Database Access Control & Privacy: Is There A Common Ground?

Surajit Chaudhuri
Microsoft Research
surajitc@microsoft.com

Raghav Kaushik
Microsoft Research
skaushi@microsoft.com

Ravi Ramamurthy
Microsoft Research
ravirama@microsoft.com

ABSTRACT

Data privacy issues are increasingly becoming important for many applications. Traditionally, research in the database community in the area of data security can be broadly classified into access control research and data privacy research. Surprisingly, there is little overlap between these two areas. In this paper, we open up a discussion that asks if there is a suitable middle-ground between these areas. Given that the only infrastructure provided by database systems where much sensitive data resides is access control, we ask the question how the database systems infrastructure can step up to assist with privacy needs.

1. INTRODUCTION

Data privacy issues are becoming increasingly important for our society. This is evidenced by the fact that the responsible management of sensitive data is explicitly being mandated through laws such as the Sarbanes-Oaxley Act and the Health Insurance Portability and Accountability Act (HIPAA). Accordingly, data privacy has received substantial attention in previous work [3]. The key technical challenge is to balance utility with the need to preserve privacy of individual data. Initial work on data privacy focused on data publishing where an "anonymized" data set is released to the public for analysis. However, the evidence increasingly points out the privacy risks inherent in this approach [9]. It is now believed that a query-based approach where the database system *answers a query in a privacy-sensitive manner* is generally superior from the privacy perspective to the data publishing paradigm [9].

However, the only support provided by database systems where much sensitive structured data reside is the mechanism for *access control*. Briefly, the idea is to authorize a user to access only a subset of the data. The authorization is enforced by explicitly rewriting queries to limit access to the authorized subset. Such a model of authorization is intuitive to application developers and users of the database system. The programming model remains the same as before — (1) data is accessed using SQL queries, (2) the results returned are deterministic, hence the utility of query results is clear, (3) there is no restriction on the class of queries that can be executed, and (4) there is no restriction on the total number of query executions. In other words, an access control mechanism is fully compatible with the functionality of a general purpose database system. Not surprisingly, access control is supported in all commercial database systems and the SQL standard. (In fact, some commercial database systems also support fine-grained access control [11].)

The main limitation of the traditional access control mechanism in supporting data privacy is that it is "black and white". Consider for example advanced data analysis tasks such as location-aware services of operational Business Intelligence that need to stitch together multiple sources of data such as sales history, demographics and location sensors many of which have sensitive data. For these examples, effective analytics needs to leverage many "signals" derived by aggregating data from sources who have sensitive private information, at the same time without revealing any sensitive individual information. But the access control mechanism offers only two choices — (1) release no aggregate information thereby preserving privacy at the expense of utility, or (2) release accurate aggregates thus risking privacy breaches for utility.

There is considerable previous work in privacy-preserving query answering that goes beyond the "black and white" world of access control [1, 8]. The class of queries is generally restricted to aggregate queries and the approach adopted broadly is to add noise to the aggregates. However, non-aggregate queries are a large class of database queries and the support for them is rather limited in the above bodies of previous work.

In this paper, we ask if we can get the best of both worlds — combine the advantages offered by access control mechanisms while at the same time going beyond the "black and white" world by leveraging previous work on privacy preserving query answering. We can break down this question as follows: (1) What is the database API that combines traditional access control mechanisms with privacy-preserving query answering? (2) How do we implement the suggested APIs in a principled manner? (3) How do we mix and match both mechanisms to ensure privacy guarantees?

We explore a natural hybrid system that combines (a) a set of authorization predicates restricting access per user to only a subset of the data, and (b) a set of "noisy" views that (as the term suggests) expose perturbed aggregate information over data not accessible through the authorization pred-

5th Biennial Conference on Innovative Data Systems Research (CIDR '11)
January 9-12, 2011, Asilomar, California, USA.

icates. For example in an employee-department database, we could allow employees to see their own employee record and also publish a "noisy" view exposing the average salary of the organization. This hybrid potentially has significant advantages. Accessing data through a set of views is natural for users of database systems and thus provides a simple extension to today's database API. It offers the functionality of access control for queries that refer only to the database tables and views, and the privacy guarantees of previous work for queries that only refer to the "noisy" views. In addition, we obtain value beyond the sum of the individual parts by allowing rich queries that refer to *both* database tables and "noisy" views.

We implement the "noisy" views by using previous work on implementing *differential privacy* [5, 8]. In order to answer question (3) above, we introduce the notion of differential privacy *relative to an authorization policy* and explain its desirable theoretical properties. We show that the seemingly ad hoc hybrid system described above satisfies differential privacy relative to the authorization policy.

We note that it is possible to expose the APIs we propose by extending libraries supporting differential privacy such as PINQ with access control mechanisms. However, given that the recent trend in the industry is to implement data security primitives in the database server, our discussion in this paper is presented as a modification to the database server. We think of our paper more as a first step in initiating discussions on how database systems can provide meaningful support to address privacy concerns. We comment on open issues, including a critique of state of the art privacy models.

2. REVIEWING ACCESS CONTROL

There has been a lot of work in the area of access control in databases. The idea with access control is that each database user gets access to a subset of the database that the user can query. The current SQL standard allows coarse grained access both to database tables as well as views.

Recent work [2, 6, 11, 13] has emphasized supporting predicate based *fine-grained* access control policies in the database server. For example, we wish to be able to grant each employee in an organization access their own record in the employee table.

In this paper, we consider fine-grained access control policies. We assume that access control policies expose per user a subset of each database table (this is the approach adopted by some commercial systems like Oracle VPD). The policy is formally captured by specifying for each user and each database table, an *authorization predicate.* Since the number of users can be potentially large, the predicate is specified succinctly as a parameterized predicate that references the function *userID*() that provides the identity of the current user. For example, we can grant each employee access to their own record as follows. (We adopt the syntax proposed in previous work [2].)

```
grant select on employee
      where (empid = userID())
to public
```

Access can be granted not only to tables but also to views. In this way, we can expose additional information such as aggregate information. For example, we can create a view that counts the total number of employees and grant access to the view to every employee. For ease of exposition and without loss of generality, in the rest of this paper, we restrict authorization predicates to only be specified for tables.

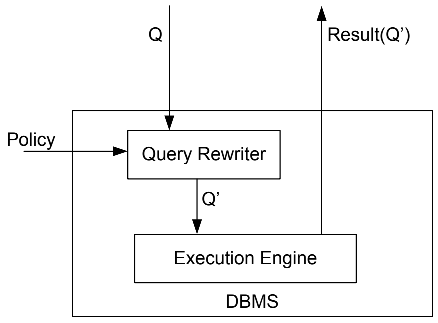

Figure 1: Leveraging Fine-Grained Access Control

We now formalize the notion of an *access control policy.*

Definition 1. A *access control policy* $\mathcal{P}$ specifies for each user and for each database table, a corresponding authorization predicate. We also refer to an access control policy as an *authorization* policy.

We assume that the function *auth(T, u)* denotes the authorization predicate on table T corresponding to user u. For instance, in the above example *auth(T, u)* is the predicate `empid = u`.

Let the tables in the database be $T_1, \ldots, T_k$. Fix a database instance A. We refer to the vector $<\sigma_{auth(T_1,u)}(T_1), \ldots, \sigma_{auth(T_k,u)}(T_k)>$ as evaluated on the instance A as the *authorized subset* for user u, denoted $\mathcal{P}(u, A)$. (In all our notation, if the user is clear from the context, we drop the reference to the user.)

Queries are executed by *rewriting* them to add the authorization predicates. For example, the query:

```
select salary from employee
```

gets rewritten to:

```
select salary from employee
where empid = userID()
```

In the same way, update statements are also rewritten to go only against the authorized subset. For example, the update:

```
update employee set nickname = 'Jeff'
```

is rewritten as follows:

```
update employee set nickname = 'Jeff'
where empid = userID()
```

We note that in general, the access control policy can allow a different "pre-update" authorization predicate than the one for queries. Further, previously proposed access control policy languages also support the notion of a "post-update" authorization predicate which checks whether the new values of the updated rows are authorized. For ease of exposition, we assume that (1) there is only one authorization predicate used for both queries and pre-update and that (2) there are no post-update predicates, while noting that our techniques and results extend if we relax this assumption.

Figure 1 illustrates the infrastructure supporting access control mechanisms in a database system.

2.1 Limitations

As we discussed in Section 1, granting access to accurate aggregations over different subsets of the data can potentially leak information. For instance, in an Employee-Department database, suppose we grant a data analyst access to the number of employees at various levels of the organizational hierarchy grouped by the department, gender and ethnicity. If the analyst knows an employee with a rare ethnicity, the level of the employee could potentially get breached. Basically, the choices offered by access control mechanisms are "black-and-white"; we can grant access either to accurate aggregate information thereby compromising privacy, or to no aggregate information compromising utility.

Previous work in data privacy has studied techniques for releasing information while preserving privacy allowing us a middle ground in the above scenario. Briefly, the idea is to add noise to the result of a computation. We next address the question of how the database API can be extended to exploit this previous work.

3. NOISY VIEWS

In Section 1, we proposed the notion of *noisy views* as a possible abstraction for integrating privacy mechanisms in a traditional database. The core properties of noisy views that we support may be summarized as follows:

- *Noisy Views are a DDL construct:* Noisy views may be defined by a data provider in the same way a traditional view is declared, e.g., through a CREATE VIEW statement.
- *Noisy Views are Non-Deterministic:* Although the DDL expression for a noisy view is no different from that of a traditional view, their semantics is non-deterministic. Intuitively, they correspond to the result of executing the DDL associated with the noisy views but enriched with a random "noise" to ensure no privacy leaks happen.
- *Noisy views and Access Control co-exist:* A query can reference both a noisy view and other authorized objects.
- *Noisy Views are Access Control Aware:* Noise is only added to the part of the result derived from unauthorized data. The part derived from authorized data is not perturbed.
- *Noisy Views are Named:* To reflect to the application developer the fact that noisy views are non-deterministic, the noisy views need to be explicitly named, much like views in the SQL standard.
- *Traditional DBMS Execution Engine:* The DBMS query execution engine is left unchanged. Only minimal changes to the database system are needed: (1) a noise-injecting function and (2) a modified query rewriter that rewrites a reference to a noisy view.
- *No Change in Data:* The privacy-sensitive database content is left unchanged.

The above properties summarize the core facets of a noisy view object. The advantages of this approach are that (1) it builds on the familiar notion of views thereby inheriting the benefits of access control mechanisms, and (2) by requiring that queries access the noisy views by explicitly naming them, it clearly separates the deterministic components from the non-deterministic components of the system. Any query that does not name the noisy views will have the same behavior as with just access control mechanisms.

However, the specific details of the privacy model being supported determines several additional details: (1) What subset of SQL can be supported as noisy views? (2) How exactly is the noise added? (3) What is the additional information that needs to be specified: for users as well as queries? (4) What are the privacy guarantees? The answers to the above questions critically depend on the specific privacy model that is adopted.

The rest of the paper answers precisely these questions for the well-known model of differential privacy [5]. We first review state of the art differential privacy in Section 4 and then describe our implementation in Section 5. Finally, we note that the model of noisy views is more general and provides a template for capturing privacy models in database systems like the approach adopted by Netz et al. [10] for encapsulating data mining models in databases.

4. REVIEWING DIFFERENTIAL PRIVACY

We now briefly discuss *differential privacy* [5] which is considered to be the current state of the art in privacy models.

4.1 Definition

Intuitively, differential privacy requires that computations be formally indistinguishable when run with and without any single record. The following definition makes the intuition precise. We denote the symmetric difference between two data sets A and B as $A \oplus B$ (for a database with multiple tables, we take the union of the symmetric difference of the corresponding tables).

Definition 2. A randomized computation M provides ϵ-differential privacy if for any two database instances A and B and any set of possible outputs $S \subseteq Range(M)$:

$$\mathbf{Pr}[M(A) \in S] \leq \mathbf{Pr}[M(B) \in S] \times \exp(\epsilon \times |A \oplus B|)$$

□

The parameter ϵ quantifies the degree of privacy. A smaller value of ϵ corresponds to a stronger guarantee — for example if we set ϵ to be 0, then M is constrained to produce the same output independent of input. On the other hand, a larger value of ϵ indicates a weaker guarantee.

Intuitively, differential privacy ensures that adding an individual record to the database does not reveal much additional information. Thus, an adversary would not be able to learn if any particular data item was used as a result of this computation. Perhaps the biggest advantage of differential privacy is that it makes no reference to (and hence no assumptions about) background knowledge. It thus relieves us from the burden of changing privacy models as assumptions about background knowledge change.

4.2 Implementation Using PINQ

Most techniques implementing differential privacy such as Privacy Integrated Queries (PINQ) [8] focus on aggregations (our discussion below is also based on PINQ). Aggregates are supported by *output* perturbation — the original aggregate is first computed and then perturbed by adding random noise (thus the noise corresponds to an absolute error in the output).

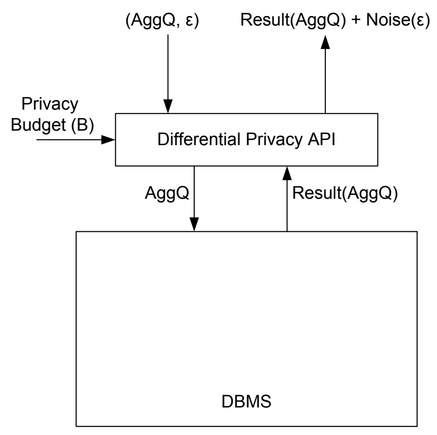

Figure 2: Leveraging Differential Privacy

Figure 2 illustrates how an application can leverage PINQ. Each aggregate query *AggQ* is issued with a privacy parameter ϵ (see Definition 2). The query execution algorithm guarantees ϵ-differential privacy by adding a carefully chosen random noise to the output; the noise is chosen as a function of ϵ and the aggregation being performed. The noise added is inversely related to ϵ — a larger value of ϵ (weaker privacy guarantee) can be accommodated with a smaller noise, whereas a smaller value of ϵ (stronger privacy guarantee) requires more noise.

As more queries are run, the overall privacy guarantee gets weaker. Formally, we have the following previously published result [8].

THEOREM 1. *Let M_i each provide ϵ_i-differential privacy. Then the sequence of M_i provides $\Sigma_i\epsilon_i$-differential privacy.*

Thus, overall the system satisfies $\Sigma_i\epsilon_i$-differential privacy where the i^{th} query is run with parameter ϵ_i.

There is previous work [4] that formally shows that if an unbounded number of queries are allowed, then eventually privacy is breached. Therefore, the notion of a *privacy budget B* is introduced that bounds the number of queries a user can run. As each query is run with its privacy parameter ϵ, the budget is decremented by ϵ. Queries can only be run so long as permitted by the remaining budget. A larger privacy budget allows a larger number of queries to be run but with a greater risk of privacy breach. Thus, both the budget and the query-specific privacy parameters can be used to trade off privacy with utility.

PINQ supports differentially private variants for all the standard SQL aggregations such as `sum`, `count` and `average`. We refer to the differentially private variants respectively as `noisySum`, `noisyCount` and `noisyAvg`. The aggregations can be computed over a restricted class of SQL operations such as filters and key-key joins. By using a partitioning operation, it also supports a limited form of grouping. Since the implementation only adds noise to the output of queries, all of the query processing can be done using the DBMS execution engine without modifying the underlying data (this is in contrast with input perturbation techniques [3]).

4.2.1 Limitations

As noted above, using differential privacy requires the programmer to set various parameters and also understand that an unbounded number of queries cannot be run by the same user. While the meaning of the parameters is intuitive, choosing appropriate values for them is non-trivial. The random noise added is a function not only of the privacy parameter ϵ but also the *sensitivity* of the aggregation, which is the maximum influence any single record can have on the output of the aggregation. Differential privacy implementations work best for low-sensitivity computations. For example, the sensitivity of aggregates such as `count` is low. On the other hand, for aggregates such as `sum` the sensitivity can be arbitrarily large (that said, when a large number of records are being summed, a large absolute error can be tolerated since it might not correspond to large relative error). For a given privacy guarantee ϵ, we need to add significantly more noise as the sensitivity increases. On the other hand, answering higher sensitivity queries without increasing the noise reduces the total number of queries that can be executed within the privacy budget. Such interactions between the parameters makes them difficult to set. We note that the above limitations hold not only for PINQ but for all previously proposed differential privacy algorithms.

Although differential privacy comes with the above "baggage", it offers a principled mechanism to navigate the privacy/utility trade off. This is in stark contrast to the "black-and-white" world of access control. Further, recent empirical work has begun to apply differential privacy to several real-world data analysis tasks successfully [8, 14]. Given this fact, it is natural to ask if we can implement our noisy view abstraction through differential privacy primitives. We study this question next.

5. DIFFERENTIALLY PRIVATE NOISY VIEWS

In this section, we discuss how we implement noisy views based on differential privacy and discuss how it can be integrated in a database system. We term noisy views implemented using differential privacy as *differentially private views* or *DPViews* in short. We present our system as an enhancement of the database server while noting that much of the functionality can also be supported through middleware requiring no changes to the server, say by enhancing PINQ with access control primitives.

Since the class of DPViews we consider is influenced by our privacy guarantees, we begin this section by discussing the privacy guarantee we seek to provide.

5.1 Differential Privacy Relative To Views

The main challenge in formalizing the privacy guarantee is that we do not want to charge the system with protecting the privacy of information that is revealed through the authorization policy. We illustrate with an example. We assume that the authorization predicates are known to all users.

Example 1. Suppose that in an organization, a user is authorized to see the records of all employees whose salary is greater than \$100000. Even though the user is not authorized to see the records of other employees, he/she knows that their salary is less than or equal to \$100000. □

A user knows that the underlying database has to be consistent with the authorized subset. This is how information is revealed about the overall data. Accordingly, we introduce the notion of differential privacy *relative to* an authorization policy as follows.

Definition 3. We say a randomized computation M provides ϵ-differential privacy *relative to an authorization policy* $\mathcal{P}$ for user u if for any two database instances A and B such that $\mathcal{P}(u, A) = \mathcal{P}(u, B)$ and any set of possible outputs $S \subseteq Range(M)$:

$$\mathbf{Pr}[M(A) \in S] \leq \mathbf{Pr}[M(B) \in S] \times \exp(\epsilon \times |A \oplus B|)$$

□

We note that the main difference from the usual notion of differential privacy is that we only consider instance pairs that agree on the authorized subsets. Definition 3 offers us a principled way to reason about privacy in the context of a given access control policy. At one end, if the user is not granted access to any data, then Definition 3 reduces to standard differential privacy. By granting access to more data, the system is only charged with providing a weaker privacy guarantee.

Finally, we carry out the discussion in this section for a single user noting that all results generalize to multiple users. So the references to the user are dropped in the notation.

5.2 Overall Architecture

The overall policy specified to the system initially consists of an authorization policy and a set of DPViews. We use the access control infrastructure to grant and/or deny access to the DPViews in the same way as with traditional views.

In general, a query can reference database tables and DPViews. An update can only reference database tables. Queries and updates are executed by rewriting them in a way that guarantees differential privacy relative to the authorization policy $\mathcal{P}$.

References to database tables are rewritten as described in Section 2 to reflect the authorization policy. For queries and updates that do not reference DPViews, the behavior is identical to only having an authorization policy. The differential privacy guarantee is obtained via the following result.

THEOREM 2. *Fix an authorization policy $\mathcal{P}$ and a query that does not refer to DPViews. The rewritten query is* 0*-differentially private relative to $\mathcal{P}$. Similarly, an update (that is not allowed to refer to DPViews) when rewritten is* 0*-differentially private relative to $\mathcal{P}$.*

PROOF. We can think of the authorization semantics for queries and updates as follows. The database instance A is replaced with the authorized subset $\mathcal{P}(A)$ and the original unrewritten statement (query or update) is run on this smaller instance. The result follows. □

Any reference to a DPView in a query is made with a privacy parameter ϵ (the ϵ here refers to Definition 3) and rewritten in a way that guarantees ϵ-differential privacy relative to the authorization policy. The way in which DPView references are rewritten is described in Section 5.3.

In general, a query can reference *both* the database tables and the DPViews. Such queries are also issued with the privacy parameter ϵ. We prove below in Theorem 4 that their rewriting guarantees ϵ-differential privacy relative to the authorization policy.

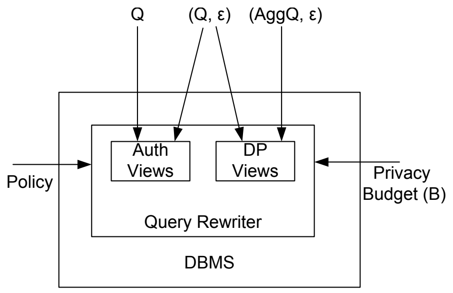

Figure 3: Leveraging Access Control and Differential Privacy

As multiple queries and updates are run, we prove that the system satisfies $\Sigma_i\epsilon_i$-differential privacy relative to the authorization policy where the i^{th} query is run with parameter ϵ_i.

THEOREM 3. *Fix an authorization policy $\mathcal{P}$. Let M_i each provide ϵ_i-differential privacy relative to $\mathcal{P}$. Then the sequence of M_i provides $\Sigma_i\epsilon_i$-differential privacy relative to $\mathcal{P}$.*

PROOF. The proof is almost identical to the analogous previously known result [8] (stated in Theorem 1) — we only have the additional restriction of focusing on instances A and B with $\mathcal{P}(A) = \mathcal{P}(B)$. □

Together with the following straightforward result, Theorem 3 can be used to infer that queries that reference database tables and a DPView with parameter ϵ satisfy ϵ-differential privacy relative to the authorization policy.

THEOREM 4. *Fix authorization policy $\mathcal{P}$. Any deterministic computation that operates on the output of a computation that is ϵ-differentially private relative to $\mathcal{P}$ is also ϵ-differentially private relative to $\mathcal{P}$.*

We also inherit the notion of a per-user privacy budget which is maintained in the same way as in PINQ. The overall architecture is shown in Figure 3.

5.3 Differentially Private Views

We now discuss the class of DPViews we support coupled with how we rewrite them to yield privacy guarantees. The class of queries we encapsulate as DPViews is based on the class of queries for which differentially private algorithms are known [8]. If differentially private algorithms are developed for larger classes of queries in the future, we could correspondingly accommodate a larger class of DPViews.

We note that when the user budget is exhausted, the DPView expression is always rewritten in accordance with the authorization policy while signaling to the application that the privacy budget is exhausted. Thus, the discussion below focuses on the rewriting method for the case when the user budget is not exhausted.

5.3.1 Single Table DPViews

We start our discussion with single-table DPViews. Suppose that a user is only authorized to see a subset of table T. In order to expose privacy-preserving computations on the unauthorized subset of T we can declare a DPView as follows.

```
create noisy view NoisySelect(agg1,...,aggn) as
(select scalarAgg(A1),...,scalarAgg(An)
 from T)
```

We consider standard SQL aggregations namely `sum`, `count` and `average`.

A reference to `NoisySelect` is rewritten as follows if the user privacy budget is not exhausted. We decompose the table T into two parts — the authorized subset and the unauthorized subset. An accurate aggregate is computed over the authorized subset and a differentially private aggregate over the unauthorized subset. The two aggregates are combined. The rewriting for the case where there is only one aggregation and the `scalarAgg` is `count` is shown below in relational algebra-like notation.

$$count(\sigma_{auth(T)}(T)) + noisyCount_{\epsilon}(\sigma_{\neg auth(T)}(T))$$

By Theorems 2 and 4, we can see that the above rewriting guarantees ϵ-differential privacy relative to the authorization policy.

5.3.2 Incorporating Joins

The main challenge in combining joins with differentially private aggregations is that joins have a large sensitivity (recall from Section 4.2.1 that the sensitivity is the maximum influence any single record can have on the output of the aggregation.) Even for the special case of key-foreign key joins the sensitivity can be large since deleting a record from the "key side" can have an unbounded effect on the output join cardinality. Therefore, PINQ essentially only supports key-key joins [8].

In our system, since our goal is to guarantee differential privacy relative to the authorization policy, we *can* support foreign key joins as follows. We introduce the notion of a *stable* transformation relative to an authorization policy.

Definition 4. A transformation T is *c-stable* relative to authorization policy $\mathcal{P}$ if for any two data sets A and B such that $\mathcal{P}(A) = \mathcal{P}(B)$,

$$|T(A) \oplus T(B)| \leq c \times |A \oplus B|$$

Example 2. If we have two tables R and S with R having a foreign key referencing S, then the join $R \bowtie \sigma_{auth(S)}S$ is 1-stable.

If we perform stable transformations relative to an authorization policy before a differentially private aggregation, then the overall computation is differentially private relative to the policy.

THEOREM 5. *Fix an authorization policy $\mathcal{P}$. Let M provide ϵ-differential privacy and let T be an arbitrary c-stable transformation relative to $\mathcal{P}$. The composite computation $M \circ T$ provides $\epsilon \times c$-differential privacy relative to $\mathcal{P}$.*

PROOF. Fix data instances A and B with $\mathcal{P}(A) = \mathcal{P}(B)$. We have:

$$\begin{aligned} & \mathbf{Pr}[M(T(A)) \in S] \\ \leq\ & \mathbf{Pr}[M(T(B)) \in S] \times \exp(\epsilon \times |T(A) \oplus T(B)|) \\ \leq\ & \mathbf{Pr}[M(T(B)) \in S] \times \exp(\epsilon \times c \times |A \oplus B|) \end{aligned}$$

□

We use Theorem 5 to extend the class of DPViews to support foreign key joins as follows.

```
create noisy view NoisyJoin(agg1,...,aggn) as
(select scalarAgg(A1), ..., scalarAgg(An)
from R, S1, ..., Sk
where pJoin and pSelect)
```

In the above expression, the predicate `pJoin` captures the join predicate and `pSelect`, additional selection predicates. We require that each S_i is joined via a foreign key lookup from one of $R, S_1, \ldots, S_{i-1}$ (the intuition behind the requirement is illustrated in Example 2).

We illustrate how the reference to `NoisyJoin` is rewritten when the user budget is not exhausted. We always enforce the authorization on each of the S_i. We decompose the result of $\sigma_{\text{pSelect}}(R \bowtie \sigma_{auth(S_1)}(S_1) \ldots \bowtie \sigma_{auth(S_k)}(S_k))$ into two parts. The *authorized join result* is the subset $\sigma_{\text{pSelect}}(\sigma_{auth(R)}(R) \bowtie \sigma_{auth(S_1)}(S_1) \ldots \bowtie \sigma_{auth(S_k)}(S_k))$ and the rest, i.e. $\sigma_{\text{pSelect}}(\sigma_{\neg auth(R)}(R) \bowtie \sigma_{auth(S_1)}(S_1) \ldots \bowtie \sigma_{auth(S_k)}(S_k))$ is the *unauthorized join result.* We decompose the join into the authorized subset and the unauthorized subset. An accurate aggregate is computed over the authorized subset and a differentially private aggregate over the unauthorized subset. The two aggregates are combined. The rewriting for the case where there is only scalar aggregate namely `count` and no predicates `pSelect` is shown below.

$$\begin{aligned} & count(\sigma_{auth(R)}(R) \bowtie \sigma_{auth(S_1)}(S_1) \ldots \bowtie \sigma_{auth(S_k)}(S_k)) + \\ & noisyCount_{\epsilon}(\sigma_{\neg auth(R)}(R) \bowtie \sigma_{auth(S_1)}(S_1) \ldots \bowtie \sigma_{auth(S_k)}(S_k)) \end{aligned}$$

Again, it is not hard to see that the above rewriting guarantees ϵ-differential privacy relative to the authorization policy.

5.3.3 Incorporating Group By

Suppose that we wish to also incorporate grouping into the class of queries that can define a DPView. Specifically we consider the following expression that extends the DPView `NoisyJoin` above with grouping columns g:

```
create noisy view NoisyGb(g,agg1,...,aggn) as
(select g, scalarAgg(A1), ..., scalarAgg(An)
from R, S1, ..., Sk
where pJoin and pSelect
group by g)
```

Intuitively, we can think of supporting group by by taking each distinct value (group) in the grouping columns and running `NoisyJoin` with additional predicates to select the given group. The first thing to note about this strategy is that a user may not be authorized to see all the groups. So we modify the strategy to only consider *authorized* groups which are the distinct values in the grouping columns in the authorized join result. Second, since the strategy invokes `NoisyJoin` in succession, the privacy guarantee we get is based on Theorem 3. However, since the groups are disjoint we can do better as we show below.

THEOREM 6. *Fix an authorization policy $\mathcal{P}$. Let M_i each provide ϵ-differential privacy relative to $\mathcal{P}$. Let p_i be arbitrary disjoint predicates over the input domain. The sequence of $M_i(\sigma_{p_i}(D))$ provides ϵ-differential privacy relative to $\mathcal{P}$.*

PROOF. The proof is almost identical to the analogous previously known result [8] — we only have the additional restriction of focusing on instances A and B with $\mathcal{P}(A) = \mathcal{P}(B)$. $\square$

We now describe how a reference to NoisyGb is rewritten (when the budget is not exhausted.) The rewriting logically invokes NoisyJoin for each authorized group with additional predicates to select the group. However, Theorems 5 and 6 are used to decrement the user's budget only once.

Note that our system is designed such that an unbounded number of queries can be run. So long as the budget permits, the rewriting invokes the unauthorized subset. But after the budget is exhausted, we fall back to basic access control mechanisms. Further, we also note that our semantics can be achieved without any changes to the query execution engine merely by rewriting the reference to the DPView suitably.

5.4 Integrating Parameters

We now sketch one possible manner in which the privacy budget and noise parameters can be integrated into our system. The privacy budget for DBMS users is set as part of the policy specification and managed as a part of the user's metadata. The noise parameter is passed with each query as a connection property. For application users, both the privacy budget and the noise parameter reside in the user application context [11].

5.5 Illustrative Example

We consider a simplified sales database extending the Employee-Department database we have used earlier in the paper. The sales database has the following tables — Sales(ProductID, EmployeeID, CustomerID, SalesAmount), Employee(EmployeeID, ManagerID), Product(ProductID, Category), Customer(CustomerID, RegionID) and Region(RegionID, NationID).

The authorization policy lets managers access all records in the Customer, Product and Region tables (note that the Customer table in our example does not store sensitive customer information) and the records of employees that are direct reports and their corresponding sales records. Under the above authorization policy, a manager can find the total sales undertaken by each direct report per product through the following query:

```
select E.EmployeeID, S.ProductID, sum(SalesAmount)
from Employee E, Sales S
where E.EmployeeID = S.EmployeeID
group by E.EmployeeID, S.ProductID
```

We note that the above query is rewritten first before execution to only reference the rows the manager is authorized to see. Using the same query above, depending on which manager logs in, we get different rewritten queries (which is the point of supporting authorizations within the database server). In this sub-section, we describe queries as issued by the application noting that they would be rewritten by the system before execution.

Suppose that in order to give a better sense of an employee's sales we wish to compare them with the total per-product sales. The above authorization policy does not grant access to the total per-product sales. We can expose the total per-product sales using the following noisy view.

```
create noisy view NoisyPerProductSales as
 select S.ProductID, sum(SalesAmount) as TotalSales
 from Sales S
 group by S.ProductID
```

Each employee's per-product sales can be compared with the total per-product sales by issuing the following query:

```
select E.EmployeeID, S.ProductID, sum(SalesAmount),
       min(NV.TotalSales)
from Employee E, Sales S, NoisyPerProductSales NV
where E.EmployeeID = S.EmployeeID and
      S.ProductID = NV.ProductID
group by E.EmployeeID, S.ProductID
```

Again, just as with the authorization predicates, NoisyPerProductSales is rewritten differently depending on which manager logs in. Thus the output of the query also changes depending on the current user.

Now we consider a different analysis task where a manager wishes to analyze the sales in her department grouped by product category and by nation. Issuing the following query accomplishes this task.

```
select R.NationID, P.Category, sum(SalesAmount)
from Product P, Sales S, Customer C, Region R
where P.ProductID = S.ProductID
  and S.CustomerID = C.CustomerID
  and C.RegionID = R.RegionID
group by R.NationID, P.Category
```

As in the previous case above, in order to compare the sales from a given department with the overall sales while at the same time preserving privacy, we can define the following noisy view.

```
create noisy view NoisyPerProductPerRegionSales as
 select P.Category, R.NationID,
        sum(SalesAmount) as TotalSales
 from Sales S, Product P, Customer C, Region R
 where S.CustomerID = C.CustomerID
   and C.RegionID = R.RegionID
   and S.ProductID = P.ProductID
 group by P.Category, R.NationID
```

The sales within the department and across all departments can be compared by issuing the following query.

```
with DeptSales(NationID,Category,TotalSales) as
(
   select R.NationID, P.Category, sum(SalesAmount)
   from Product P, Sales S, Customer C, Region R
   where P.ProductID = S.ProductID
     and S.CustomerID = C.CustomerID
     and C.RegionID = R.RegionID
   group by R.NationID, P.Category
)
select N.Category, N.NationID,
       N.TotalSales, D.TotalSales
from DeptSales D,
     NoisyPerProductPerRegionSales N
where D.Category = N.Category
  and D.NationID = N.NationID
```

5.6 Summary

The hybrid architecture we sketched above satisfies the noisy view properties outlined in Section 3. We can reduce to the functionality of PINQ using DPViews. On the other hand, if have only authorization predicates, we reduce to standard access control mechanisms. Further, the examples in Section 5.5 illustrate that we can combine access to authorized and unauthorized portions of data in sophisticated ways, joining them and aggregating them. This is only made possible by integrating the mechanisms of access control and differential privacy (specifically PINQ). In this way, we obtain value beyond the sum of the individual parts. We have also shown that our architecture, while seemingly ad-hoc is in fact a principled approach to integrate differential privacy primitives while preserving its privacy guarantees.

However, we also inherit the baggage associated with existing differential privacy implementations. Therefore, the utility of our framework for complex queries over real datasets remains to be studied. We view our implementation only as a first step in opening up a debate in our community on how database systems can provide meaningful support to address privacy concerns.

6. RELATED WORK

There has been considerable amount of previous work in data privacy [3] and access control [2, 6, 11, 13]. However, to the best of our knowledge, ours is the first paper that studies how access control primitives and privacy preserving mechanisms can be integrated within a database system in a principled manner. The previous work that is most closely related is the Airavat system [14] that combines access control primitives with differential privacy. However, Airavat focuses on the cloud setting where the execution engine is MapReduce. Further, the model of access control considered is mandatory access control rather than discretionary access control supported by the SQL standard and addressed in this paper. Finally, in contrast with Airavat, we formally analyze the privacy implications of combining access control with differential privacy.

Other related work includes techniques for access control over probabilistic data [12] where since the data is probabilistic, the system may not be able to decide whether or not a user has access to a tuple. This is addressed by returning a perturbed tuple — the noise added is such that when it is certain that the user has access, the noise added is 0 and when it is certain that the user does not have access, the value returned is random.

Recent work [7] has addressed setting privacy policies for releasing information about search logs. The privacy policy is implemented using PINQ for differential privacy by setting different privacy budgets for different users. This work is complementary to what we study in this paper.

7. CONCLUSIONS

Data privacy issues are increasingly becoming important for database applications. However the current "black and white" world of access control primitives supported by database systems is clearly inadequate for supporting data privacy. In this paper, we sketch an architecture for a hybrid system that enhances an authorization policy with the abstraction of noisy views that encapsulate previously proposed privacy mechanisms. Accessing data through a set of views is natural for users of database systems and thus the noisy views abstraction represents a natural progression of the concept of authorization views.

We also discuss how we can implement noisy views based on differentially private algorithms. A key advantage of the proposed hybrid system is its flexibility. It can support queries that refer to both the base tables and the differentially private views thus resulting in a system that is more powerful than using access control techniques or differential privacy techniques in isolation. While combining authorizations and differentially private views in this manner seems ad-hoc, we show that it is a principled way to integrate differential privacy primitives with privacy guarantees.

However, our system also inherits some of the limitations of state of the art differential privacy. Therefore, the utility of our framework for complex queries over real datasets remains to be studied. On the whole, we think of our paper as a first step in initiating discussions on how database systems can provide meaningful support to address privacy concerns.

8. ACKNOWLEDGMENTS

We are very grateful to Arvind Arasu for his many thoughtful and insightful comments that have greatly influenced this paper. We also thank the anonymous referees for their valuable feedback.

9. REFERENCES

[1] R. Agrawal, R. Srikant, and D. Thomas. Privacy preserving olap. In *SIGMOD*, 2005.

[2] S. Chaudhuri, T. Dutta, and S. Sudarshan. Fine grained authorization through predicated grants. In *ICDE*, 2007.

[3] B.-C. Chen, D. Kifer, K. LeFevre, and A. Machanavajjhala. Privacy-preserving data publishing. *Foundations and Trends in Databases*, 2(1-2), 2009.

[4] I. Dinur and K. Nissim. Revealing information while preserving privacy. In *PODS*, 2003.

[5] C. Dwork. Differential privacy. In *ICALP (2)*, 2006.

[6] G. Kabra, R. Ramamurthy, and S. Sudarshan. Redundancy and information leakage in fine-grained access control. In *SIGMOD*, 2006.

[7] P. B. Kodeswaran and E. Viegas. Applying differential privacy to search queries in a policy based interactive framework. In *CIKM-PAVLAD*, 2009.

[8] F. McSherry. Privacy integrated queries: an extensible platform for privacy-preserving data analysis. In *SIGMOD*, 2009.

[9] A. Narayanan and V. Shmatikov. Myths and fallacies of "pii". *Communications of the ACM*, 2010.

[10] A. Netz, S. Chaudhuri, J. Bernhardt, and U. M. Fayyad. Integration of data mining with database technology. In *VLDB*, 2000.

[11] Oracle Corporation. Oracle virtual private database. http://www.oracle.com/technology/deploy/security /database-security/virtual-private-database/index.html.

[12] V. Rastogi, D. Suciu, and E. Welbourne. Access control over uncertain data. *PVLDB*, 1(1), 2008.

[13] S. Rizvi, A. O. Mendelzon, S. Sudarshan, and P. Roy. Extending query rewriting techniques for fine-grained access control. In *SIGMOD*, 2004.

[14] I. Roy, S. Setty, A. Kilzer, V. Shmatikov, and E. Witchel. Airavat: Security and Privacy for MapReduce. In *NSDI*, 2010.