# Detecting Privacy Violations in Database Publishing using Disjoint Queries

[Extended Abstract]

Millist W. Vincent
School of Computer and
Information Science
University of South Australia
Adelaide, Australia
millist.vincent@unisa.edu.au

Mukesh Mohania
IBM India Research
Laboratory
New Delhi, India
mkmukesh@in.ibm.com

Mizuho Iwaihara
Department of Social
Informatics
Kyoto University
Kyoto, Japan
iwaihara@i.kyoto-u.ac.jp

## ABSTRACT

We present a new method of detecting privacy violations in the context of database publishing. Our method defines a published view $V$ to preserve the privacy of a secret query $Q$ if $V$ and $Q$ return no tuples in common, over all possible database instances. We then establish necessary and sufficient conditions that characterize when $V$ preserves the privacy of $Q$ in terms of the projected inequalities in the queries, both for conjunctive queries and queries with negation. We also show that integrity constraints have an effect on privacy, and derive a test for ensuring privacy preservation in the presence of FD constraints. The issue of privacy preservation in the presence of multiple views is investigated, and we show that it can reduced to the single view case for a suitably chosen view.

## 1. INTRODUCTION

With the widespread use of digital technology and the vast amount of information now stored on the Web and in other repositories, the possibility of disclosure of sensitive information has increased greatly in recent years [14]. However developing techniques to prevent the inappropriate disclosure of sensitive information is a complex one, since often the need to protect sensitive information in a database has to be balanced against the need to also have non-sensitive data available for users, often referred to as the *utility* of the data [18, 26]. Such situations occur frequently in P2P and other peer-based settings, where different autonomous sources interact and share data [25, 17]. One specific such situation is *database publishing* [22, 13], whereby the owner of a database makes publicly available views of the database for use by peers, while at the same time trying to ensure that sensitive information is not leaked.

The focus of this article is on the problem of *data privacy* in database publishing, which we now outline. The owner of a database $I$ wishes to publish a view $V(I)$, or more generally a set of views $\{V_1(I), \ldots, V_n(I)\}$. However, the owner also wishes to ensure that publication of the views does not compromise the privacy of some secret data in the database, which we assume is specified by a query $Q(I)$. We note that this setting differs from, and is orthogonal to, the traditional database security problem of preventing unauthorized access to the database, for which access control is the most common solution [4]. However, in data publishing the views are made public (and so can be combined), and thus the problem is how to ensure that users cannnot use the published views to gain information about $Q(I)$. Because of these differences, access control mechanisms are not a solution to the issue of privacy protection in database publishing, and thus new methods must be developed. However, in the context of database publishing where the secret data is specified at the logical level by a query $Q$, rather than at the physical level by an attribute as in access control, it is not immediately apparent how to determine whether leakage about $Q$ has occurred and, if so, to quantify how much.

The main approach to determining if leakage of private data has occurred in database publishing is the method of *perfect-security* [23, 24], which is based on a probabilistic model of data [6]. In this approach one compares the prior knowledge of an adversary about $Q(I)$, with the knowledge of the adversary about $Q(I)$ after $V(I)$ has been published. The knowledge of the adversary is modelled by assuming that they know a probability distribution $\mathbf{P}$ for the random instance $I$, and the definition of $Q$, and hence can compute the prior probability $\mathbf{P}(Q)$. Upon publication of $V(I)$, the adversary can revise their guess of $Q(I)$ using this new knowledge by computing $\mathbf{P}(Q \mid V)$. In the case that $\mathbf{P}(Q) = \mathbf{P}(Q \mid V)$, then the publication of $V(I)$ has not improved the adversary's chance of guessing $Q(I)$ and so no leakage of information is considered to have occurred. Clearly this test depends on the specific probability distribution $\mathbf{P}$ that the adversary is assumed to know, and so perfect-security is defined to occur if $\mathbf{P}(Q) = \mathbf{P}(Q \mid V)$, for all possible $\mathbf{P}$. Although this condition still involves probability distributions, one of the main results in [23, 24] was to show that it can be converted from a test involving probability distributions to a logical test involving the existence of certain common tuples in the database.

While the notion of perfect-security represents a fundamental advance in the understanding of data privacy, it has

*EDBT 2009*, March 24–26, 2009, Saint Petersburg, Russia.

some practical limitations in the context of database publishing, as acknowledged by the originators of the approach [6]. The first limitation is that the notion of perfect-security can be too strict - it classifies some cases as security violations that are acceptable in practice. We also note that in database publishing the database owner has to balance the need to protect private data against the need to publish useful data, and so a privacy protection technique that is too strict will limit the utility of the published data. The second limitation is that checking whether a secret query $Q$ is perfectly-secure w.r.t. $V$ is computationlly intractable, even for conjunctive queries where the problem has been shown to be $\Pi_2^p$-complete [23, 24]. Thirdly, collusion cannot occur in the perfect-security approach since one of the fundamental results in [23, 24] was to show that if $Q$ is perfectly-secure w.r.t. a set of views taken individually, then $Q$ is perfectly secure w.r.t. the set as a whole. The orginators of the perfect-security approach consider this to be a weakness [6], since in practice collusion can and does occur.

In this article we present a new method of detecting privacy violations in database publishing, with the aim of overcoming the limitations just mentioned of the perfect-security approach and thus provide an alternative in applications where a less stringent, but computationally tractable, method of detecting privacy violations is required. Other proposals for relaxing the notion of perfect-security [8, 7] are based on a probabilistic model, and hence may require knowledge of distributions for data. Our method is based on the intuitive notion that if a published view $V$ is not to leak information about a secret query $Q$, then $Q$ and $V$ should not return any tuples in common, that is $Q(I)$ and $V(I)$ should be *disjoint*, a notion first introduced in [11] in another context. In general our notion of privacy preservation is a weaker notion than perfect-security, yet seems to capture the intuitive notion of privacy in many applications. Moreover, we show that the problem of testing whether $V$ preserves the privacy of $Q$ can be done in polynomial time (for conjunctive queries), and that collusion can occur in our model. Another feature of our approach is that it is sensitive to changes in the secret query $Q$ that leave the query body unchanged but alter the schema. This allows the database owner to adjust the level of privacy until the correct balance between privacy and utility has been achieved.

We note also that the topic of disjoint queries investigated in this article has application in another area, apart from that of privacy preservation, namely that of *irrelevant updates* [15, 16]. Given a view $V$ over a database instance $I$, an irrelevant update to $I$ is one that does not change the result of $V$. This topic was investigated by several works in the 1990's because of its importance in view maintenance [12, 19], and has also found application more recently in query processing in probabilistic databases [3]. The application of disjoint queries to irrelevant updates is as follows. Suppose that we update a set of tuples in the database defined by $Q(I)$. If we denote the tuples in $I$ that are involved in a derivation of a tuple in $V(I)$ by $V^{-1}(I)$, then the update will be irrelevant if $Q(I)$ and $V^{-1}(I)$ are disjoint.

We now summarize the main contributions of this article.

- We define data privacy based on requiring that the tuples in a published view $V$ be disjoint from the secret query $Q$.

- For the case of conjunctive queries, we prove a necessary and sufficient result for $Q$ and $V$ to be disjoint based on the projections of the inequalities in the queries. We then show that testing whether $V$ preserves the privacy of $Q$ can be done in polynomial time in the size of $Q$ and $V$.

- We extend the previous result to the case of queries involving negation.

- We investigate the effect of integrity constraints in the database and show that they affect whether or not the privacy of $Q$ is preserved. We then establish a result which gives a necessary condition for $V$ to preserve the privacy of $Q$, when the queries are conjunctive and the database satisfies a set of FD constraints.

- We investigate the issue of privacy preservation in the presence of multiple published views, and we show that collusion can occur in our model. We then show that the problem of determining if a set of views preserves the privacy of $Q$ can be solved by reducing it to the problem of testing whether a single view, defined as a join query over all the published views, preserves the privacy of $Q$.

- We do a detailed comparison of our notion of privacy preservation with perfect-security and prove several results which characterizes when the two notions coincide, and when they differ.

The rest of this article is organized as follows. In Section 2 we present the main features of our approach for detecting privacy violations, and in Section 3 we present some basic definitions and notations. In Section 4 we establish necessary and sufficient conditions for $V$ to be privacy preserving, both for conjunctive queries and also for queries involving negation. We also show in Section 4 that for conjunctive queries, testing whether $V$ preserves the privacy of $Q$ can be done in polynomial time in the size of $Q$ and $V$. In Section 5 we examine the effect of FD constraints on privacy and derive a sufficient condition for $V$ to preserve the privacy of $Q$ when the database satisfies a set of FD constraints. In Section 6, we investigate privacy in the presence of multiple published views, and show that the problem of determining if a set of queries preserves the privacy of $Q$ can be reduced to determining if a single suitably chosen view preserves the privacy of $Q$. Section 7 contains a detailed comparison of privacy preservation versus perfect-security, and we survey related work in Section 8. Finally, Section 9 contains some concluding comments.

## 2. PRIVACY AND DISJOINT QUERIES

Our approach to determining whether a view $V$ preserves the privacy of a secret query $Q$ is based on the following notion of *disjoint queries*, a topic first studied in [12], and assuming for the moment that $Q$ and $V$ have the same schema.

DEFINITION 1. *If $I$ is an instance of a database schema $D$ and $Q$ and $V$ are queries that have the same schema, then $Q$ and $V$ are defined to be* disjoint *if for every $I, Q(I) \cap V(I) = \emptyset$.*

Our motivation for defining privacy based on disjoint queries comes from the view-based access control used in database-security [4]. In this method, users can only access the database through a view(s) and a user query is considered legal only if it returns a subset of the view for all possible database states. Our approach is the inverse of the view-based approach since rather than specifying the information that can be accessed, we specify the information that is private by a query $Q$, and then the notion of a user query being legal translates to the requirement in our approach that the user query must be disjoint from $Q$ for all possible database states.

We now illustrate our definition by an example.

EXAMPLE 1. *Let $D = \{R_1(C\#, N), R_2(C\#, A, B)\}$ be a database schema where: $C\#$ represents the id of a customer, $N$ represents the name of a customer, $A$ represents an account number and $B$ represents an account balance. Suppose also that we have a secret query $Q$ and published queries $V_1$ and $V_2$ defined by the following conjunctive queries:*

$$Q(N,B) \leftarrow R_1(C\#, N), R_2(C\#, A, B), B > 10000$$
$$V_1(N,B) \leftarrow R_1(C\#, N), R_2(C\#, A, B), A = \text{‘}A101\text{’}$$
$$V_2(N,B) \leftarrow R_1(C\#, N), R_2(C\#, A, B), B < 10000$$

*The query $V_1$ is not disjoint from $Q$ since $Q(I)$ and $V_1(I)$ overlap for some database instances, such as $I = \{R_1\{\langle' C100', 'Bill' \rangle\ \}, R_2\{\langle 'C100',' A101', 1500\rangle\}\}$. However, $V_2$ and $Q$ are disjoint because the conflict between the selection conditions in the two queries ensures that the returned tuples will always be disjoint.*

We observe that in our approach, the intersection is at the tuple level, not at the attribute level. For instance, in the previous example it is possible for $Q$ and $V_2$ to return a common value for $N$, although the queries are still defined to be disjoint. This is because we regard the basic semantic unit of information to be a tuple, not an attribute value, and so we define a privacy violation as occurring only when the same tuple is returned by the secret query and the user query, for some database instance.

We now consider the situation where the schema of secret query and the user query differ. There are two possible cases.

The first case is where the schema of $Q$ contains an attribute that is not in $V$. Based on our assumption that the semantic unit of information is a tuple, we do not regard this situation as a privacy violation. So, for instance, if in Example 1 we let $Q$ remain unchanged but change the schema of $V_1$ to $V_1(N, A)$, then the new $V_1$ preserves the privacy of $Q$.

The other case is where the schema of $Q$ is strictly contained in the schema of $V$. In this case, since a tuple returned by $Q$ can be the projection of a tuple returned by $V$, we consider that a privacy violation can occur in this situation. For instance, if in Example 1 the schema of $V_1$ was extended to $V_1(C\#, N, B)$, then we would still consider this as a privacy violation because the projection of a tuple returned by the new $V_1$ could be equal to a tuple returned by $Q$.

We note that with our definition of privacy, whether a privacy violation occurs is sensitive to the schema of the secret query and so care is needed in defining the secret query schema to ensure that the correct semantics are obtained. For instance, the secret query $Q$ in Example 1 specifies that the name and balance of a customer are to be kept secret. This means that any user query with a schema containing only name or balance alone would not violate the privacy of $Q$. However, if the schema of $Q$ was changed to $Q(N)$, then this implies that every user query that had a schema with $N$ would violate the privacy of $Q$ since there are no restrictions placed on $N$ in the body of $Q$. So, in Example 1 for instance, under the new definition of $Q$ the user query $V_2$ now violates the privacy of $Q$. In general, reducing the schema of $Q$, while keeping the body of $Q$ unchanged, results in a stricter notion of privacy preservation. This point will be explored in more detail in a later section.

## 3. BASIC DEFINITIONS

In this section we formalize our notion of privacy preservation discussed in the previous section.

We assume a relational setting. A *database schema* $D$ consists of a finite set of *relational schemas* $\{R_1, \ldots, R_n\}$, where each relational schema consists of a unique name and a finite set of *attributes*. The set of attributes in a relational schema $R_i$ is denoted by $sch(R_i)$, and without loss of generality we assume that each attribute appears in at most one relational schema. We also assume that every attribute is defined over $\mathbb{Z}$, the set of positive and negative integers. A *tuple* for a relational schema $R_i$, where $sch(R_i) = \{X_1, \ldots, X_k\}$, is an element of the set $\mathbb{Z} \times \ldots \times \mathbb{Z}$ ($k$ times). A *relation* $r$ defined over relational schema $R_i$, denoted by $r(R_i)$ or simply by $r$ if $R_i$ is understood, consists of a finite set of tuples defined over $R_i$. A *database instance* defined over a schema $D = \{R_1, \ldots, R_n\}$, denoted by $I(D)$, or simply by $I$ if $D$ is understood, is a finite set of relations $\{r_1(R_1), \ldots, r_n(R_n)\}$.

The queries we consider are nonrecursive DATALOG queries with inequalities, and possibly with negation, defined as follows. We first note that while we define queries using the well known rule-based approach [1], our presentation differs slightly from the usual formulation since the variables in a query have a type, which is the corresponding attribute of the relational schema in which it appears. This is done because we use both the schema of the secret query, and its contents, in determining privacy preservation, and so we need to know the attribute in the schema that corresponds to each variable in the schema of the query.

To simplify the presentation, we assume that the head of the query contains only variables, though this is not essential and our results can be easily extended to the case where constants appear in the head of the query.

DEFINITION 2. *We assume a set of variable names $\Omega$, and the function typ which is a mapping from $\Omega$ to the set $sch(R_1) \cup \ldots \cup sch(R_n)$, where $D = \{R_1, \ldots, R_n\}$. A query $Q$ is defined by:*

$$Q(\bar{X}) \leftarrow t_1, \ldots, t_n, C_{n+1}, \ldots, C_{n+m}$$

*where $t_1, \ldots, t_n$ are* terms, *$C_{n+1}, \ldots, C_{n+m}$ are* inequalities, *and $\bar{X} \subseteq \Omega$. A term is either of the form:*

1. *$R(X_1, \ldots, X_n)$, where $R$ is a relational schema in $D$ and $\{X_1, \ldots, X_n\} \subseteq \Omega$. This is called a* positive term*; or*

2. *$\neg R(X_1, \ldots, X_n)$, where $R$ is a relational schema in $D$ and $\{X_1, \ldots, X_n\} \subseteq \Omega$. This is called a* negative term*.*

*An inequality is either of the form:*

1. *$X$ **op** $a$, where $X \in \Omega$, $a \in \mathbb{Z}$, **op** $\in \{<, \leq, =, >, \geq, \}$; or*

2. *$X$ **op** $Y$, where $\{X, Y\} \subseteq \Omega$, **op** $\in \{<, \leq, =, >, \geq, \}$.*

$Q(\bar{X})$ *is called the* head *of the query* $Q$*, and* $t_1, \ldots, t_n, C_{n+1}, \ldots, C_{n+m}$ *is called the* body *of* $Q$*.*
$\bar{X}$ *is called the* schema *of* $Q$ *and is also denoted by* $sch(Q)$*.*
*We also place the following restrictions on queries:*
*(i) We assume that a variable can appear at most once in the body of the query, and at most once in the head of a query;*
*(ii) If a variable* $X_i$ *appears as the ith variable in a term* $R(\_, \ldots, X_i, \ldots)$ *or* $\neg R(\_, \ldots, X_i, \ldots)$*, then* $typ(X_i) = A_i$*, where* $A_i$ *is the ith attribute of* $R$*;*
*(iii)* $\bar{X} \neq \emptyset$*, i.e. Boolean queries are not allowed.*

We note that condition (i) above does not prevent join queries from being defined in our framework, since a join query can be modelled by including inequalities in the query which force the join variables to be equal. Similarly, allowing constants as well as variables in the terms of a query can also be modelled using inequalities. Finally, we note that self-joins are allowed in queries but Boolean queries are not. This last restriction is to simplify the presentation, and our approach can easily be extended to Boolean queries.

Since the inequality $X < a$ is equivalent to $X \leq a - 1$, and $X > a$ is equivalent to $X \geq a + 1$, we assume for the rest of the article that $\mathbf{op} \in \{\leq, \geq,\}$ for any inequality of the form $(X\ \mathbf{op}\ a)$. Also, because of symmetry, we assume that $\mathbf{op} \in \{<, \leq,\}$ for any inequality of the form $X\ \mathbf{op}\ Y$. We also note that the equality operator is redundant since it can be modelled as two inequalities, but we include it so as to simplify the presentation of some of the examples.

If all terms in $Q$ are positive terms, then $Q$ is called a *conjunctive query with inequalities* and will be denoted by DATALOG. Queries with at least one negative term will be denoted by DATALOG$^{\neg}$. We also assume that the queries are range restricted, i.e. every variable $X$ in $\bar{X}$ also appears in some positive term in the body of $Q$, or there exists a variable $Y$ such that $C_{n+1}, \ldots, C_{n+m}$ imply that $X = Y$ and $Y$ appears in some positive term in the body of $Q$. We now define query evaluation in the standard fashion.

Definition 3. *Let* $Q(X_1, \ldots, X_p)$ *be a query, let* $vars(Q)$ *denote the set of variables which appear in the body of* $Q$*, and let* $I$ *be a database instance defined over* $D$*.*
*A* valuation $\rho$ *for* $Q$ *is a function from* $vars(Q)$ *to* $\mathbb{Z}$*, and the evaluation* $\rho(Q, I)$*, where* $I = \{r_1, \ldots, r_n\}$ *is defined as follows:*

1. *A term* $R_i(X_1, \ldots, X_n)$ *evaluates to true if the tuple* $\langle \rho(X_1), \ldots, \rho(X_n) \rangle \in r_i$*;*
2. *A term* $\neg R_i(X_1, \ldots, X_n)$ *evaluates to true if the tuple* $\langle \rho(X_1), \ldots, \rho(X_n) \rangle \notin r_i$*;*
3. *An inequality* $C = X\ \mathbf{op}\ a$ *evaluates to true if* $\rho(X)\ \mathbf{op}\ a$ *is true;*
4. *An inequality* $C = X\ \mathbf{op}\ Y$ *evaluates to true if* $\rho(X)\ \mathbf{op}\ \rho(Y)$ *is true;*
5. $\rho(Q, I)$ *evaluates to true if every term and every inequality in the body of* $Q$ *evaluates to true.*

The result of the query $Q$ on on the database instance $I$ is defined by:

$$Q(I) = \{\langle x_1, \ldots, x_p \rangle \mid \text{exists a valuation } \rho \text{ such that } \rho(X_1) = x_1, \ldots, \rho(X_p) = x_p \text{ and } \rho(Q, I) \text{ evaluates to true}\}$$

We also need to compare the schemas of queries based on the types of the variables in their schemas in defining privacy, and this is done as follows.

Definition 4. *Given queries* $Q(X_1, \ldots, X_n)$ *and* $V(X'_1, \ldots, X'_m)$*,* $sch(Q)$ *is contained in* $sch(V)$*, denoted by* $sch(Q) \preceq sch(V)$*, if* $n \leq m$ *and* $typ(X_i) = typ(X'_i)$ *for all* $i \in \{1, \ldots, n\}$*.*
*The schemas* $sch(Q)$ *and* $sch(V)$ *are equivalent, denoted by* $sch(Q) \equiv sch(V)$*, if* $sch(Q) \preceq sch(V)$ *and* $sch(V) \preceq sch(Q)$*.*
*The difference between* $sch(V)$ *and* $sch(Q)$*, denoted by* $sch(V) \ominus sch(Q) = \{X'_i \mid \nexists X_j(typ(X'_i) = typ(X_j))\}$*.*

Based on the discussion in the previous section, we now define privacy preservation as follows.

Definition 5. *A query* $V$ *is* privacy preserving *with respect to a secret query* $Q$ *if either:*
*(i)* $sch(Q) \ominus sch(V) \neq \emptyset$*; or*
*(ii)* $sch(Q) \preceq sch(V)$ *and* $Q$ *and* $V'$ *are disjoint, where* $V'$ *is the query defined by:*

$$V'(sch(Q)) \leftarrow V$$

*(assuming, without loss of generality, that the variables in* $V$ *are renamed if necessary so that* $sch(Q) \subseteq sch(V)$*).*

Case (i) is the situation where there is a variable in the schema of $Q$ for which there is no matching variable in the schema of $V$ which, as discussed in the previous section, we do not consider to be a privacy violation. Case (ii) is where every variable in $sch(Q)$ has a matching variable in $sch(V)$, in which case we require that the projection of $V$ on $sch(Q)$ be disjoint from $Q$.

We now define satisfiability and implication of a set of inequalities.

Definition 6. *Let* $\bar{C} = \{C_1, \ldots, C_n\}$ *be a set of inequalities.* $\bar{C}$ *is defined to be* satisfiable *if there is a valuation* $\rho$ *on* $vars(C)$ *for which every* $C_i \in \bar{C}$ *evaluates to true, otherwise it is* unsatisfiable*.*
*A set of inequalities* $\bar{C}$ implies *another inequality* $C$*, denoted by* $\bar{C} \models C$*, if every valuation* $\rho$ *on* $vars(C)$ *in which every inequality in* $\bar{C}$ *evaluates to true,* $C$ *also evaluates to true.*
*The set of all inequalities implied by* $\bar{C}$ *is denoted by* $\bar{C}^+$*.*
*Two sets of inequalities,* $\bar{C}_1$ *and* $\bar{C}_2$*, are* equivalent*, denoted by* $\bar{C}_1 \equiv \bar{C}_2$*, if* $\bar{C}_1^+ = \bar{C}_2^+$*.*
*If* $\bar{X} \subseteq \Omega$ *and* $\bar{C}$ *is a set of inequalities, then we define:*

$$\pi_{\bar{X}}(\bar{C}) = \{C | C \in \bar{C}^+ \wedge vars(C) \subseteq \bar{X}\}$$

We note that it may be the case that $\pi_{\bar{X}}(\bar{C})$ is satisfiable, but $\bar{C}$ is not satisfiable. For example, if $\bar{C} = \{X \leq 10, Y > Z, Y < Z\}$, then $\bar{C}$ is not satisfiable, but $\pi_X(\bar{C}) \equiv \{X \leq 10\}$ is satisfiable. However, the converse is not true, since it follows immediately from the definition of $\pi_{\bar{X}}(\bar{C})$ that if $\bar{C}$ is satisfiable, then $\pi_{\bar{X}}(\bar{C})$ is also satisfiable.

We now define the sets of upper and lower bounds for a variable in an inequality.

Definition 7. *Let* $\bar{C}$ *be a satisfiable set of inequalities, and let* $X \in vars(\bar{C})$*. Then:*

- $LOW(X, \bar{C}) = \{a \mid X \geq a \in \bar{C}^+\}$.
- $HIGH(X, \bar{C}) = \{a \mid X \leq a \in \bar{C}^+\}$.

We now establish the important result that either $LOW$ is empty or it has an upper bound, and either $HIGH$ is empty or it has a lower bound.

Lemma 1. *If $\bar{C}$ is a satisfiable set of inequalities and $X \in vars(\bar{C})$, then:*
*(i) either $LOW(X, \bar{C}) = \emptyset$, or there exists a constant $k$ such that for every $a$ in $LOW(X, \bar{C})$, $a \leq k$;*
*(ii) either $HIGH(X, \bar{C}) = \emptyset$, or there exists a constant $k$ such that for every $a$ in $HIGH(X, \bar{C})$, $a \geq k$.*

As a consequence of this result, we can define the lower and upper bounds for a variable in a set of inequalities, denoted by *low* and *high*, as follows.

Definition 8. *Let $\bar{C}$ be a satisfiable set of inequalities and let $X \in vars(\bar{C})$.*

- *If $LOW(X, \bar{C}) = \emptyset$ then $low(X, \bar{C}) = -\infty$, otherwise $low(X, \bar{C}) = max\{a_i \mid X \geq a_i \in \bar{C}^+\}$.*
- *If $HIGH(X, \bar{C}) = \emptyset$ then $high(X, \bar{C}) = +\infty$, otherwise $high(X, \bar{C}) = min\{a_i \mid X \leq a_i \in \bar{C}^+\}$.*

For example, if $\bar{C} = \{X > Y, Y \leq 4, Y \geq 2\}$ then $low(Y, \bar{C}) = 2$, $high(Y, \bar{C}) = 4$, $low(X, \bar{C}) = 3$, $high(X, \bar{C}) = +\infty$

For every constant $k$, we also define that the inequalities $k < +\infty$ and $-\infty < k$ hold.

## 4. CHECKING PRIVACY

In this section we establish some of the main results of our article which characterize when a published query $V$ preserves the privacy of a secret query $Q$, first for the case of DATALOG queries and then for DATALOG$^\neg$ queries. The important point to note concerning these results is that privacy preservation can be determined using only the structure of $Q$ and $V$, and so can be checked at compile time.

We first establish a result which characterizing when two queries are disjoint, and to do this we introduce the notion of conflicting sets of inequalities.

Definition 9. *Two sets of inequalities $\bar{C}_Q = \{C_{n+1}, \ldots, C_{n+m}\}$ and $\bar{C}_V = \{C_{n'+1}, \ldots, C_{n'+m'}\}$ are said to* conflict *w.r.t. a set of variables $\bar{X}$ if the set of inequalities $\pi_{\bar{X}}(\bar{C}_Q) \cup \pi_{\bar{X}}(\bar{C}_V)$ is unsatisfiable.*

We now illustrate the definition by an example.

Example 2. *Let $\bar{C}_Q = \{X < Y, Y \leq 5\}$ and let $\bar{C}_V = \{X > Y, Y \geq 10\}$ and let $\bar{X} = \{X\}$. Then $\pi_{\bar{X}}(\bar{C}_Q) \equiv \{X \leq 4\}$ and $\pi_{\bar{X}}(\bar{C}_V) \equiv \{X \geq 11\}$ and so $\bar{C}_Q$ and $\bar{C}_V$ conflict. However, if we let $\bar{C}'_V = \{X > Y, Y > Z, Z \geq 0\}$, then $\bar{C}'_V \equiv \{X \geq 2\}$ and so $\bar{C}_Q$ and $\bar{C}'_V$ do not conflict.*

We now characterize when two queries are disjoint depending on whether or not their sets of inequalities projected onto the schema of the secret query conflict.

Theorem 1. *Let $Q$ be the DATALOG query*

$$Q(\bar{X}) \leftarrow t_1, \ldots, t_n, C_{n+1}, \ldots, C_{n+m}$$

*and let $V$ be the DATALOG query*

$$V(\bar{X}) \leftarrow t_{1'}, \ldots, t_{n'}, C_{n'+1}, \ldots, C_{n'+m'}$$

*and let $\bar{C}_Q = \{C_{n+1}, \ldots, C_{n+m}\}$ and $\bar{C}_V = \{C_{n'+1}, \ldots, C_{n'+m'}\}$.*
*If $\bar{C}_Q$ and $\bar{C}_V$ are separately satisfiable, then $Q$ and $V$ are disjoint iff $\bar{C}_Q$ and $\bar{C}_V$ conflict w.r.t. $\bar{X}$.*

Corollary 1. *If $Q$ and $V$ are DATALOG queries over a database scheme $D$ such that $\bar{C}_Q$ and $\bar{C}_V$ are separately satisfiable, then $V$ preserves the privacy of $Q$ if either:*
*(i) $sch(Q) \ominus sch(V) \neq \emptyset$; or*
*(ii) the query $V'$ conflicts with $Q$ w.r.t. $sch(Q)$, where $V'$ is the query defined by $V'(sch(Q)) \leftarrow V$.*

We now address the issue of developing an algorithm for testing when two queries are disjoint. We have the following result (where $|Q|$ is the number of inequalities in $Q$).

Theorem 2. *If $Q$ and $V$ are two DATALOG queries, then testing whether $Q$ and $V$ are disjoint can be done in $O(|Q|^4 + |V|^4)$ time.*

Corollary 2. *Testing whether a query $V$ preserves the privacy of a secret query $Q$ can be done in $O(|Q|^4 + |V|^4)$ time when both $Q$ and $V$ are DATALOG queries.*

In contrast, we note that the problem of testing whether $Q$ is perfectly secure w.r.t. $V$ has been shown to be intractable, even for DATALOG queries [23, 24].

Before extending Theorem 1 to the case of DATALOG$^\neg$ queries, we first extend the definition of when two queries conflict to DATALOG$^\neg$ queries (by renaming variables if necessary and without any loss of generality, we assume that $(vars(Q) - \bar{X}) \cap (vars(V) - \bar{X}) = \emptyset$).

Definition 10. *Let $Q$ be the DATALOG$^\neg$ query*

$$Q(\bar{X}) \leftarrow t_1, \ldots, t_n, C_{n+1}, \ldots, C_{n+m}$$

*and let $V$ be the DATALOG$^\neg$ query*

$$V(\bar{X}) \leftarrow t_{1'}, \ldots, t_{n'}, C_{n'+1}, \ldots, C_{n'+m'}$$

*Then $Q$ and $V$ are defined to conflict if either:*
*(i) $\bar{C}_Q$ and $\bar{C}_V$ conflict w.r.t. $\bar{X}$; or*
*(ii) for every valuation $\rho$ which returns true on $\bar{C}_Q \cup \bar{C}_V$, there exists a term $t_i = R_i(X_1, \ldots, X_k, X'_1, \ldots, X'_n)$ from $Q$ and a term $t'_j = \neg R_i(X_1, \ldots, X_k, X''_1, \ldots, X''_n)$ from $V$, or a term $t_i = \neg R_i(X_1, \ldots, X_k, X'_1, \ldots, X'_n)$ from $V$ and a term $t'_j = R_i(X_1, \ldots, X_k, X''_1, \ldots, X''_n)$ from $Q$, such that $\rho(X'_1) = \rho(X''_1), \ldots, \rho(X'_n) = \rho(X''_n)$.*

Essentially, what condition (ii) does is ensure that there cannot exist a tuple that is in both $Q(I)$ and $V(I)$, since otherwise this would mean that there was a tuple both in, and not in, relation $R_i$. We also note that condition (ii) is a weaker condition than requiring that $C_Q \cup C_V \models \{X'_1 = X''_1, \ldots, X'_n = X''_n\}$. This is illustrated in the following example.

Example 3. *Let*

$$Q(X_1) \leftarrow R_1(X_1, Y_1), R_2(X'_1, Y'_1), X_1 = X'_1, 1 \leq X_1, X_1 \leq 2, 1 \leq Y_1, Y_1 \leq X_1, X_1 \leq Y'_1, Y'_1 \leq 2$$
$$V(X_1) \leftarrow \neg R_1(X_1, Y_2), \neg R_2(X'_1, Y'_2), R_3(X''_1, Y_2, Y''_2), X_1 = X'_1 = X''_1, 1 \leq Y_2, Y_2 \leq X_1, X_1 \leq Y'_2, Y'_2 \leq 2$$

*Then $Q$ and $V$ satisfy (ii) of the definition of conflict. To see this, from the inequalities in $Q$, $\rho(X_1) = 1$ or $\rho(X_1) = 2$ for any valuation $\rho$ which returns true on $C_Q \cup C_V$. In the case that $\rho(X_1) = 1$, it follows from the inequalities $1 \leq Y_1$ and $Y_1 \leq X_1$ that $\rho(Y_1) = 1$, and from $1 \leq Y_2$ and $Y_2 \leq X_1$ that $\rho(Y_2) = 1$. Thus $\rho(Y_1) = \rho(Y_2)$ and so the terms $R_1(X_1, Y_1)$ and $\neg R_1(X_1, Y_2)$ satisfy (ii).*

*If instead $\rho(X_1) = 2$, it follows from the inequalities $X_1 \leq Y_1'$ and $Y_1' \leq 2$ that $\rho(Y_1') = 2$, and from $X_1 \leq Y_2'$ and $Y_2' \leq 2$ that $\rho(Y_2) = 2$. Thus $\rho(Y_1') = \rho(Y_2')$ and so the terms $R_2(X_1, Y_1)$ and $\neg R_2(X_1, Y_2)$ satisfy (ii).*
*However, $C_Q \cup C_V \not\models Y_1 = Y_2$ since under the valuation $\rho(X_1) = 2, \rho(Y_1) = 1, \rho(Y_1') = 2, \rho(Y_2) = 2, \rho(Y_2') = 2$, $C_Q \cup C_V$ evaluates to true, but $Y_1 = Y_2$ evaluates to false. Also, $C_Q \cup C_V \not\models Y_1' = Y_2'$ since under the valuation $\rho(X_1) = 1, \rho(X_1') = 1, \rho(Y_1') = 1, \rho(Y_2) = 1, \rho(Y_2') = 2$, $C_Q \cup C_V$ evaluates to true, but $Y_1' = Y_2'$ evaluates to false.*

This leads to the other main result of the section, which characterizes when two DATALOG$^\neg$ queries are disjoint.

THEOREM 3. *Let $Q$ be the DATALOG$^\neg$ query:*

$$Q(\bar{X}) \leftarrow t_1, \ldots, t_n, C_{n+1}, \ldots, C_{n+m}$$

*and let $V$ be the DATALOG$^\neg$ query:*

$$V(\bar{X}) \leftarrow t_{1'}, \ldots, t_{n'}, C_{n'+1}, \ldots, C_{n'+m'}$$

*and $\bar{C}_V$ and $\bar{C}_Q$ are separately satisfiable. Then $Q$ and $V$ are disjoint iff $Q$ and $V$ conflict.*

COROLLARY 3. *If $Q$ and $V$ are DATALOG$^\neg$ queries over a database scheme $D$ such that $\bar{C}_Q$ and $\bar{C}_V$ are separately satisfiable, then $V$ preserves the privacy of $Q$ if either:*
*(i) $sch(Q) \ominus sch(V) \neq \emptyset$; or*
*(ii) the query $V'$ conflicts with $Q$, where $V'$ is the query defined by $V'(sch(Q)) \leftarrow V$.*

# 5. PRIVACY UNDER CONSTRAINTS

In this section we investigate privacy in the presence of FD constraints. We show that a secret query and a user query may be disjoint in the presence of FD constraints, but not when FD constraints are absent. This is similar to what happens with perfect-security [23, 24, 8]. We now illustrate this observation by an example.

EXAMPLE 4. *Suppose we have a relational schema $R(P, I, S)$, where: $P$ is a patient, $I$ is an illness, $S$ is a specialist. Suppose also that we have a secret query $Q$ and a published query $V$ defined by:*

$$Q(P, I) \leftarrow R(P, I, S), S = \text{`Jones'}$$

*and*

$$V(P, I) \leftarrow R(P, I, S), S = \text{`Allen'}.$$

*If there are no FDs present, then $V$ does not preserve the of $Q$ since if the database contains a patient who is treated by both 'Jones' and 'Allen' for the same illness, then the results of $Q$ and $V$ will overlap. However, if $R$ satisfies the FD $P \rightarrow S$ then we claim that $Q$ and $V$ must be disjoint and so $V$ preserves the privacy of $Q$. This is because if $Q$ and $V$ returned a common tuple $\langle p, i \rangle$, then we could deduce that patient $p$ is treated by both 'Jones' and 'Allen', which contradicts the FD $P \rightarrow S$.*

We now extend the definition of disjoint queries in the obvious manner to the case where FDs are present, and then extend the results of Section 4 to cover this new case. We first make the assumption that any FD involves only attributes that belong to a single relational schema.

DEFINITION 11. *Let $Q$ and $V$ be queries defined over a database schema $D$, and let $\Sigma$ be a set of FD constraints that apply to $D$. If $I$ is an instance of $D$, and $Q$ and $V$ are queries such that $sch(Q) = sch(V)$, then $Q$ and $V$ are defined to be* disjoint *if for all database instances $I$ which satisfy $\Sigma$, $Q(I) \cap V(I) = \emptyset$.*

We then have the following result which gives a sufficient condition for two DATALOG queries to be disjoint in the presence of FD constraints.

THEOREM 4. *Let $Q$ be the DATALOG query*

$$Q(\bar{X}) \leftarrow t_1, \ldots, t_n, C_{n+1}, \ldots, C_{n+m}$$

*and let $V$ be the DATALOG query*

$$V(\bar{X}) \leftarrow t_{1'}, \ldots, t_{n'}, C_{n'+1}, \ldots, C_{n'+m'}$$

*defined over a database scheme $D$, and let $\bar{C}_Q = \{C_{n+1}, \ldots, C_{n+m}\}$ and $\bar{C}_V = \{C_{n'+1}, \ldots, C_{n'+m'}\}$.*
*If $\bar{C}_Q$ and $\bar{C}_V$ are separately satisfiable and $\bar{C}_Q$ and $\bar{C}_V$ conflict w.r.t. $\bar{X}^+$, then $Q$ and $V$ are disjoint.*

COROLLARY 4. *If $Q$ and $V$ are DATALOG queries over a database scheme $D$ such that $\bar{C}_Q$ and $\bar{C}_V$ are separately satisfiable, and $\Sigma$ is a set of FD constraints defined over $D$, then $V$ preserves the privacy of $Q$ if either:*
*(i) $sch(Q) \ominus sch(V) \neq \emptyset$; or*
*(ii) the query $V'$ conflicts with $Q$ w.r.t. $sch(Q)^+$, where $V'$ is the query defined by $V'(sch(Q)) \leftarrow V$.*

As we now demonstrate, the converse of Theorem 4 does not hold because of interaction between FDs and the inequalities.

EXAMPLE 5. *Let $D$ be the database schema $\{R_1(X, Y), R_2(Y, Z)\}$ and suppose we have a secret query $Q$ and a published query $V$ and set of FDs $\Sigma$ defined by:*

$$Q(X) \leftarrow R_1(X, Y), R_2(Y, Z), Y = 2, Z < Y$$
$$V(X) \leftarrow R_1(X, Y'), R_2(Y', Z), Y' = 2, Z > Y'$$
$$\Sigma = \{Y \rightarrow Z\}.$$

*In this case $X^+ = X$ and $\bar{C}_Q$ and $\bar{C}_V$ do not conflict w.r.t. $X$, yet we claim that $Q(I) \cap V(I) = \emptyset$ for every instance $I$ which satisfies $\Sigma$. To see this, suppose that there exists an instance $I$ and a tuple $\langle x \rangle$ that is in both $Q(I)$ and $V(I)$. Then since $\langle x \rangle \in Q(I)$, this means that there exists tuples $\langle x, y \rangle \in r_1$ and $\langle y, z \rangle \in r_2$, for some $y$ and $z$. However, because of the inequalities in $Q$, we can deduce that these two tuples are $\langle x, 2 \rangle$ and $\langle 2, z \rangle$. Using similar reasoning, we also deduce that since $\langle x \rangle \in V(I)$, then there exists a tuple $\langle x, 2 \rangle \in r_1$ and $\langle 2, z' \rangle \in r_2$. However, we also have from the definitions of $Q$ and $V$ that $z < 2$ and $z' > 2$, so the tuples $\langle 2, z \rangle$ and $\langle 2, z' \rangle$ are distinct and hence the FD $Y \rightarrow Z$ is violated in $r_2$, which is a contradiction and so $Q(I)$ and $V(I)$ must be disjoint.*

# 6. MULTIPLE VIEWS

In this section we extend the notions of privacy and disjointedness first to unions of conjunctive queries defined over the same schema, and then to an arbitrary set of conjunctive queries.

The first observation we make is that using our approach for defining privacy, collusion can occur when there are multiple published views. That is, a combination of published views can violate the privacy of $Q$ even though individually the published views do not violate the privacy of $Q$. We now illustrate this by an example.

Example 6. *Suppose we have a relational schema $R(P, A, I)$, where $P$ represents a patient, $A$ represents an address and $I$ represents an illness. Suppose we also have the following secret query $Q$, which keeps the details of patients suffering from Aids secret*

$$Q(P, A, I) \leftarrow R(P, A, I), I = \text{`}Aids\text{'}$$

*and the published views*

$$V_1(P, A) \leftarrow R(P, A, I)$$

*and*

$$V_2(P', I) \leftarrow R(P', A, I).$$

*Individually, views $V_1$ and $V_2$ preserve the privacy of $Q$ since the schemas of both $V_1$ and $V_2$ are proper subsets of the schema of $Q$. However if we combine $V_1$ and $V_2$ by the query*

$$V(P, A, I) \leftarrow V_1(P, A), V_2(P', I), P = P',$$

*then $V$ violates the privacy of $Q$ since in the case that $P$ is a key in $R$, $V(I) = Q(I)$ for every instance $I$.*

As noted previously, this situation does not occur in the perfect-security approach [23, 24].

We now define unions of DATALOG queries as follows, for the case where the schemas of every pair of DATALOG queries are equivalent. Without loss of generality, we assume that by renaming variables where necessary, the schemas of the queries are identical.

Definition 12. *If $Q_1, \ldots, Q_n$ are DATALOG queries defined over the same schema, then the union of the queries, $Q = (Q_1, \ldots, Q_n)$ on a database instance $I$ is the query defined by $Q(I) = Q_1(I) \cup \ldots \cup Q_n(I)$.*

The following result follows immediately from this definition, and characterizes when the union of a set of DATALOG queries preserves the privacy of a secret query $Q$.

Lemma 2. *If $Q$ is a DATALOG query and $\bar{V} = (V_1, \ldots, V_n)$ is a union of DATALOG queries, then $Q$ and $\bar{V}$ are disjoint iff for every $i \in \{1, \ldots, n\}$, $Q$ and $V_i$ are disjoint.*

Corollary 5. *If $Q$ is a DATALOG query and $\bar{V} = (V_1, \ldots, V_n)$ is a set of DATALOG queries defined over the same schema such that the set of inequalities of $Q$, $\bar{C}_Q$, and the set of inequalities of every query $V_i$ in $\bar{V}$, $\bar{C}_{V_i}$, are satisfiable, then $\bar{V}$ preserves the privacy of $Q$ if either:*
*(i) $sch(Q) \ominus sch(V) \neq \emptyset$; or*
*(ii) for all $i \in \{1, \ldots, n\}, \bar{C}_Q$ and $\bar{C}_{V_i}$ conflict.*

We now consider the case where the published views may be defined over non-equivalent schemas. So our starting point is a set of published queries $S = \{V_1, \ldots, V_k\}$, and a secret query $Q$. We then group the queries of $S$ into sets, where the queries in each set are defined over equivalent schemas. If we denote these sets of queries by $\{V_{1_1}, \ldots, V_{1_m}\}, \ldots, \{V_{i_1}, \ldots, V_{i_m}\}$, where in each set we assume without loss of generality that the queries are defined over the same schema, then from each of these sets we generate the union of the queries as just outlined, denoted by $\{\bar{V}_1, \ldots, \bar{V}_i\}$ (assuming that variables are renamed if necessary so that each variable appears at most once in $\{\bar{V}_1, \ldots, \bar{V}_i\}$), and then combine to generate the query $\bar{V}$ defined by:

$$\bar{V}(\bar{X}) \leftarrow \bar{V}_1, \ldots, \bar{V}_i, C_1, \ldots, C_k,$$

where $\bar{X} = sch(\bar{V}_1) \cup \ldots \cup sch(\bar{V}_i)$, and $\{C_1, \ldots, C_k\}$ is a set of inequalities such that for every pair of variables $X, Y$ in $\{\bar{V}_1, \ldots, \bar{V}_i\}$ where $typ(X) = typ(Y)$, the inequality $X = Y$ is in $\{C_1, \ldots, C_k\}$. In effect, the inequalities $\{C_1, \ldots, C_k\}$ specify, in addition to joins within the queries $\bar{V}_1, \ldots, \bar{V}_i$, additional joins for every pair of variables that correspond to the same attribute in the database schema. Finally, we define the set $S$ to preserve the privacy of $Q$ if either:
(i) $sch(Q) \ominus sch(\bar{V}) \neq \emptyset$; or
(ii) the query $\bar{V}'$, defined by

$$\bar{V}'(sch(Q)) \leftarrow \bar{V},$$

is disjoint from $Q$.

From this definition, testing whether a set of views preserves the privacy of a secret query $Q$ can be reduced to testing whether a single query $\bar{V}'$ preserves the privacy of $Q$, which can be tested using Theorem 1 or Theorem 3.

A question that arises from our definition of privacy of a set of views is whether it is possible for the set $S$ to preserve the privacy of $Q$, but some proper subset of $S$ does not, i.e. the definition is not monotonic. As we now show, this cannot happen.

Theorem 5. *Let $Q$ be a secret DATALOG query, $S$ a set of DATALOG queries and $S'$ any subset of $S$. If $S$ preserves the privacy of $Q$, then so does $S'$.*

# 7. RELATIONSHIP BETWEEN PRIVACY AND PERFECT-SECURITY

In the section we examine the relationship between the notion of data privacy defined in this paper and perfect-security as defined in [23, 24]. Firstly, we present some definitions and results from [23, 24].

While the perfect-security approach allows tuples to contain values from any finite domain, we assume here that the domain is $\mathbb{Z}$, the set of positive and negative integers, in order to be consistent with our privacy approach. Assuming a relational database schema $D = \{R_1, \ldots, R_n\}$, let $tup(D, \mathbb{Z})$ denote the set of all possible tuples in any relation defined over a schema in $D$ using constants from $\mathbb{Z}$. The set $Inst(D)$ is the set of all databases defined over $D$ with tuples from $tup(D, \mathbb{Z})$. For each tuple $t \in tup(D, \mathbb{Z})$, we assume that there exists a probability distribution $\mathbf{P}$ such that $\mathbf{P}(t) \rightarrow [0, 1]$, which represents the probability that the tuple $t$ will appear in a database instance. Assuming that tuples are independent, $\mathbf{P}$ induces a probability distribution on the set of all instances defined over $D$ as follows: if $I \in Inst(D)$ then

$$\mathbf{P}[I] = \prod_{t_i \in I} \mathbf{P}(t_i) \cdot \prod_{t_j \notin I} (1 - \mathbf{P}(t_j)).$$

Given the probability distribution defined over instances, one then defines the probability that a query $Q$ returns an answer $q$ by:

$$\mathbf{P}(Q(I) = q) = \sum_{\{I \in Inst(D) | Q(I) = q\}} \mathbf{P}[I].$$

Perfect-security is then defined as follows.

Definition 13. *A query $Q$ is* perfectly-secure *w.r.t. a user query $V$ if for every possible probability distribution $\mathbf{P}$ and every possible answer $q$ to $Q$ and $v$ to $V$, the following holds:*

$$\mathbf{P}(Q(I) = q) = \mathbf{P}(Q(I) = q \mid V(I) = v).$$

Intuitively speaking, the definition says that perfect-security occurs when knowing the answer to $V$ never improves the chances of an adversary guessing the answer to $Q$.

Clearly the notion of perfect-security is with respect to the set of all possible probability distributions. However, the fundamental result from [23, 24] was to show that perfect-security can be reduced to logical statements, without probabilities, which require the existence of certain tuples. Before giving this result, the notion of a critical tuple is needed.

Definition 14. *A tuple $t \in tup(D, \mathbb{Z})$ is* critical *to a query $Q$ if there exists an instance $I \in Inst(D)$ and a tuple $t \in I$ such that $Q(I - \{t\}) \neq Q(I)$.*
*The set of all critical tuples for $Q$ is denoted by $crit(Q)$.*

The following main result from [23, 24] (Theorem 3.5) then relates perfect-security to critical tuples.

Theorem 6. *A secret query $Q$ is perfectly-secure w.r.t. a published query $V$ iff $crit(Q) \cap crit(V) = \emptyset$.*

## 7.1 Conjunctive Queries Defined over a Single Relation

The first case we consider is where $Q$ and $V$ contain a single term, i.e. they are selection-projection queries over a single relation. In general, perfect-security and privacy are not comparable for this case, as we now show by the following examples. The first example shows a case where $Q$ is perfectly-secure w.r.t. $V$, but $V$ does not preserve the privacy of $Q$.

Example 7. *Suppose we have a relational schema $R(P, I, S)$, as defined in Example 4, and define $Q$ and $V$ by:*

$$Q(P, I) \leftarrow R(P, I, S), S = \text{‘s1’}$$

$$V(P, I) \leftarrow R(P, I, S),\ S = \text{‘s2’}.$$

*We first note that $Q$ is perfectly-secure w.r.t. $V$. The reason is that any critical tuple in $r$ for $Q$ must have the form $\langle \_, \_, \text{‘s1’}\rangle$, and any critical tuple in $r$ for $V$ must have the form $\langle \_, \_, \text{‘s2’}\rangle$. Such tuples cannot be identical, and so $crit(Q)\ \cap crit(V) = \emptyset$ and hence $Q$ is perfectly secure w.r.t. $V$ from Theorem 6.*

*On the other hand, $V$ is not privacy preserving w.r.t. $Q$. To see this, if $r = \{\langle p1, i1, \text{‘s1’}\rangle, \langle p1, i1, \text{‘s2’}\rangle\}$, then $Q(r) \cap V(r) = \langle p1, i1\rangle \neq \emptyset$ and so $V$ is not privacy preserving w.r.t. $Q$.*

The next example shows the reverse situation - a case where $Q$ is not perfectly-secure w.r.t. $V$ but $V$ preserves the privacy of $Q$.

Example 8. *Let the schema $R$ be as in the previous example, and define $Q$ and $V$ by:*

$$Q(P, I, S) \leftarrow R(P, I, S)$$
$$V(P', I') \leftarrow R(P', I', S')$$

*Then $Q$ is not perfectly secure w.r.t. $V$ since any tuple in a non empty instance of $R$ is critical for both $Q$ and $V$, and so $crit(Q) \cap crit(V) \neq \emptyset$ and thus applying Theorem 6 shows that $Q$ is not perfectly-secure w.r.t. $V$. However, $V$ is privacy preserving w.r.t. $Q$ since $sch(Q) \ominus sch(V) \neq \emptyset$.*

These examples also illustrate an important difference between privacy and perfect-security, mentioned briefly previously, namely privacy is sensitive to changes in the schema of either the private query or user query, whereas perfect-security is not. To illustrate this, consider Example 7 and suppose that the bodies of both $Q$ and $V$ were unchanged but the schemas in the heads of both $Q$ and $V$ were changed to $(P, I, S)$. Then under this change, $Q$ would remain perfectly secure w.r.t. $V$ but now $V$ would preserve the privacy of $Q$ since the $S$-values of tuples in $Q$ and $V$ are different. Also, if in Example 8 the body of $V$ was unchanged but the schema in the head was changed to $(P', I', S')$, then $Q$ would remain not perfectly secure w.r.t. $V$, but now $V$ would violate the privacy of $Q$ since it is clear that $Q(r) \cap V(r) \neq \emptyset$ for any non-empty $r$.

These observations can be made more precise by the following result, which shows that for DATALOG queries, perfect-security is invariant under changes to the schema of the secret query, provided that the body of the query is left unchanged.

Lemma 3. *Let $Q, Q'$ and $V$ be DATALOG queries such that the bodies of $Q$ and $Q'$ are the same but the heads of the queries may be different. Then $Q$ is perfectly secure w.r.t. $V$ iff $Q'$ is perfectly secure w.r.t. $V$.*

In contrast, privacy preservation is dependent on the schema of the private query $Q$ and that of the published query $V$, as noted earlier. It follows directly from the definition of privacy preservation that if $V$ preserves the privacy of $Q$, then $V'$ also preserves the privacy of $Q'$, where $V'$ and $Q'$ are queries with the same bodies as $V$ and $Q$ but $sch(V) \preceq sch(V')$ and $sch(Q) \preceq sch(Q')$. However the converse of this is false, and so in general reducing the schema of $Q$ or $V$ results in the condition of privacy preservation being more difficult to satisfy.

While the previous examples show that in general perfect-security and privacy preservation cannot be compared in the case of queries defined over a single relation, the next result shows that they can be compared in the case where the private query and published query have the same schema. In this case, we now show that privacy preservation is a stronger notion than perfect-security for conjunctive queries.

Theorem 7. *Let $Q$ and $V$ be satisfiable DATALOG defined over a single relation such that $sch(Q) \equiv sch(V)$. Then $Q$ is perfectly-secure w.r.t. $V$ if $V$ preserves the privacy of $Q$.*

Our next result shows that privacy and perfect-security coincide if the schema of the published query is the same as the schema of the single relation in the database.

Theorem 8. *Let $Q$ and $V$ be satisfiable DATALOG queries defined over a single relation schema $R_1$ such that $sch(Q) \equiv sch(V) \equiv R_1$. Then $Q$ is perfectly-secure w.r.t. $V$ iff $V$ preserves the privacy of $Q$.*

## 7.2 Conjunctive Queries Defined over Multiple Relations

The first thing we note is that Theorem 7 does not hold for queries with multiple terms, i.e. privacy preservation is not a stronger condition than perfect-security, even when $Q$ and $V$ have equivalent schemas. We show this in the next example.

EXAMPLE 9. *Let $D = \{R_1(P, I, S), R_2(P, T)\}$, where $P$, $I, S$ have the same meaning as in Example 4 and $T$ represents the date that the patient was admitted. Also, let $Q$ and $V$ be defined by:*

$$Q(P, I, S, T) \leftarrow R_1(P, I, S), R_2(P, T), (T > 01/01/05)$$
$$V(P', I', S', T') \leftarrow R_1(P', I', S'), R_2(P', T'), (T' < 01/01/05)$$

*Now $V$ is privacy preserving w.r.t. $Q$ since the value of $T$ in every tuple returned by $V$ will be different from the value of $T'$ in every tuple returned by $Q$.*

*However, if patients can have more than one admission date, then $Q$ is not perfectly-secure w.r.t. $V$. To show this, let $I$ be the instance where $r_1 = \{\langle p, a, d\rangle\}$ and $r_2 = \{\langle p, 01/02/07\rangle, \langle p, 01/02/03\rangle\}$. Then the tuple $\langle p, a, d\rangle$ is in both $crit(Q)$ and $crit(V)$ since the results of both $Q(I)$ and $V(I)$ become empty when $\langle p, a, d\rangle$ is removed from $I$, and so $Q$ is not perfectly secure w.r.t. $V$ from Theorem 6. Intuitively speaking, $Q$ is not perfectly secure w.r.t. $V$ because even though $Q(I)$ and $V(I)$ are disjoint, they are still correlated through being generated from a common tuple in $R_1$.*

So in the case of queries with multiple terms, perfect-security and privacy are not comparable, even if the secret query and the published query are defined over the same schema. However, the following results show that perfect-security and privacy are comparable for some restricted cases. The first result shows the situation where perfect-security is a stronger condition than privacy.

LEMMA 4. *Let $Q$ and $V$ be DATALOG queries defined over a database $D = \{R_1, \ldots, R_n\}$ such that $sch(Q) \equiv sch(V)$. If there exists a relation schema $R_i \in D$ such that such that $R_i \preceq sch(Q)$ and $R_i$ is a term in $Q$, then $V$ preserves the privacy of $Q$ if $Q$ is perfectly-secure w.r.t. $V$.*

The next result shows a case where the reverse holds true, i.e. privacy is a stronger condition than perfect-security.

LEMMA 5. *Let $Q$ and $V$ be DATALOG queries defined over a database $D = \{R_1, \ldots, R_n\}$ such that $sch(Q) \equiv sch(V)$. If every tuple in $crit(Q)$ belongs to only one schema $R_i$, and $sch(Q) \preceq R_i$ and $R_i$ appears only once in the body of $Q$, then $Q$ is perfectly-secure w.r.t. $V$ if $V$ preserves the privacy of $Q$.*

Combining Lemma 4 and Lemma 5, we characterize when perfect-security and privacy coincide.

COROLLARY 6. *Let $Q$ and $V$ be DATALOG queries defined over a database $D = \{R_1, \ldots, R_n\}$ such that $sch(Q) \equiv sch(V)$. If every tuple in $crit(Q)$ belongs to only one schema $R_i$, and $sch(Q) \equiv R_i$ and $R_i$ appears only once in the body of $Q$, then $Q$ is perfectly-secure w.r.t. $V$ iff $V$ preserves the privacy of $Q$.*

We now summarize the results of this section. In the case of queries with a single term, perfect-security and privacy preservation are not comparable unless the schema of the secret query $Q$ and the user query $V$ are equivalent. In this case, privacy preservation implies perfect security. However in the case of the queries with multiple terms, privacy preservation and perfect-security are not comparable, even when $Q$ and $V$ have equivalent schemas. However, we also note in this case it follows Lemma 4 that by increasing the size of the schema of the secret query $Q$, privacy preservation can always be made a weaker condition than perfect-security.

We note that although the results of this section show that in general perfect-security and privacy are not comparable, we consider that (by adjusting the schema of the secret $Q$ if necessary) privacy should be used as a weaker method than perfect-security. This is because although privacy can be made to be a stricter condition than perfect-security, there seems to be little point in doing so in practice since, as a result of its information-theoretic foundation, perfect-security is already the most stringent definition that is required in practice. Moreover, as discussed in [6], the disadvantage of perfect-security is not that it is not restrictive enough, but that it is too restrictive and so can severely restrict the utility of published data. Thus we see that our notion of privacy would be applicable in situations where perfect-security was too strict, or computationally infeasible. In such situations, our privacy technique is an alternative method which has the feature that the degree of leakage that occurs can be controlled by adjusting the schema of the secret query (while keeping the body of the secret query unchanged).

## 8. RELATED WORK

The topic of privacy in statistical databases is a classical one that has been extensively investigated [2], and continues to be investigated [10]. The issue of privacy in statistical databases differs from the one considered in this article and in other recent articles in database publishing [23, 24, 7, 8]. In statistical databases, the database is a table of records about individuals which contains both sensitive attributes and non-sensitive attributes. The aim of privacy preservation in statistical databases is then to allow the publication of aggregated views over the table without compromising sensitive information about individuals. In contrast, in our scenario both the sensitive data and the published data are specified by non-aggregated queries, possibly over multiple tables. We also note that when publishing aggregated views in statistical databases, some leakage on sensitive information is unavoidable. Hence the focus is then on minimizing the leakage of sensitive information or ensuring that it is within predefined bounds by using techniques such as generalization or adding noise to the views, rather than on preventing any leakage at all [10].

As noted previously, several articles have addressed the issue of perfect-security in database publishing. The notion of perfect-security was formally defined in [23, 24], and the fundamental result that reduces the test for perfect-security to checking for common critical tuples was established. Several other issues were also investigated in [23, 24]. The first was to investigate the effect of an adversary having extra information in addition to knowing the probability distribution $\mathbf{P}$ of the tuples in the database. This additional information could take the form of knowledge of integrity constraints, or cardinality constraints, that hold in the database. The fundamental result characterizing perfect-security was then extended to the case of prior knowledge for Boolean queries. The other topic investigated in [23, 24] was to formally define the amount of leakage that occurs, given that perfect-security is not satisfied, and to then prove a result for ensuring that the amount of leakage is less than some predefined bound.

Given that perfect-security is sometimes too strict, the issue of relaxing it was later investigated [7, 6]. The model proposed in [7] was for individual tuples to have a uniform probability distribution, but such that the expected size of each relation instance remains a constant while the size of the underlying domain tends to infinity. One can then define $\mu_n[Q]$ as the probability that a Boolean query $Q$ is true on a domain size $n$, and it was then shown that lim $_{n\to\infty}\mu_n[Q]$ can be computed for conjunctive queries. Given a user query $V$, practical-security was defined to occur if $\lim_{n\to\infty}\mu_n[Q \mid V] = 0$. It was then shown that for conjunctive Boolean queries $Q$ and $V$, $\lim_{n\to\infty}\mu_n[Q \mid V] = 0$ exists and an algorithm was provided for computing it, and thus also providing a method for determining whether or not practical-security occurs. In [7], the complexity of several problems relating to computing $\lim_{n\to\infty}\mu_n[Q \mid V]$ were investigated.

Given that testing for perfect-security is computationally intractable [23, 24], the focus in [20] was on identifying sub-cases where testing is tractable. Using a method based on reducing perfect-security to the well known problem of conjunctive query containment [1], several important tractable sub-classes, such as queries with no self-joins or acylic queries, were identified.

A different aspect of privacy in database publishing was investigated in [8], which is as follows. It is assumed that a database owner has already published views $\{V_1, \ldots, V_n\}$, which may have leaked some information about the secret query $Q$. The owner of the database is comfortable with this level of leakage, but wants to ensure that the publication of an additional view $V_{n+1}$ does not leak additional sensitive information. The approach adopted is similar to the one used in perfect-security, and models the adversary's knowledge as a probability distribution $\mathbf{P}$ over the set of possible database instances. Hence extra leakage is defined to occur if the publication of $V_{n+1}$ results in the adversary changing their guess about $Q$ from what it was after knowing $\{V_1, \ldots, V_n\}$, i.e. $\mathbf{P}(Q \mid V_1, \ldots, V_n) \neq \mathbf{P}(Q \mid V_1, \ldots, V_n, V_{n+1})$. Similar to what occurs in perfect-privacy, it was shown that testing for additional leakage can be reduced from a probabilistic test to a logical one involving the intersection of possible database instances. The effect of assuming that the database instance satisfies a set of integrity constraints, and the computational complexity of testing for additional information leakage, were also investigated.

A chase style algorithm for determining when two conjunctive queries are disjoint was also investigated in [11], in a different setting to the one used in this article. The underlying domain was assumed to be dense (e.g. the rational numbers), and $\neq$ was allowed as a comparison operator.

In addition to the issue of detecting privacy violations, the related topic of transforming published data for data analysis or mining so that privacy violations do not occur has also been widely investigated, mainly for the case where the sensitive information is specified by specific attributes in tabular data. Two of the most common methods are $k$-anonymity [27] and $l$-diversity [21]. Both techniques rely on the generalization of data, in which specific attributes are replaced by a less specific attribute so that the value of a sensitive attribute for an individual cannot be uniquely determined from the publication of the sanitized data. For example, a "city" attribute may be replaced by a "region" attribute, or an "age" attribute may be replaced by an "age range" attribute. Other techniques for transforming published data to prevent privacy violations include perturbation [9] and adding noise [5].

# 9. CONCLUSIONS

We have investigated the issue of detecting privacy violations in database publishing. We have proposed a new method of detecting possible privacy violations, based on determining if a secret query and a published view can return tuples in common. We then showed that for conjunctive queries, privacy preservation can be computed in polynomial time and also gave a necessary and sufficient condition for privacy preservation for queries with negation. The effect of FD constraints on privacy was investigated, and we gave a sufficient condition for privacy preservation in the presence of FD constraints. We then investigated privacy preservation when multiple views are published, and showed that collusion can occur in our model. We then showed that privacy violations for multiple views can be detected by reducing the problem to that of detecting privacy violation by a single query that is a join over the published views. Finally, we investigated the relationship between our definition of privacy and that of an alternate approach, namely that of perfect-security [23, 24].

Several conclusions can be drawn from our work. The first is that it is unlikely that a single definition of privacy will apply in all situations. The method adopted in a specific application will depend on both the type and sensitivity of the private data, and also on the amount of computational resources available to check for privacy violations. It also depends on where the database owner would like to have the balancing point between privacy and utility - if privacy takes precedence, then a strict definition of privacy violation may be preferred, if utility takes precedence then a more relaxed method may be preferable. Also, in general there appears to be tradeoff between strictness and efficiency and so the more stringent the definition of a privacy violation is, the more expensive it is to detect.

There are several extensions of the work in this article that we intend to pursue. Firstly, we have investigated the effect of integrity constraints on privacy for a limited class of constraints. It would be useful to extend this work to more general classes of integrity constraints, such as those considered in [8]. Secondly, our approach is intended to be a weaker approach to privacy violations than that of perfect-security, and so this means that some leakage of sensitive information may occur, even when a published query preserves the privacy of a secret query in our definition. The perfect-security approach allows one to quantify the amount of leakage that occurs in publishing a view that is not perfectly-secure, and it would be useful to use this approach to derive an estimate of the amount of leakage that occurs when a secret query is not perfect-secure w.r.t. a published view, but the view preserves the privacy of the secret query according to our definition.

# 10. REFERENCES

[1] Abiteboul, S., Hull, R., Vianu, V.: Foundations of databases. Addison Wesley (1996)

[2] Adam, N.R., Wortman, J.: Security-control methods in. ACM Computing Surveys **21**(4), 515–556 (1989)

[3] Antova, L., Koch, C.: On API's for probabilistic databases. In: Workshop on management of Uncertain Data (2008)

[4] Castano, S., Fugini, M., Martella, G., Samarati, P.: Database Security. Addison-Wesley (1995)

[5] Chawla, S., Dwork, C., McSherry, F., Smoth, A., Wee, H.: Towards privacy in public databases. In: TCC, pp. 363–385 (2005)

[6] Dalvi, N., D.Suciu: Management of probabilistic data foundations and challenges. In: PODS, pp. 1–12 (2007)

[7] Dalvi, N., Miklau, G., D.Suciu: Asymptotic conditional probabilities for conjunctive queries. In: ICDT, pp. 289–305 (2005)

[8] Deutsch, A., Papakonstantinou, Y.: Privacy in database publishing. In: ICDT, pp. 230–245 (2006)

[9] Dunar, I., Nissim, K.: Revealing information while preserving privacy. In: PODS, pp. 202–210 (2003)

[10] Dwork, C.: Ask a better question, get a better answer: a new approach to private data analysis. In: ICDT, pp. 18–27 (2007)

[11] Elkan, C.: A decision procedure for conjunctive query disjointness. In: PODS, pp. 134–139 (1989)

[12] Elkan, C.: Independence of logic database queries and updates. In: ACM PODS Conference, pp. 154–160 (1990)

[13] Fan, W.: XML publishing: Bridging theory and practice. In: DBPL, pp. 1–16 (2007)

[14] Garfinkel, S.: Database nation: the death of privacy in the 21st century. O'Reillly and Associates (2001)

[15] Gupta, A., Mumick, I.: Maintenance of materialized views: Problems, techniques, and applications. Data Engineering Bulletin **18**(2), 1–16 (1995)

[16] Gupta, A., Mumick, I.S.: Materialized Views. MIT Press (1999)

[17] Halevy, A., Ives, Z., Suciu, D., Tatarinov, I.: Schema mediation in peer data management systems. In: ICDE, pp. 505– (2003)

[18] Kifer, D., Gehrke, J.: Injecting utility into anonymized data sets. In: SIGMOD, pp. 217–228 (2006)

[19] Levy, A.Y., Sagiv, Y.: Queries independent of update. In: VLDB, pp. 171 – 181 (1993)

[20] Machanavajjhala, A., Gehrke, J.: On the efficiency of checking perfect privacy. In: PODS, pp. 163–172 (2006)

[21] Machanavajjhala, A., Gehrke, J., Kifer, D., Venkitasubramanian, M.: $l$-diversity: Privacy beyond $k$-anonymity. In: ICDE (2006)

[22] Miklau, G., D.Suciu: Controlling access to published data using cryptography. In: VLDB, pp. 898–909 (2003)

[23] Miklau, G., D.Suciu: A formal analysis of information disclosure in data exchange. In: SIGMOD, pp. 575–586 (2004)

[24] Miklau, G., D.Suciu: A formal analysis of information disclosure in data exchange. JCSS **73** (2007)

[25] Ng, W.S., Ooi, B.C., Tan, K.L., Zhou, A.: PeerDB: a P2P based system for distributed data sharing. In: ICDE, pp. 633–644 (2003)

[26] Rastogi, V., Hong, S., Suciu, D.: The boundary betwen privacy and utility in database publishing. In: VLDB, pp. 531–542 (2007)

[27] Sweeney, L.: k-anonymity: a model for protecting privacy. International Journal on Uncertainty, Fuzziness and Knowledge-based Systems **10**(5) (2002)

# APPENDIX

Proofs of all results in the this article can be found in the full version:
http://www.cis.unisa.edu.au:80/∼cismwv/papers/edbt-09-full.pdf.