tuples in Q that are certain answers given the $\text{views}(S)$. This problem is hard even for the deterministic case. Methods have been proposed to solve this problem approximately in the deterministic setting [7]. The basic idea is the following: rewrite Q into another query Q_v s.t., (1) $Q_v \subseteq Q$, and (2) Q_v is a conjunctive query defined only using views in $\text{views}(S)$. All tuples in Q_v are then ensured to be certain answers given $\text{views}(S)$. Note that this is just a sufficient condition: there may be tuples in Q that are certain answers but do not lie in Q_v . However, for the probabilistic case, even tuples in Q_v may not be certain answers given the $\text{views}(S)$. We need to verify certain additional conditions to determine the certain answers. For this purpose, we use efficient PTIME tests developed in [17]. They provide a sufficient condition for tuples in Q_v to be certain answers for the class of tuple-independent databases.

Answering non-boolean queries As described above, given a conjunctive query Q , it is first rewritten as Q_v , a conjunctive query in terms of $\text{views}(S)$. We describe here how to answer Q_v using our perturbation method. We describe the method for the simplest query $Q_v(\bar{x}) = V_i(\bar{x})$, where V_i is in $\text{views}(S)$ and \bar{x} is the list of head variables of V_i . The method extends similarly for other conjunctive queries defined over $\text{views}(S)$.

We start by computing the query V_i over the probabilistic database (see query evaluation methods in [6]). The result is a set of tuples, with each tuple t in the set having a probability p_t associated with it. As described in Sec. 4, each tuple corresponds to a grounding of head variables \bar{x} of V_i . Moreover, for each tuple t of V_i , there is a grounding of head variables in C_i that results in a context tuple c . The probability p_c of the context tuple controls the privacy requirement of t . Thus, we perturb the probability of each tuple t depending on p_c to get the perturbed response r_t . During perturbation, we also introduce noisy tuples, which were not initially in the query result. This corresponds to perturb-

ing a tuple that has original probability $p_t = 0$. A naive way is to compute the query V_i over the entire domain, and perturb each tuple in the result individually. This is clearly not efficient. However, we note that we need to perturb a tuple t , only if its corresponding context tuple has $p_c > 0$. This greatly reduces the number of noisy tuples that need to be considered. In fact, we can compute efficiently the left outer join of C_i with V_i . The resulting query when evaluated contains all the interesting tuples including the noisy tuples. Then we can efficiently perturb their probabilities using A_{opt} .

7. EXPERIMENTS

We conducted a series of experiments to test the effectiveness of our approach in a real scenario, and to compare it with the simpler methods consisting of sampling and thresholding. We used two kinds of data, which we call *synthetic data* and *real data*. The real data was to test our method in a RFID scenario: here the data was collected by actual RFID readers in real life scenarios. The synthetic data allows a very fine control over each tuple. It is generated as follows: first we create a deterministic table, then we generate probabilities for each tuple by using random values drawn from a gaussian distribution. This allows us to generate both probabilities and have a deterministic ground truth.

Real Data We used the infrastructure of the RFID Ecosystem project at the University of Washington [1]. We collected real data (as part of a larger project) by designing several scenarios involving about a dozen graduate students (called *actors* in these experiments). The scenarios included real life events like meeting, coffee breaks, etc. Actors enacted the scenarios as per predefined instructions while wearing RFID tags. Two sets of data were collected during the experiment. One set was the *application* data, consisting of location data first collected by RFID readers then processed (cleaned) using a particle filter algorithm that is part of the RFID Ecosystem: this resulted in the probabilistic table `LocatedAt`, which consists of 65,434 tuples. We assumed that this data is tuple-independent: in reality some temporal correlations exist between tuples with close timestamps (see Sec. 6.1 for a discussion), but we ignored them in our experiments. The second set of data is *ground truth* data. This was collected by having the actors mark their exact location with a hand-held tablet PC which ran a map-based self-tracking tool. The actors continuously marked their location with the tool during each scenario, generating the deterministic ground truth. This location data was stored in the deterministic `LocatedAtDet` table, which has 12,787 tuples.

Together the application data `LocatedAt` (probabilistic) and the ground truth data `LocatedAtDet` (deterministic) allowed us to conduct a realistic evaluation of the Access Control method proposed in this paper. We performed our tests using the PAC rule in Fig. 1: the rule says that a user Alice is allowed to query the location of a user Bob at some time t only if Alice and Bob were colocated at a time t . Since our scenarios incorporated meetings between actors, we had sufficient data to test this rule.

Boolean First, we evaluate the perturbation method for boolean queries, e.g. $\phi = \text{LocatedAt}(\text{Bob}, \text{Room57}, 5\text{PM})$. PAC rule requires that response to the query depends on Alice’s location, e.g. ϕ should be answered if the context $c = \text{LocatedAtDet}(\text{Alice}, \text{Room57}, 5\text{PM})$ is true. The system

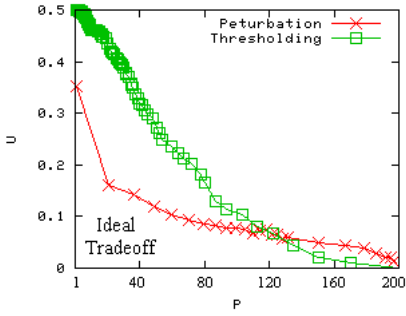


Fig. 7(a) Real Data: Privacy vs. Utility

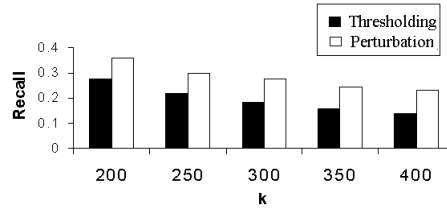


Fig. 7(b) Recall for Top k queries

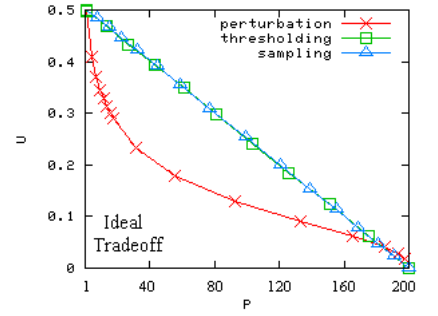


Fig. 7(c) Synthetic data: Privacy vs. Utility

does not have the `LocatedAtDet` data. To answer ϕ , the system first computes p_ϕ using the probabilistic `LocatedAt` table. The system then computes the perturbed response based on p_c , the probability of the context c . The privacy parameter ρ was computed using the function $\rho(p_c) = \frac{1-p_c}{1+\epsilon p_c}$, where p_c is the probability of the context.

We ran multiple such queries, and computed a privacy metric \mathbf{P} and a utility metric \mathbf{U} , by examining the answer returned to the user with the ground truth in `LocatedAtDet`. We explain now how we computed the privacy and utility. For each UCAL rule of the form IF c_i GRANT t_i , we note that a privacy breach occurs at tuple t_i when $c_i = 0$ but information about t_i is revealed. As we saw in Sec. 6, the information leaked about t_i for a user can be quantified as $\frac{Posterior(t_i)}{Prior(t_i)}$. The privacy metric \mathbf{P} measures the amount of systematic leakage: it is the mean leakage for all tuples t_i for which the context c_i is 0.

$$\mathbf{P} = \frac{\sum_{i:c_i=0} \frac{Posterior(t_i)}{Prior(t_i)}}{\sum_i (1 - c_i)}$$

We fixed $Prior(t_i) = 1/200$, i.e. we assumed that a user Alice typically believes tuples like $t_i = \text{LocatedAt}(\text{Bob}, \text{Hall}, 5)$ to have a priori probability $1/200$, that is the user thinks Bob is equally likely to be near any one of the 200 RFID readers in the building. Lower the value of \mathbf{P} , higher the privacy. \mathbf{P} is 1 when no information about t_i is revealed for all i . It is $\frac{1}{Prior(t_i)} = 200$ when complete information about t_i is released for all i .

Loss in utility occurs when $c_i = 1$ but the algorithm does not reveal complete information about t_i . Suppose the algorithm returns the response r_i instead of $Pr_s(t_i)$. The utility loss is then quantified as $|r_i - Pr_s(t_i)|$. The utility metric measures the amount of systematic error: it is the mean error for all tuples t_i for which the context c_i is 1.

$$\mathbf{U} = \frac{\sum_{i:c_i=1} |r_i - Pr_s(t_i)|}{\sum_i c_i}$$

Lower the value of \mathbf{U} , higher the utility. An algorithm that has $r_i = Pr_s(t_i)$ for all i has perfect utility $\mathbf{U} = 0$. An algorithm that returns a uniformly chosen random r_i in $[0, 1]$ has utility $\mathbf{U} = 0.5$.

Motivation behind \mathbf{U} & \mathbf{P} Our utility metric \mathbf{U} is the empirical expected error (corresponding to Def. 3.2). For \mathbf{P} , we would ideally like to use the notion of ρ -privacy. However, we cannot use ρ -privacy as it does not apply to the threshold method (it only applies to randomized algorithms). Thus we use the privacy metric \mathbf{P} of comparing the posterior probability to the prior probability, which is also

a standard privacy definition for perturbation algorithms. Additionally, the notions of ρ -privacy and \mathbf{P} are connected, and Theorem 6.1 formalizes the relationship between them.

Figure 7(a) shows the values for \mathbf{P} and \mathbf{U} values over all possible boolean queries about Bob's location, for the thresholding and the perturbation methods. The plot \mathbf{U} vs. \mathbf{P} for each method was obtained by varying the values of the parameters involved. For thresholding, we vary the threshold. For perturbation, we vary the parameter ϵ used in the function $\rho(p_c) = \frac{1-p_c}{1+\epsilon p_c}$. Note that the ideal region is the lower left corner, i.e. $\mathbf{P} \approx 1$, $\mathbf{U} \approx 0$. Clearly, the perturbation algorithm is closer to this region than thresholding. This implies that if the parameters for thresholding and perturbation are chosen to have the same privacy, the utility for the perturbation method would be much better.

Non-boolean queries We also evaluate the perturbation method for non-boolean queries. Assume Alice issues the conjunctive query $Q(l, t) = \text{LocatedAt}(\text{Bob}, l, t)$. The query asks the locations of Bob at all times during the scenarios. The answer to the query is a list of tuples indicating the location of Bob at different times. Since the data is probabilistic, each tuple is associated a probability value. If Alice issues the query, she would like to view these tuples in decreasing order of their probabilities. That is Alice would like to see the more likely tuples first. Let L be the set of the top- k tuples for Q if there were no access control restrictions. Now we enforce access control using thresholding and using perturbation. The parameters for the methods were chosen so that both thresholding and perturbation have the same privacy metric \mathbf{P} in the boolean experiment described above. Let L_1 be the set of top k answers returned by thresholding. Let L_2 be the set of top k answers returned by perturbation. Since privacy for both methods is fixed to the same value, we compared their utilities. Utility is defined as the recall ratio, i.e. it is the fraction $\frac{|L \cap L_i|}{|L_i|}$. Figure 7(b) shows that perturbation has a higher recall than thresholding. Thus, the fraction of relevant tuples in the top- k result for perturbation is more than that of thresholding.

Performance Here we comment on the space and time requirements of conditional perturbation as compared to the sampling and thresholding methods. Since, we use output perturbation, no perturbed probability is stored. Thus there are no additional space requirements. For time, we note that the requirement comes from three steps: a query evaluation on a probabilistic database, an outer join with the contexts (see sec. 6.2), and finally the output perturbation. The first two are common among sampling, thresholding and perturbation. These two steps are also the most expensive ones. The only step that differs is the output perturbation,

which requires a simple linear pass over the answers to the query. Thus, compared to other methods, the conditional perturbation algorithm does not introduce new performance bottlenecks.

Synthetic data We wanted to reproduce our experiments in a more controlled environment, where we could fine tune the characteristics of the data. We considered a database with n contexts c_i , and n tuples t_i , each guarded by a UCAL rule **IF** c_i **GRANT** t_i , for $i = 1, n$. We set $n = 100,000$. The tuples t_i 's are deterministic (their value $Pr_s(t_i)$ is 1). For the contexts we proceeded as follows. We first randomly chose a ground truth for c_i , either true or false: this ground truth was not known to the system. Then we generated a probability p_i , which is the system's belief about c_i . If $c_i = 1$, then we set $p_i = \hat{N}(\mu_1, \sigma)$. Here $\hat{N}(\mu_1, \sigma)$ represents the gaussian distribution with mean μ_1 and variance σ^2 truncated to the interval $[0, 1]$. If c_i is 0, p_i is obtained as $\hat{N}(\mu_0, \sigma)$. Finally, we evaluated each query t_i , for each $i = 1, n$, and computed the responses returned by each of the three methods: perturbation, thresholding and sampling.

Fig. 7(c) compares the privacy utility tradeoff for the different methods. The graph is obtained as following: we fix the gaussian distribution that is added to simulate uncertainty. We use $\mu_0 = 0.1$, $\mu_1 = 0.9$ and $\sigma = 0.5$. We then plot **U** vs. **P** for each method by varying the values of the parameters involved. Again we see a very similar privacy utility curves for the methods. Perturbation clearly outperforms both sampling and thresholding. The two naive algorithms offer an almost linear tradeoff between privacy and utility, while perturbation has the lower curve.

8. RELATED WORK

The basic principle of access control was originally introduced by Lampson [12]; a good overview of the connection between access control languages and logic can be found in [2]. Recently the database community has studied *fine-grained access control* and has proposed to use views to define access to the items in a database [19, 18]. In [18], the access is defined by a collection of *authorization views*, and the access/deny semantics is defined in terms of certain answers: users are allowed access to answers that are *certain* given the views, and are denied access to all other answers. A basic principle underlying this approach is that of **soundness** [21], which ensures that the information released to the user is never incorrect. Soundness makes the **access/deny** semantics necessary. For example, in [18], **truman** and **non-truman** models for query answering are proposed and the latter is preferred over the former as it ensures soundness.

Most prior work on access control has been in the context of certain data. Recent work [4] has considered access control when the *surveillance* is uncertain and has studied the risks that an illegal action will remain unpunished. We do not consider punishments in this paper, but instead consider the case when the *data* is uncertain, and change the **access/deny** semantics. This means that we relax the soundness requirement: if the system is uncertain, it is unavoidable to occasionally return incorrect answers.

A related problem where the soundness principle is relaxed is that of *privacy* in database; for this problem various

data perturbation methods have been proposed recently [10, 8, 14], and arguably any perturbed data violates the soundness principle. In privacy, a clear distinction is made between public and private information. For example, in a medical database, the individual's disease is considered **private**, whereas statistical facts such as number of individuals having cancer are considered **public**. Here the setting is different: a single piece of information is either **private** or **public** based on the values of some uncertain context.

The idea of adding noise for location privacy protection has also been studied before [5]. However, there the context is deterministic, and the protected data is spatial. In our setting, the context is uncertain, and the protected data is tabular.

9. CONCLUSION

In this paper we studied access control when the data controlling the access is uncertain. We have given a formal definition of privacy in this setting, and proposed a perturbation based access control algorithm that is provably private. We evaluated our approach on real data with uncertainties, from an application of RFID data.

10. REFERENCES

- [1] <http://rfid.cs.washington.edu/>.
- [2] M. Abadi. Logic in access control. In *LICS*, 2003.
- [3] S. Abiteboul and O. Duschka. Complexity of answering queries using materialized views. In *PODS*, 1998.
- [4] P. Balbiani. Acces control with uncertain surveillance. In *International Conference on Web Intelligence*, 2005.
- [5] A. R. Beresford and F. Stajano. Location privacy in pervasive computing. In *IEEE Pervasive Computing*, 2003.
- [6] N. N. Dalvi and D. Suciu. Efficient query evaluation on probabilistic databases. *VLDB J*, 2007.
- [7] O. M. Duschka and M. R. Genesereth. Answering recursive queries using views. In *PODS*, 1997.
- [8] C. Dwork. Differential privacy. In *ICALP*, 2006.
- [9] C. Dwork, F. McSherry, K. Nissim, and A. Smith. Calibrating noise to sensitivity in private data analysis. In *Proceedings of Theory of Cryptography*. Springer, 2006.
- [10] A. Evfimievski, J. Gehrke, and R. Srikant. Limiting privacy breaches in privacy preserving data mining. In *PODS*, 2003.
- [11] S. R. Jeffery, M. N. Garofalakis, and M. J. Franklin. Adaptive cleaning for RFID data streams. In *VLDB*, 2006.
- [12] B. Lampson. Protection. In *Proceedings of the 5th Annual Princeton Conference on Information Sciences and Systems*.
- [13] A. Motro. An access authorization model for relational databases based on algebraic manipulation of view definitions. In *IEEE Data Engineering*, 1989.
- [14] V. Rastogi, S. Hong, and D. Suciu. The boundary between privacy and utility in data publishing. In *VLDB*, 2007. ACM.
- [15] V. Rastogi, D. Suciu, and E. Welbourne. Access control over uncertain data. In *Technical Report*, 2008.
- [16] C. Re, J. Letchner, M. Balazinska, and D. Suciu. Event queries on correlated probabilistic streams. In *SIGMOD*, 2008.
- [17] C. Re and D. Suciu. Materialized views in probabilistic databases: for information exchange and query optimization. In *VLDB*, 2007.
- [18] S. Rizvi, A. O. Mendelzon, S. Sudarshan, and P. Roy. Extending query rewriting techniques for fine-grained access control. In *SIGMOD*. ACM, 2004.
- [19] A. Rosenthal and E. Sciore. Abstracting and refining authorization in sql. In *Secure Data Management*, 2004.
- [20] P. Sen and A. Deshpande. Representing and querying correlated tuples in probabilistic databases. In *ICDE*. IEEE, 2007.
- [21] Q. Wang, T. Yu, N. Li, J. Lobo, E. Bertino, K. Irwin, and J.-W. Byun. On the correctness criteria of fine-grained access control in relational databases. In *VLDB*, 2007.