Access Control for Data Integration in Presence of Data Dependencies

Mehdi Haddad[1] *, Jovan Stevovic[2,3], Annamaria Chiasera[3], Yannis Velegrakis[2], and Mohand-Saïd Hacid[4]

[1] INSA-Lyon, CNRS, LIRIS, UMR5205, F-69621, France,
[2] Information Engineering and Computer Science, University of Trento, Italy,
[3] Centro Ricerche GPI, Trento, Italy
[4] Université Lyon 1, CNRS, LIRIS, UMR5205, F-69621, France,
{mehdi.haddad, mohand-said.hacid}@liris.cnrs.fr
{stevovic, velgias}@disi.unitn.eu
annamaria.chiasera@cr-gpi.it

Abstract. Defining access control policies in a data integration scenario is a challenging task. In such a scenario typically each source specifies its local access control policy and cannot anticipate data inferences that can arise when data is integrated at the mediator level. Inferences, e.g., using functional dependencies, can allow malicious users to obtain, at the mediator level, prohibited information by linking multiple queries and thus violating the local policies. In this paper, we propose a framework, *i.e.,* a methodology and a set of algorithms, to prevent such violations. First, we use a graph-based approach to identify sets of queries, called violating transactions, and then we propose an approach to forbid the execution of those transactions by identifying additional access control rules that should be added to the mediator. We also state the complexity of the algorithms and discuss a set of experiments we conducted by using both real and synthetic datasets. Tests also confirm the complexity and upper bounds in worst-case scenarios of the proposed algorithms.

1 Introduction

Data integration offers a convenient way to query different data sources while using a unique entry point that is typically called *mediator*. Although this ability to synthesize and combine information maximizes the answers provided to the user, some privacy issues could arise in such a scenario. The authorization policies governing the way data is accessed are defined by each source at local level without taking into consideration data of other sources. In relational and other systems, data constraints or hidden associations between attributes at the mediator level could be used by a malicious user to retrieve prohibited information. One type of such constraints are the *functional dependencies* (FDs). When FDs are combined with authorized information, they may allow the disclosure of

* This work is partially supported by Rhône-Alpes Region in the Framework of Explora'Doc.

some prohibited information. In these cases, there is a need for providing additional mechanisms at the mediator level to forbid the leakage of any prohibited information.

In this work we aim at assisting administrators in identifying such faults and defining additional access control rules at the mediator level to remedy the inference problem. Given a (relational) schema of the mediator, the sources' policies and a set of FDs, we propose a set of algorithms that are able to identify violating transactions. These transactions correspond to sets of queries that violate the sources' policies if used in conjunction with FDs. To avoid the completion of a transaction, and therefore the violation of any source's policy, we propose a query cancellation algorithm that identifies a *minimum set* of queries that need to be forbidden. The identified set of queries is then used to generate additional rules to be added to the existing set of rules of the mediator.

The reminder of the paper is organized as follows. Section 2 gives an overview of research effort in related areas. Section 3 provides definitions of the main (technical) concepts we use in the paper. Section 4 introduces a motivating scenario, the integration approach and challenges posed by functional dependencies. In Section 5 we describe our methodology. Section 6 describes the *detection phase* that identifies the policy violations. Section 7 describes the *reconfiguration phase* that deals with flaws identified in the detection phase. Section 8 describes the experiments. Finally, we conclude in Section 9.

2 Related Work

Access control in information integration is a challenging and fundamental task to be achieved [7]. In [3], the authors consider the use of metadata to model both purposes of access and user preferences. Our work does not consider these concepts explicitly, but they could be simulated by using predicates while defining the authorization rules. The authors of [14] analyzed different aspects related to access control in federated contexts [25]. They identified the role of administrators at mediator and local levels and proposed an access control model that accommodates both mediator and source policies. In [21], the authors propose an access control model based on both allow and deny rules and algorithms that check if a query containing joins can be authorized. This work, like the previous one, does not consider any association/correlation between attributes or objects that can arise at global level when joining different independent sources.

Sensitive associations happen when some attributes, when put together, lead to disclosure of prohibited information. Preventing the access to sensitive associations became crucial (see, e.g., [2, 11]) in a distributed environment where each source could provide one part of it. In [2], the authors proposed a distributed architecture to ensure no association between attributes could be performed while in [11] fragmentation is used to ensure that each part of sensitive information is stored in a different fragment. In [12], the authors propose an approach to evaluate whether a query is allowed against all the authorization rules. It targets

query evaluation phase while our goal is to *derive additional authorization* rules to be added to the mediator.

In [28], the authors provide an anonymization algorithm that considers FDs while identifying which portion of data needs to be anonymized. In our case, we focus on defining access control policies that should be used to avoid privacy breaches instead of applying privacy-preserving techniques [16].

Inferences allow indirect accesses that could compromise sensitive information (see [15] for a survey). A lot of work has been devoted to the inference channel in multilevel secure database. In [13] and [26] for each inference channel that is detected, either the schema of the database is modified or security level is increased. In [13], a conceptual graph based approach has been used to capture semantic relationships between entities. The authors show that this kind of relationships could lead to inference violations. In [26], the authors consider inference problem using FDs. This work does not consider authorization rules dealing with implicit association of attributes; instead, the authors assume that the user knows the mapping between the attributes of any FD. Other approaches such as [9, 27] analyze queries at runtime and if a query creates an inference channel then it is rejected. In [27], both queries and authorization rules are specified using first order logic. While the inference engine considers the past queries, the functional dependencies are not taken into account. In [9], a history-based approach has been considered for the inference problem. The authors have considered two settings: the first one is related to the particular instance of the database. The second is only related to the schema of both relations and queries. In our work, we focus on inferences to identify additional access rules to be added to the mediator. In [17], we investigated how join queries could lead to authorization violations. In this current paper, we generalize the approach in [17] by considering the data inference problem.

3 Preliminaries

Before describing our approach, a number of introductory definitions are needed:

Definition 1 (Datalog rule). *[1] A (datalog) rule is an expression of the form* $R_1(u_1) :- R_2(u_2), ..., R_n(u_n)$*, where* $n \geq 1$*,* $R_1, ..., R_n$ *are relation names and* $u_1, ..., u_n$ *are free tuples of appropriate arities. Each variable occurring in* u_1 *must also occur in at least one of* $u_2, ..., u_n$*.*

Definition 2 (Authorization policy). *An authorization policy is a set of authorization rules. An authorization rule is a view that describes the part of data that is prohibited to the user. An authorization rule will be expressed using an augmented datalog rule. This augmentation consists in adding a set of predicates characterizing the users to whom the authorization rule applies.*

Definition 3 (Violating Transaction). *A violating transaction* T *is a set of queries such that if they are executed and their results combined, they will lead to disclosure of sensitive information and thus violating the authorization policy.*

Definition 4 (Functional Dependency). *[23] A functional dependency over a schema R (or simply an FD) is a statement of the form:*
$R : X \rightarrow Y$ (or simply $X \rightarrow Y$ whenever R is understood from the context), where $X, Y \subseteq$ schema(R). We refer to X as the left hand side (LHS) and Y as the right hand side (RHS) of the functional dependency $X \rightarrow Y$.

A functional dependency $R : X \rightarrow Y$ is satisfied in a relation r over R, denoted by $r \models R : X \rightarrow Y$, iff $\forall\ t_1, t_2 \in r$ if $t_1[X] = t_2[X]$, then $t_1[Y] = t_2[Y]$.

Definition 5 (Pseudo transitivity rule). *[23] The pseudo transitivity rule is an inference rule that could be derived from Armstrong rules [5]. This rule states that if $X \rightarrow Y$ and $YW \rightarrow Z$ then $XW \rightarrow Z$.*

Without loss of generality we consider functional dependencies having only one attribute in their RHS. A functional dependency of the form $X \rightarrow YZ$ could always be replaced by $X \rightarrow Y$ and $X \rightarrow Z$ by using the decomposition rule [23] which is defined as follows: if $F \vdash X \rightarrow YZ$, then $F \vdash X \rightarrow Y$ and $F \vdash X \rightarrow Z$.

4 Motivating Scenario

We consider a healthcare scenario inspired by one of our previous works while developing an Electronic Health Record (EHR) in Italy [4]. The EHR represents the mediator which provides mechanisms to share data and to enforce the appropriate authorizations [21]. From that scenario we extract an example that describes how FDs can impact access control and can be challenging to tackle at the mediator level.

Global as View Integration. We consider a data integration scenario where a Global As View (GAV) [22] approach is used to define a mediator over three sources. Particularly, we consider the sources S_1, S_2 and S_3 with the following local schemas: $S_1(SSN, Diagnosis, Doctor)$ contains the patient social security number (SSN) together with the diagnosis and the doctor in charge of her/him, $S_2(SSN, AdmissionT)$ provides the patient admission timestamp, $S_3(SSN, Service)$ provides the service to which a patient has been assigned.

The mediator virtual relation, according to the GAV integration approach, is defined by using relations of the sources. We consider a single virtual relation to simplify the scenario but the same reasoning applies for a mediator's schema composed by a set of virtual relations. In our example, the mediator will combine the data of the sources joined over the SSN attribute as shown by rule (1).

$$M(SSN, Diagnosis, Doctor, AdmissionT, Service) :- \\ S_1(SSN, Diagnosis, Doctor), S_2(SSN, AdmissionT), S_3(SSN, Service). \quad (1)$$

Authorization Policies are specified by each source on its local schema and propagated to the mediator. In our example, we assume two categories of users: doctors and nurses. For S_1, doctors can access SSN and $Diagnosis$ while nurses can access either SSN or $Diagnosis$ but not their association (i.e., simultaneously). The rule (2) expresses this policy in form of a prohibition.

$$R_1(SSN, Diagnosis) : -S_1(SSN, Diagnosis), role = nurse. \quad (2)$$

The other sources allow accessing to their content without restrictions both for doctors and nurses, therefore there are no more authorization rules to specify.

At the Mediator, authorization rules are propagated by the sources aiming at preserving their policies. The propagation can lead to policy inconsistencies and conflicts [14]. These issues are out of the scope of this paper. In our example there is only one rule defined by S_1 to be propagated at the mediator.

We then assume that at the mediator the following FDs are identified, either manually during the schema definition or by analyzing the data with algorithms such as TANE[20]:

$$(AdmissionT, Service) \rightarrow SSN \quad (F_1)$$
$$(AdmissionT, Doctor) \rightarrow Diagnosis \quad (F_2)$$

F_1 holds because at each service there is only one patient that is admitted at a given time $AdmissionT$. Note that $AdmissionT$ represents the admission timestamp including hours, minutes and seconds. F_2 holds because at a given timestamp, a doctor could make only one diagnosis.

Let see how $\mathcal{FD}$ could be used by a malicious user to violate the rule (2). Let us assume the following queries are issued by a nurse: $Q_1(SSN, AdmissionT, Service)$ and then $Q_2(Diagnosis, AdmissionT, Service)$. Combining the results of the two queries and using the functional dependency F_1, the nurse can obtain SSN and $Diagnosis$ simultaneously, which induces the violation of the authorization rule (2). To do so, the nurse could proceed as follows: (a) join the result of Q_1 with those of Q_2 on the attributes $AdmissionT$ and $Service$; (b) take advantage of F_1 to obtain the association between SSN and $Diagnosis$.

From now on, we refer to a query set like $\{Q_1, Q_2\}$ as a *violating transaction*. Indeed, both F_1 and F_2 do not hold in any source. They both use attributes provided by different sources. Thus, the semantic constraints expressed by these functional dependencies could not be considered by any source while defining its policy. This example highlights the limitation of the naïve propagation of the policies of the sources to the mediator. In the next section, we propose an intuitive approach for solving this problem.

5 Approach

We propose a methodology that aims at detecting all the possible violations that could occur at the mediator level by first identifying all the violating transactions and then disallowing completion of such violating transactions.

Our approach relies on the following settings: we consider the relational model as the reference model, both user queries and datalog expressions denoting authorization rules (see Section 3) are conjunctive queries and the mediator is defined following the GAV (Global As a View) data integration approach. This means that each virtual relation of the mediator is defined using a conjunctive query over some relations of the sources.

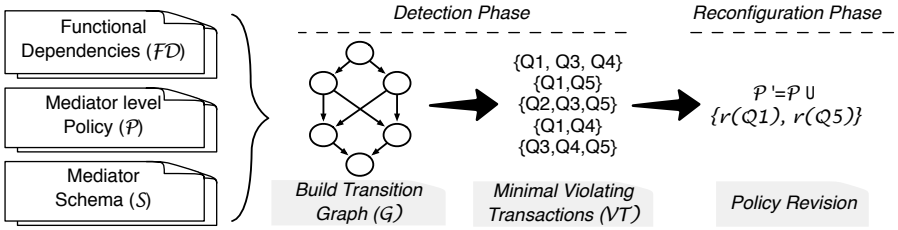

Fig. 1. The proposed methodology to identify violating transactions and define additional rules.

Currently we do not consider other types of inferences or background, external or adversarial knowledge that refer to the additional knowledge the user may have while querying a source of information [24, 10]. These aspects are important but they are out of the scope of this paper.

The proposed methodology, as shown in Figure 1, consists of a sequence of phases and steps involving appropriate algorithms. It takes as input a set of functional dependencies ($\mathcal{FD}$), the policy ($\mathcal{P}$) and the schema ($\mathcal{S}$) of the mediator and applies the following phases:

1. ***Detection phase:*** aims at identifying all the violations that could occur using $\mathcal{FD}$. Each of the resulting transactions represents a potential violation. Indeed, as shown previously, the combination of all the queries of a single transaction induces an authorization violation. This phase is performed by the following steps:
   - *Construction of a transition graph ($\mathcal{G}$):* this is done for each authorization rule by using the set of provided functional dependencies ($\mathcal{FD}$).
   - *Identification of the set of Minimal*[5] *Violating Transactions ($\mathcal{VT}$):* it consists in identifying all the different paths between nodes in $\mathcal{G}$ to generate the set of minimal violating transactions.
2. ***Reconfiguration phase:*** it proposes an approach to forbid the completion of each transaction in $\mathcal{VT}$ identified in the previous phase. By completion of a transaction we mean issuing and evaluating all the queries of that transaction. A rule is violated only if the entire transaction is completed. This phase modifies/repairs the authorization policy in such a way that no $\mathcal{VT}$ could be completed.

6 Detection phase

In the detection phase we enumerate all the violating transactions that could occur considering the authorization rules as queries that need to be forbidden. The idea is to find all the transactions (i.e., a set of queries) that could match the query corresponding to the authorization rule.

[5] The concept of minimality is detailed in Section 6.2.

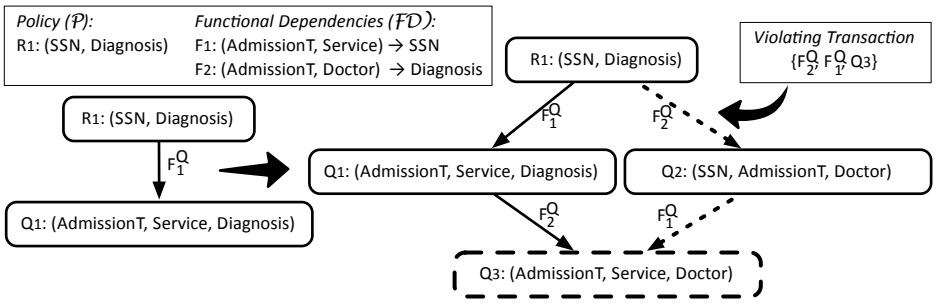

Fig. 2. Graph construction and violating transactions identification

6.1 Building the Transition Graph

The aim of the transition graph is to list all the queries that could be derived from an authorization rule using functional dependencies. For each authorization rule we use $\mathcal{FD}$ to derive a transition graph ($\mathcal{G}$) as shown in Figure 2. To build $\mathcal{G}$ we resort to Algorithm 1 as follows:

1. Consider the set of attributes of an authorization rule as the initial node.
2. For each FD in $\mathcal{FD}$ that has the RHS attribute inside the current node (starting from the root):
   (a) Create a new node by replacing the RHS attribute of the node with the set of attributes of the LHS of FD.
   (b) Create an edge between the two nodes and label it with F^Q (see Definition 6) corresponding to the FD that has been used.
3. Apply the same process for the new node.

6.2 Identifying Violating Transactions

The set of minimal violating transactions ($\mathcal{VT}$) is constructed as follows. First a path between the initial node (the node representing the authorization rule) and every other node is considered. As shown in Figure 2, from this path a transaction (*i.e.,* a set of queries) is constructed. Each query that is used as a label on this path is added to the transaction. Finally, the query of the final node of the path is also added to the transaction. This is done for all nodes and paths in $\mathcal{G}$. Before showing how minimality of the $\mathcal{VT}$ is ensured let us introduce the following definitions.

Definition 6 (Building a query from a functional dependency). *Let F be a functional dependency. We define F^Q as the query that projects on all the attributes that appear in F, either in the RHS or in the LHS. For example, let $R(A_1, A_2, A_3, A_4)$ be a relation and let F be the functional dependency $A_1, A_2 \rightarrow A_3$ that holds on R. In this case F^Q is the query that projects on all the attributes that appear in F. F^Q is the query $F^Q(A_1, A_2, A_3) :- R(A_1, A_2, A_3, A_4)$.*

Algorithm 1: BuildTransitionGraph (BuildG)

input : r_i the rule $r_i \in P$,
$\mathcal{FD}$ the set of functional dependencies.
output: $\mathcal{G}(V, E)$ the transition graph

```
1  V := {v(r_i)}; // create the root v with the attributes of r_i
2  W := {v(r_i)}; // add v also to a set W of vertexes to visit
3  foreach w ∈ W do
4      W := W − {w};
5      foreach FD(LHS → RHS) ∈ FD do
6          if RHS ∈ w then                          // RHS is one attribute
7              x := w − {RHS} + LHS; // create new vertex
8              if x ∉ V then
9                  V := V + {x};
10                 W := W + {x};
11             e := (w, x, LHS + {RHS}); // e is a new edge from w to x
                   with as transition the attributes LHS + {RHS}
12             if e ∉ E then                  // if not already in E add it
13                 E := E + {e};
14 return G(V, E) ;
```

Definition 7 (Minimal Query). *A query Q is minimal if all its attributes are relevant, that is $\forall Q' \subset Q : Q'$ cannot be used instead of Q in a violating transaction.*

Definition 8 (Minimal Violating Transaction). *A violating transaction T (see Section 3) is minimal if: (a) all its queries are minimal, and (b) all its queries are relevant i.e. $\forall Q \in T : T \setminus \{Q\}$ is not a violating transaction.*

To generate the minimal set of transactions ($\mathcal{VT}$) that is compliant with the definition 8, we use the recursive Algorithm 2. The initial call to the algorithm is: $\mathcal{VT} := FindVT(\mathcal{G}, root, \emptyset, \emptyset)$

The example in Figure 2 contains three nodes Q_1, Q_2 and Q_3 in addition to the initial node R_1. If we apply Algorithm 2, it will generate, for each node Q_i, a transaction containing each F^Q on the path between R_1 and Q_i, and Q_i itself. For example, to generate T_3 that represents the path between R_1 and Q_3, we start by adding each F_i on the path from R_1 to Q_3. Here, F_1 and F_2 are translated into F_1^Q and F_2^Q respectively. Finally, we add Q_3. Thus, we obtain $T_3 = \{F_1^Q, F_2^Q, Q_3\}$. In the example the returned $\mathcal{VT}$ is: $\mathcal{VT} = \{T_1 = \{Q_1, F_1^Q\}, T_2 = \{Q_2, F_2^Q\}, T_3 = \{Q_3, F_1^Q, F_2^Q\}\}$. At this stage we emphasized the fact that $\mathcal{FD}$ could be combined with authorized queries to obtain sensitive information. In our example, this issue is illustrated by the fact that if all the queries of any transaction T_i are issued then the authorization rule $R_1(SSN, Diagnosis)$ is violated. To cope with this problem and prohibit transaction completion, we propose an approach that

Algorithm 2: FindViolatingTransactions (FindVT)

input : $\mathcal{G}(V, E)$ the transition graph, v current vertex, c_t current path, $\mathcal{VT}$ current set of transactions.
output: $\mathcal{VT}$ the set of minimal violating transactions.

```
1 foreach e ∈ outgoing edges of v do
2     t := c_t + e.transition + e.to ;
      // e.transition is the set of attributes of the transition
         while e.to is the destination node
3     if ∄k ∈ VT | k ⊆ t then //if t is minimal with respect to ∀k ∈ VT
4         VT := VT + {t} ;
5         foreach k ∈ VT do
6             if t ⊆ k then     // if k is not minimal with respect to t
7                 VT := VT − {k} ;
                  // reducing further VT
8     return FindVT(G, e.to, c_t + e.transition, VT);
      // recursive call with the v reached by e (e.to) by adding the
         e.transition to the current VT
```

repairs the set of authorization rules with additional rules in such a way that no violation could occur.

7 Reconfiguration phase

This phase aims at preventing a user from issuing all the queries of a violating transaction. If a user could not complete the execution of all the queries of any violating transaction then no violation could occur.

The reconfiguration phase revises the policy by adding new rules such that no violating transaction could be completed. A naïve approach could be to deny one query for each transaction. Although this naïve solution is safe from an access control point of view, it is not desired from an availability point of view. To achieve a trade off between authorization enforcement and availability, we investigate the problem of finding the minimal set of queries that denies at least one query for each violating transaction. We refer to this problem as *query cancellation problem*. We first *formalize* and characterize the *complexity* of the query cancellation problem for one rule. Then, we discuss the case of a policy (*i.e.,* a set of rules).

7.1 Problem formalization

Let $\mathcal{VT} = \{T_1, \ldots, T_n\}$ be a set of minimal violating transactions and let $\mathcal{Q} = \{Q_1, \ldots, Q_m\}$ be a set of queries such that $\forall i \in \{1, \ldots, n\} : T_i \in \mathcal{P}(Q) \setminus \emptyset$. We define the following *Query Cancellation (QC)* recognition (decision) problem as follows:

– **Instance:** a set $\mathcal{VT}$, a set $\mathcal{Q}$ and a positive integer k.
– **Question:** is there a subset $Q \subseteq \mathcal{Q}$ with $|Q| \leq k$ such that $\forall i \in \{1, \ldots, n\}$: $T_i \setminus Q \neq T_i$? Here, $|Q|$ denotes the cardinality of Q.

Algorithm 3: QueryCancellation

input : $\mathcal{VT}$ is the set of minimal violating transactions.
Q is the set of all the queries that appear in $\mathcal{VT}$
output: S is the set of all solutions

```
1 foreach q ∈ Q do
2     if ∀t ∈ VT, t ∩ q ≠ ∅ then
3         S := S ∪ q ;
4 return S ;
```

Thus, the optimization problem, which consists in finding the *minimum number of queries* to be cancelled is called *Minimum Query Cancellation (MQC).*

7.2 Problem complexity

In this section, we show the NP-completeness of QC. We propose a reduction from the domination problem in split graphs [8]. In an undirected graph $\mathcal{G} = (V, E)$, where V is the node (vertex) set and E is the edge set, each node *dominates* all nodes joined to it by an edge (neighbors). Let $D \subseteq V$ be a subset of nodes. D is a dominating set of $\mathcal{G}$ if D dominates all nodes of $V \setminus D$. The usual *Dominating Set (DS)*[8] decision problem is stated as follows:

– **Instance:** a graph G and a positive integer k.
– **Question:** does G admits a dominating set of size at most k ?

This problem has been proven to be NP-complete even for split graphs [8]. Recall that a split graph is a graph whose set of nodes is partitioned into a clique C and an independent set I. In other words, all nodes of C are joined by an edge and there is no edge between nodes of I. Edges between nodes of C and nodes of I could be arbitrary.

Theorem 1. *QC is NP-complete.*

Proof. QC belongs to NP since checking if the deletion of a subset of queries affects all transactions could be performed in polynomial time. Let G be a split graph such that C is the set of nodes forming the clique and I is the set of nodes forming the independent set. We construct an instance QC of query cancellation problem from $\mathcal{G}$ as follows: $\mathcal{Q} = C$, $\mathcal{VT} = I$ and each transaction T_i is the set of queries that are joined to it by an edge in G. We then prove that G admits a dominating set of size at most k if and only if QC admits a subset $Q \subseteq \mathcal{Q}$ of size at most k such that $\forall i \in \{1, \ldots, n\} : T_i \setminus Q \neq T_i$.

Assume QC admits a subset $Q \subseteq \mathcal{Q}$ of size at most k such that $\forall i \in \{1, \ldots, n\} : T_i \setminus Q \neq T_i$. Q is also a dominating set of $\mathcal{G}$. In fact, all nodes of I are dominated since all the transactions are affected by Q and all remaining nodes in the clique C are also dominated since they all are connected with nodes of Q. Assume $\mathcal{G}$ admits a dominating set D of size at most k. Observe that D could be transformed into a dominating set D' of same size and having all its nodes in C. To ensure this transformation it is sufficient to replace all nodes of D that are in I by any of their neighbors in C. Note that the obtained set D' is also a dominating set of $\mathcal{G}$. The subset of queries to be canceled is then computed by setting Q to D'.

Thus, we can deduce the following:

Corollary 1. *MQC is NP-hard.*

To generate the set of queries that need to be canceled we use Algorithm 3. It returns all the (candidate) sets of queries that have a non-empty intersection with each violating transaction. We can use different metrics to determine which set to choose. The first metric is the cardinality of the smallest set. Other metrics could be defined by the administrator. Indeed, some queries can be identified as more relevant to the application. In this case, the set of queries to be chosen could be the one that does not contain any relevant query. The minimal set of queries MQ is defined using one of the previous metrics. For each query Q in MQ a new authorization rule is added to prevent from the evaluation of Q.

In our example, the QC algorithm will return three different candidate sets of solutions to be added to $\mathcal{P}$. These sets are: $\{r(Q_1), r(F_2^Q)\}$, $\{r(Q_2), r(F_1^Q)\}$, $\{r(F_1^Q), r(F_2^Q)\}$. If we choose the first candidate set then we will have $\mathcal{P} = \{R_1(SSN, Diagnosis), R_2(AdmT, Service, Diag.), R_3(AdmT, Doctor, Diag.)\}$.

Algorithm 4: GenerelizationForPolicy

input : $\mathcal{P}$ the set of authorization rules.
output: $\mathcal{P}$ augmented with new rules.

```
1  foreach r_i ∈ P do
2      G := BuildG(r_i);
3      VT := FindVT(G, root, ∅, ∅);
4      S := QueryCancellation(VT, Q); // Q is obtained listing VT
5      N_R := ∅; // N_R is the set of new rules
6      foreach q ∈ S do
7          N_R := N_R ∪ {r(q)}; // Generate a new authorization rule r
               from q
8      if N_R is not empty then
9          N_R := GenerelizationForPolicy(N_R);
10 P := P ∪ N_R;
11 return P ;
```

7.3 Generalization for a policy

Algorithm 4 deals with query cancellation for the whole policy. We denote by $\mathcal{P}$ the policy (i.e., the set of rules). We denote by N_R the set of new rules that has been generated. The new policy set ($\mathcal{P}$) will be the union of $\mathcal{P}$ and N_R ($\mathcal{P} = \mathcal{P} \cup N_R$). A new rule could generate other new rules and so on until no rule is added. Let N_S be the set of attributes of the mediator schema. Since N_S is finite then the maximum number N_r of rules that could be defined is also finite. Let $N_\mathcal{P}$ be the number of rules in $\mathcal{P}$. Let n be the difference between N_r and $N_\mathcal{P}$. At each recursive call of the algorithm either no rule has been generated or n decreases since N_r increases. Thus, the algorithm terminates.

8 Experiments

We have conducted a number of experiments on real and synthetic datasets to validate each of the steps of our methodology. With synthetic datasets we generated particular configurations (e.g. worst-case scenarios) while with the real datasets (downloaded from the UCI ML Repository [6]) we first extracted $\mathcal{FD}$ by using a well-known algorithm called TANE [20] and then we run our algorithms with sets of rules having different number of attributes (from 2 to 10). We also tested the algorithms on specific subsets of $\mathcal{FD}$ (i.e., 100 and 200 extracted from the Bank dataset) that were not present in real datasets (Sub 1 and 2 in Table 1). The source code of the algorithms is released under GPL v3 free software licence and is available at the following address [18].

Table 1. Features data sets together with results of the experiments

Dataset desc.		Identified $\mathcal{FD}$		Performed experiments and results					
Name	$\|S\|$	$\|\mathcal{FD}\|$	$\mathcal{FD}_l$	$\|\mathcal{G}(V)\|$	$\|\mathcal{G}(E)\|$	$BuildG$	$\|\mathcal{VT}\|$	$FindVT$	$\|P'\|$
Yeast	8	10	3.88	6	10	5	5	4	7
Chess	20	22	9.14	21	20	3	20	14	21
Breast W.	11	37	4.13	41	165	26	37	65	20
Abalone	8	44	3.79	87	835	60	17	42	23
Sub 1	17	100	4.41	217	1312	193	130	197	54
Sub 2	17	200	4.92	453	8152	1502	1737	16596	263
Bank	17	433	6.47	14788	879241	3826	9137	335607	513

The reports about measures performed on each dataset shown in Table 1 are as follows:

1. **Detection phase:** FD_l is the average number of attributes that appear in $\mathcal{FD}$, $|\mathcal{G}(V)|$ is the number of nodes and $|\mathcal{G}(E)|$ is the number of edges of the generated graph, $BuildG$ is the time in *ms* to build $\mathcal{G}$, $|\mathcal{VT}|$ is the number of generated $\mathcal{VT}$ and $FindVT$ is the time in *ms* to construct $\mathcal{VT}$.
2. **Reconfiguration phase:** $|P'|$ is the number of rules that need to be added to the policy in order to forbid the completion of any transaction in $\mathcal{VT}$.

For each of the tests reported in Table 1 we calculated the mean value for 100 different executions generating rules with a number of attributes ranging from 2 to 10.

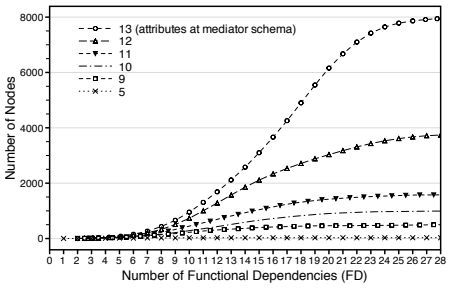

Fig. 3. Number of nodes with respect to number of FDs

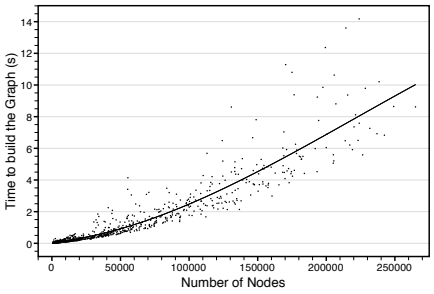

Fig. 4. Time to build the graph with respect to the number of nodes

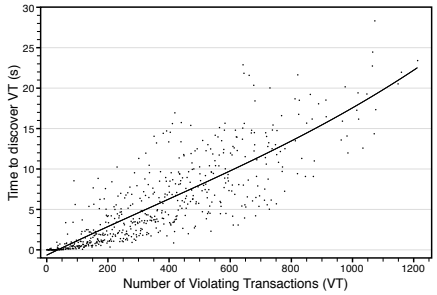

Fig. 5. Number of VTs and time to identify them

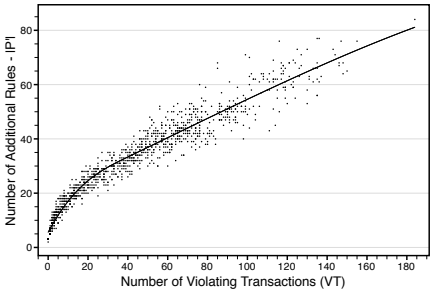

Fig. 6. Number of VTs and number of added rules

While Table 1 reports on the approach practicability on real datasets, the graphs in Figures 3, 4, 5 and 6 show tests performed on synthetic datasets. Also in this case we run multiple tests while varying parameters that are not subject to the evaluation. In particular, Figure 3 shows the relation between the number of nodes and the cardinality of randomly generated $\mathcal{FD}$. We report different tests while varying the number of attributes at the mediator schema. The tests show that by increasing the cardinality of $\mathcal{FD}$, the number of nodes increases very fast until, at a certain point, it starts slowing and approaching its upper bound as expected theoretically. Figure 4 shows the relation between the number of nodes and the time needed for building $\mathcal{G}$ with fixed attributes in the mediator schema. As we can see, the time to build $\mathcal{G}$ increases proportionally with respect to the number of nodes. This is mainly because we use binary trees to manage the nodes. The dots in figures represent single executions while the line has been generated using the Spline algorithm [19]. Figure 5 reports the performances on identifying $\mathcal{VT}$ from previously built graphs. The time grows proportionally

with respect to the number of transactions. With the discovered $\mathcal{VT}$ we extract the additional rules by applying Algorithm 4 to forbid transaction completion. Figure 6 shows the relation between the number of transactions and the number of additional rules that are extracted. In particular, at each cycle, we pick as decision metric the new rule that appear more often in $\mathcal{VT}$. We observe that the more FDs are discovered the more rules need to be added. This is due to the fact that more FDs induce more alternatives to policy violations.

The experiments show the practicability of our methodology on different datasets with different characteristics. The approach showed some limitation only when the cardinality of $\mathcal{FD}$ becomes very large (e.g., greater than 1500 for a single relation) being not able to discover transactions in an acceptable amount of time. We believe that this amount of FDs does not represent a typical scenario. Nevertheless, we will further investigate such situations.

9 Conclusions

In this work we have investigated the problem of illicit inferences that result from combining semantic constraints with authorized information showing that these inferences could lead to policy violations. To deal with this issue, we proposed an approach to detect the possible violating transactions. Each violating transaction expresses one way to violate an authorization rule. Once the violating transactions are identified, we proposed an approach to repair the policy. This approach aims at adding a minimal set of rules to the policy such that no transaction could be completed.

As future work we are extending this approach to partial FDs (i.e., the FDs that are not satisfied by all tuples but can lead to policy violations). We also plan to investigate other kinds of semantic constraints such as inclusion dependencies and multivalued dependencies. Finally, we could consider other integration approaches such as LAV and GLAV where same issues can arise.

References

1. S. Abiteboul, R. Hull, and V. Vianu. *Foundations of databases*, volume 8. Addison-Wesley Reading, 1995.
2. G. Aggarwal, M. Bawa, P. Ganesan, H. Garcia-Molina, K. Kenthapadi, R. Motwani, U. Srivastava, D. Thomas, and Y. Xu. Two can keep a secret: A distributed architecture for secure database services. *CIDR 2005*, 2005.
3. R. Agrawal, J. Kiernan, R. Srikant, and Y. Xu. Hippocratic databases. In *VLDB '02*, pages 143–154, 2002.
4. G. Armellin, D. Betti, F. Casati, A. Chiasera, G. Martinez, and J. Stevovic. Privacy preserving event driven integration for interoperating social and health systems. In *Proc. of the 7th VLDB conf. on Secure Data Management*, pages 54–69, 2010.
5. W. W. Armstrong. Dependency structures of data base relationships. In *IFIP Congress'74*, pages 580–583, 1974.
6. K. Bache and M. Lichman. UCI machine learning repository, 2013.

7. E. Bertino, S. Jajodia, and P. Samarati. Supporting multiple access control policies in database systems. In *IEEE Symp. on Security and Privacy*, pages 94–107, 1996.
8. A. A. Bertossi. Dominating sets for split and bipartite graphs. *Inf. Process. Lett.*, 19(1):37–40, Sept. 1984.
9. A. Brodsky, C. Farkas, and S. Jajodia. Secure databases: Constraints, inference channels, and monitoring disclosures. *TKDE '00*, 12(6):900–919, 2000.
10. B.-C. Chen, K. LeFevre, and R. Ramakrishnan. Privacy skyline: privacy with multidimensional adversarial knowledge. VLDB '07, pages 770–781, 2007.
11. V. Ciriani, S. D. C. Di Vimercati, S. Foresti, S. Jajodia, S. Paraboschi, and P. Samarati. Keep a few: Outsourcing data while maintaining confidentiality. In *ESORICS 2009*, pages 440–455. Springer, 2009.
12. S. De Capitani di Vimercati, S. Foresti, S. Jajodia, S. Paraboschi, and P. Samarati. Assessing query privileges via safe and efficient permission composition. CCS '08, pages 311–322, 2008.
13. H. S. Delugach and T. H. Hinke. Wizard: A database inference analysis and detection system. *IEEE Trans. on Knowl. and Data Engineering*, 8(1):56–66, 1996.
14. S. D. C. di Vimercati and P. Samarati. Authorization specification and enforcement in federated database systems. *J. Comput. Secur.*, 5(2):155–188, Mar. 1997.
15. C. Farkas and S. Jajodia. The inference problem: a survey. *ACM SIGKDD Explorations Newsletter*, 4(2):6–11, 2002.
16. B. C. M. Fung, K. Wang, R. Chen, and P. S. Yu. Privacy-preserving data publishing: A survey of recent developments. *ACM Comput. Surv.*, 42(4):14:1–14:53, June 2010.
17. M. Haddad, M.-S. Hacid, and R. Laurini. Data integration in presence of authorization policies. In *IEEE 11th Int. Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*, pages 92–99, 2012.
18. M. Haddad, J. Stevovic, A. Chiasera, Y. Velegrakis, and M.-S. Hacid. Access control for data integration project homepage, http://disi.unitn.it/%7estevovic/acfordi.html, 2013.
19. T. J. Hastie and R. J. Tibshirani. *Generalized additive models*. London: Chapman & Hall, 1990.
20. Y. Huhtala, J. Kärkkäinen, P. Porkka, and H. Toivonen. Tane: An efficient algorithm for discovering functional and approximate dependencies. *The Computer Journal*, 42(2):100–111, 1999.
21. M. Le, K. Kant, and S. Jajodia. Cooperative data access in multi-cloud environments. In *DBSec 2011*, volume 6818, pages 14–28. 2011.
22. M. Lenzerini. Data integration: A theoretical perspective. In *Proc. of the Symp. on Principles of Database Systems*, pages 233–246, 2002.
23. M. Levene and G. Loizou. *A guided tour of relational databases and beyond*. Springer Verlag, 1999.
24. D. J. Martin, D. Kifer, A. Machanavajjhala, J. Gehrke, and J. Y. Halpern. Worst-case background knowledge in privacy. In *ICDE*, pages 126–135, 2007.
25. A. P. Sheth and J. A. Larson. Federated database systems for managing distributed, heterogeneous, and autonomous databases. *ACM Comput. Surv.*, 22(3):183–236, Sept. 1990.
26. T.-A. Su and G. Özsoyoglu. Data dependencies and inference control in multilevel relational database systems. In *IEEE S. on Sec. and Privacy*, pages 202–211, 1987.
27. M. Thuraisingham. Security checking in relational database management systems augmented with inference engines. *Computers & Security*, 6(6):479–492, 1987.
28. H. W. Wang and R. Liu. Privacy-preserving publishing data with full functional dependencies. DASFAA'10, pages 176–183. Springer-Verlag, 2010.