Rule Configuration Checking in Secure Cooperative Data Access

Meixing Le, Krishna Kant, Sushil Jajodia
Center for Secure Information Systems
George Mason University, Fairfax, VA
{*mlep, kkant, jajodia*}*@gmu.edu*

***Abstract*—In this paper, we consider an environment where a group of parties have their own relational databases and provide restricted access to other parties. In order to implement desired business services. each party defines a set of authorization rules over the join of basic relations, and these rules can be viewed as the configurations of the accessible information in the cooperative data access environment. However, authorization rules are likely to be developed by each enterprise somewhat independently based on their business needs and may not be sufficiently well defined to be enforceable. That is, the rules may be missing some crucial access capabilities that are essential for implementing the desired restrictions. In this paper, we propose a mechanism to check the rule enforceability for each given authorization rule. For the partially enforceable rules, we also present an algorithm to modify the rules so as to make them totally enforceable.**

I. Introduction

Providing rich services to clients with minimal manual intervention or paper documents requires the enterprises involved in the service path to collaborate and share data in an orderly manner. For instance, an automated determination of patient coverage and costs requires that a hospital and insurance company be able to make certain queries against each others' databases. Similarly, to arrange for automated shipping of merchandise and to enable automated status checking, the e-commerce vendor and shipping company should be able to exchange relevant information, perhaps in form of database queries. In such environments, data must be released only in a controlled way among cooperative parties, subject to the authorization policies established by them. The authorization policies are the configurations for the accessible information in such an environment. In this paper, we expose and study the accuracy and enforceability of such configurations in a collaboration setting.

In general, enterprise data may appear in a variety of forms, including the simplistic key-value forms like Google's BigTable. However, for concreteness, we assume that all data is stored in relational form, with all tables in a standard form such as BCNF. In such a model, data access privileges are given by a set of authorization rules, each of which is defined either on original tables belonging to an enterprise or over the lossless join of two or more of these. The join operations, coupled with appropriate projection and selection operations define the access restrictions; although in order to enable working with only the schemas, we do not consider selection operation. The access rules must be enforceable, and each incoming query should be efficiently verifiable against them.

Although the problem is rather straightforward, there are many hurdles in properly specifying and implementing the access rules. First, since the enterprises are allowed to specify an arbitrary set of access rules, it is possible that there is no way to derive a safe execution plan for certain rules. The simplest way to illustrate this problem is by considering the following: situation: a rule specifies access to $R \bowtie S$ (where R and S are relations owned by two different parties); however, no party has access to R and S individually and thus no party is able to do the join operation! Thus, a basic problem is to determine enforceability of the given rules. If a rule is not enforceable, we should either remove it or make it enforceable. If not, this will cause problems for the queries. For instance, a query for the information of $R \bowtie S$ is authorized by the rule configuration, but cannot be properly answered.

We address the configuration checking problem in two steps. First, we examine the enforceability of each authorization rule in a constructive bottom-up manner, and build a graph structure that captures the relationships among the rules. In a collaborative environment, a rule can be enforced with not only the locally available information but also the remote information from the cooperative parties. If a rule is not totally enforceable, we consider two ways to deal with it. The first option is to remove the unenforceable part of the rule, so that only enforceable rules are retained. The second option is to modify related existing rules to make the inspected rule totally enforceable. We use the property of the graph to find the solution that has minimal impact on the existing rules.

The rest of the paper is organized as follows. Section II briefly discusses the related work. Section III defines the problem of cooperative access formally, introduces a number of definitions and concepts, which are illustrated via a running example. Section IV discusses the mechanism to check rule enforceability. Section V describes the algorithm adding more privileges to make rules totally enforceable. Finally, Section VI concludes the discussion.

II. Related Work

The problem of controlled data release among collaborating parties has been studied in [9]. The basic model in this paper is identical to ours and provides the motivation for our work. Its main contribution is an algorithm to check if a query with

a given query plan tree can be safely executed. However, it does not address the problem of rule enforceability. Without a trusted third party, the unenforceable rules are inaccurate configurations and need to be revised, and we address that in this work. In another work [8], the same authors evaluate whether the information release the query entails is allowed by all the authorization rules given to a particular user, which considers the possible combinations of rules and assumes that the rules are defined in an implicit way. In our work, we assume authorizations are explicitly given, and data release is prohibited if there is no explicit authorization. As they focus on the problem of query authorization, we emphasize the retrievability of the authorized queries.

Processing distributed queries under protection requirements has been studied in [6], [14]. In these works, data access is constrained by a limited access pattern called binding pattern, and the goal is to identify the classes of queries that a given set of access patterns can support. We have a very different authorization model involves independent parties who may cooperate in the execution of a query. There are also classical works on distributed query processing [5], [12], but they do not deal with constraints made by the data owners.

Answering queries using views [11], [10], [15] has been extensively studied, and the technique is useful for query optimization, data integration and so on. The given view definitions is these works can be similar constraints to our authorization rules. However, these works usually consider the queries and views in the form of conjunctive queries, and they do not consider the collaboration relationships among different parties. These make our problem different from their works.

In the area of outsourced database services, some works [1], [7] discuss how to secure the data in such environments, and there are also services like Sovereign joins [2]. It gets encrypted relations from the participating data providers, and sends the encrypted results to the recipients. These methods are useful to enforce our authorization rules, but we discuss the problem without any involvement of third parties.

The given authorization rules is also similar to the firewall rules, which indicates what types of queries can go through. As firewall rules are need to be enforceable and accurate [4], [16], we have the same requirements in our situation.

III. Problem and Definitions

We consider a group of cooperating parties, each of which maintains its data in a standard relational form such as BCNF. (It is possible to consider more complex normal forms than BCNF, but we do not consider them here.) We assume simple select-project-join queries (e.g., no cyclic join schemas or queries). The query may be answered by any of the parties that has the required permissions. We assume that the join schema is given – i.e., all the possible join attributes between relations are known. Each join in the schema is lossless so a join attribute is always a key attribute of some relations. We assume the rules to be "**upwards closed"** which is the same as [9]. That is, if two rules expressly grant permission to access two different relations, say R and S, then there also exists a rule providing access to their join result $R \bowtie S$. We study the problems only involving existing cooperative parties, without any third parties.

The basic problems considered here are as follows: Given a set of authorization rules R on N cooperating parties, (a) identifies the subset of R that can be enforced, and determines the maximal portion of the rules that can be enforced. (b) deals with the unenforceable portion of the rules, and make the resulting rule set R' all enforceable.

A. A Running Example

Our running example for illustration models an e-commerce scenario with four parties: (a) *E-commerce*, denoted as E, is a company that sells products online, (b) *Customer_Service*, denoted C, that provides customer services (potentially for more than one Company), (c) *Shipping*, denoted S, provides shipping services (again, potentially to multiple companies), and (d) *Warehouse*, denoted W, is the party that provides storage services. To keep the example simple, we assume that each party owns but one relation described as follows:

1) E-commerce (order_id, product_id, total) as E
2) Customer_Service (order_id, issue, assistant) as C
3) Shipping (order_id, address, delivery_type) as S
4) Warehouse (product_id, location) as W

In the following, we use oid to denote $order\_id$ for short, pid stands for $product\_id$, and $delivery$ stands for $delivery\_type$. The possible join schema is also given in figure 1. Relations E, C, S can join over their common attribute oid; relation E can join with W over the attribute pid. In the example, relations are in BCNF, and the only FD (Functional Dependency) in each relation is the underlined key attribute determines the non-key attributes.

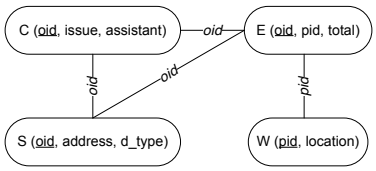

Fig. 1. The given join schema for the example

B. Authorization model and definitions

Following the definition of authorization model in [9], an **authorization rule** r_t is a triple$[A_t, J_t, P_t]$, where J_t is called the join path of the rule, A_t is the authorized attribute set, and P_t is the party authorized to access the data.

Definition 1: A **join path** is the result of a series of join operations over a set of relations $R_1, R_2 ... R_n$ with the specified equi-join predicates $(A_{l1}, A_{r1}), (A_{l2}, A_{r2}) ... (A_{ln}, A_{rn})$ among them, where (A_{li}, A_{ri}) are the join attributes from two relations. We use the notation J_t to indicate the join path of rule r_t. We use JR_t to indicate the set of relations in a join path J_t. The **length** of a join path is the cardinality of JR_t.

We can consider a join path as the result of join operations without limitations on the attributes. Thus, A_t is the set of attributes projection on the join path that is authorized to be

Rule No.	Authorized attribute set	Join Path	Party
1	{pid, location}	W	P_W
2	{oid, pid}	E	P_W
3	{oid, pid, location}	$E \bowtie_{pid} W$	P_W
4	{oid, pid, total}	E	P_E
5	{oid, pid, total, issue}	$E \bowtie_{oid} C$	P_E
6	{oid, pid, total, issue, address}	$S \bowtie_{oid} E \bowtie_{oid} C$	P_E
7	{oid, pid, location, total, address}	$S \bowtie_{oid} E \bowtie_{pid} W$	P_E
8	{oid, pid, issue, assistant, total, address, delivery}	$S \bowtie_{oid} E \bowtie_{oid} C \bowtie_{pid} W$	P_E
9	{oid, address, delivery}	S	P_S
10	{oid, pid, total}	E	P_S
11	{oid, pid, total, address, delivery}	$E \bowtie_{oid} S$	P_S
12	{oid, pid, total, location}	$E \bowtie_{pid} W$	P_S
13	{oid, location, pid, total, address, delivery}	$S \bowtie_{oid} E \bowtie_{pid} W$	P_S
14	{oid, pid}	E	P_C
15	{oid, issue, assistant}	C	P_C
16	{oid, pid, issue, assistant}	$E \bowtie_{oid} C$	P_C
17	{oid, pid, issue, assistant, total, address, location}	$S \bowtie_{oid} C \bowtie_{oid} E \bowtie_{pid} W$	P_C

TABLE I
AUTHORIZATION RULES FOR E-COMMERCE COOPERATIVE DATA ACCESS

accessed by party P_t. Table I shows the set of rules given to these parties. The first column is the rule number, the second column gives the attribute set of the rules, join paths of the rules are shown in the third column, and the last column shows the authorized parties of the rules. We make one more assumption regarding the rules we are considering. We assume that each given authorization rule always includes *all of the key attributes of the relations that appear in the rule join path.* In other words, a rule has all the join attributes on its join path. We argue that this is a reasonable assumption as in many cases when the information is released, it is always released along with the key attributes.

When a query is given, it should be answered by one of the parties that have the authorization. Since our authorization model is based on attributes, any attribute appearing in the Selection predicate in an SQL query is treated as a Projection attribute. In other words, the authorization of a PSJ query is transformed into an equivalent Projection-Join query authorization. Therefore, a query q can be represented by a pair $[A_q, J_q]$, where A_q is the set of attributes appearing in the Selection and Projection predicates, and the query join path J_q is the FROM clause of an SQL query. For instance, there is an SQL query Q_1:

"Select $oid, total, address$ From E Join S On $E.oid = S.oid$ Where $delivery$ = 'ground'"

The query can be represented as the pair $[A_q, J_q]$, where A_q is the set $\{oid, total, address, delivery\}$; J_q is the join path $E \bowtie_{oid} S$.

In fact, each join path defines a new relation/view. To better understand the authorization relationships between the queries and the rules, we give the definition for join path equivalence.

Definition 2: Two join path J_i and J_j are equivalent, noted as $J_i \cong J_j$, if any tuple in J_i appears in J_j and any tuple in J_j appears in J_i.

For two join paths to be equivalent, a necessary condition is $JR_i = JR_j$. However, if several relations joins over the same attributes, then the join predicates among the join paths can be different, but they are still equivalent. To decide join path equivalence, we put join paths into join graphs. A **join graph** is a graph where each node indicates a relation, and each edge is the join attribute for the possible join between two nodes. The given join schema is an example of join graph. For a given join path, we also put its relations and join predicates into a graph. A valid join path is a spanning tree of a join graph, and two join paths are equivalent if they are both spanning trees of the same join graph. Figure 2 (a) shows the join graph of the given join schema, (b) is the graph representation of the join path of example rule r_8, and (c) is for the join path of rule r_{17}. Since figure 2 (b) and (c) are both spanning trees of (a), these two join paths are equivalent.

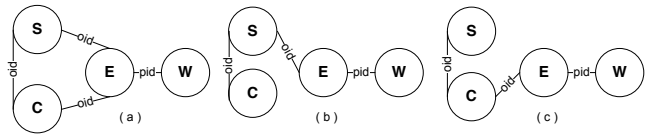

Fig. 2. Join path equivalence

C. Query authorization and rule enforcement

Authorization rules define the set of queries that are authorized to retrieve information from the parties. A query q is called **authorized** if there exists a rule r_t such that $J_t \cong J_q$ and $A_q \subseteq A_t$. The join paths must be equivalent. Otherwise, the relation/view defined by the rule will have fewer or more tuples than the query asks for. Here we don't consider the situation where the projections on two different join paths get the same result (e.g., by joining on foreign keys) since data coming from different parties usually does not have foreign key constrains. For instance, the example query Q_1 is authorized by r_{11}, but it cannot be authorized by r_{13}. Although all the required attributes are authorized by r_{13}, the information regulated by such a rule usually authorizes fewer tuples than the correct answer of Q_1, since the pid attribute in relation W filters some tuples in the join results.

On the other hand, "authorized" is only a necessary condition for a query to be answered but not sufficient. To actually answer a query, we need at least one query execution plan.

A **query execution plan** or "query plan" for short, includes several ordered steps of operations over authorized and obtainable information and provides the composed results to a party. The result of a query execution plan pl is also relational, and it can also be presented with the triple $[A_{pl}, J_{pl}, P_{pl}]$. A valid query plan should be authorized by a given authorization rule r_t. Therefore, a plan pl answers a query q, if $J_{pl} \cong J_q \cong J_t$, $A_q = A_{pl} \subseteq A_t$ and $P_{pl} = P_t$. An authorization rule defines the maximal set of attributes that a query on the equivalent join path can retrieve. Therefore, a rule can also be viewed as a query. We call the query plan to enforce a rule as an **enforcement plan** or "plan" for short below.

Definition 3: A rule r_t can be *totally enforced*, if there exists a plan pl such that $J_t \cong J_{pl}$, $A_t = A_{pl}$, $P_t = P_{pl}$. r_t is *partially enforceable*, if it is not totally enforceable and there is a plan pl that $J_t \cong J_{pl}$, $A_t \supset A_{pl}$, $P_t = P_{pl}$. Otherwise, r_t is not enforceable. A join path J_t is enforceable if there is a plan pl that $J_t \cong J_{pl}$.

At the very beginning, only the rules indicating the data owners have their own data are known to be totally enforceable. As a plan contains steps bringing information together to enforce a rule, an enforcement plan can have following 3 operations over the enforceable information: A projection (π) is performed on a single party to select attributes; A join ($\bowtie$) operation is also performed at a single party, and it combines two pieces of information and generates information on a longer join path; Data transmission ($\rightarrow$) is an operation that happens between two parties, and one party sends information to the other. It is required that the two parties have two rules on the equivalent join paths and the information transmitted is based on such join path. In addition, the rule on the receiving party should have an attribute set that includes all the attributes of the information being transmitted. Otherwise, the transmission is not safe. For instance, P_S can send information on r_{13} to P_E as $J_{13} \cong J_7$, but it cannot send attribute $delivery$ to P_E. Such information cannot be sent to P_C because P_C does not have rule on join path equivalent to r_{13}.

It is obvious that a rule defined on basic relation is totally enforceable as a data transmission operation will give the enforcement plan. However, a rule with longer join path is not always enforceable. Whether an enforcement plan exists depends on whether pieces of enforceable information on shorter join paths are available at the same party and whether they can be joined losslessly. Besides, the enforceable information on remote cooperative parties may also be helpful to construct an enforcement plan. Overall, in the cooperative scenario, a rule can have many enforcement plans.

IV. Checking rule enforcement

In this section, we first introduce some concepts and results, then we present the algorithm that works from bottom-up to check the enforceability of each given authorization rules.

A. Key attributes hierarchy

Given the BCNF form, there is only one possible lossless join between any two relations. It is known that for a lossless join, the join attribute in at least one of the two joining relations must be a key attribute [3]. Therefore, for any pair of joins, the key attribute of one relation is also the key of the join result. For instance, relations W and E can do a lossless join over the attribute pid, and pid is the key attribute of W. Let $K(X)$ denote the key attributes of the relation X, and $\stackrel{*}{\rightarrow}$ the functional dependency. Then, $K(E) \stackrel{*}{\rightarrow} K(W)$ for the resulting relation, and $K(E)$ becomes the key for it. We call this situation as *key hierarchy*, since the key simply extends over the joined relation. In some cases such as relations E and S, they join over oid, which is the key for both relations. Since $K(E) = K(S)$, there is no hierarchy. The key attributes from basic relations in a join path also form a hierarchal structure. Otherwise, there will be a relation in the join path that is determined by two or more different upper level key attributes. In such case, relations with upper level keys join with each other over the non-key attribute, and this creates a forbidden graph [3], so the join path is lossy. Based on these conditions, there always exists a key attribute from one basic relation that is also the key attribute of a join path. We call this attribute as the key attribute of the join path (or the key of the rules defined on such join path). If the join result of two join paths forms a valid longer join path, the join operation is always lossless. We call a plan as **joinable plan** if such a plan contains all the key attributes of the basic relations in its join path.

Lemma 1: If a join path J_t is enforceable, there exists a joinable plan pl that $J_t \cong J_{pl}$.

Proof: As we assume all the rule definitions contain the key attributes of the relations in its join path, these attributes are always authorized in data transmission operations. All the plans start from the rules on basic relations which are totally enforceable, and a longer join path is enforced by join operations over plans on shorter join paths. Therefore, if there is a plan for join path J_t, there always exist one plan that never project out any of these key attributes through the different operations in the entire plan, and such plan is joinable. ■

In some cases, a rule does not have a total enforcement plan, but only some partial plans. A partial plan only enforces a rule with an attribute set that is a proper subset of the rule attribute set. We say that an attribute set is a **maximal enforceable attribute set** for a rule, if it is enforced by a plan of the rule, and there is no other plan of the same rule that can enforce a superset of these attributes. If a rule is totally enforceable, its maximal enforceable attribute set is the rule attribute set, and we have the following lemma.

Lemma 2: A rule has only one maximal enforceable attribute set.

Proof: Firstly, a totally enforceable rule only has one maximal enforceable attribute set. Thus, a rule defined on basic relation only has one such set. To get the maximal attribute set, we do not eliminate any attributes via projections of plans, and maximal information is exchanged in data transmission operations, and such plans are always joinable. Therefore, if a rule is not totally enforceable, even it has several partial plans, these joinable plans are on the same join path and can always be merged by joining over the key attributes of the join path. Consquently, a partially enforceable rule have one maximal enforceable attribute set. At last, if a rule is not enforceable, its enforceable attribute set is empty. ■

B. Enforcement checking mechanism

As discussed above, it is desired to have a mechanism to check the rule enforceability for given set of rules. Such a mechanism can tell which rules can be enforced and what are their maximal enforceable attribute sets, and that also gives the answer of what are the set of authorized queries that can be safely answered according to the given configurations of the rules.

We have two options with the given rules that are not enforceable. The first choice is that we keep only the found enforceable rules with their maximal enforceable attribute sets, and rules that are not enforceable as well as the unenforceable attributes are removed from the rule configurations. In other words, the algorithm finds all the information that can be safely retrieved according to the given set of rules, and all inaccurate and unenforceable configurations are removed. This solution can be thought as a conservative one since it prohibits some authorized information to be released because of the enforceability. In contrast, we can also modify the rule configurations in an aggressive way. In such scenario, we think all the information regulated by the rules are authorized, and authorized information should be retrievable. Therefore, whenever any information in the defined rules cannot be enforced, we change the rule configurations by granting more privileges so as to make these information enforceable. However, this releases more information, and such situation may not be desired by the cooperative parties. As there are different ways to modify the rules, we prefer to find the way that has minimum impact on the existing rules. That is, we try to find the minimum amount of information to release.

To that end, we first propose a constructive mechanism that checks the rules in a bottom-up manner. In general, an enforcement plan for a rule combines pieces of information available and generates the information authorized by the rule. For each rule, the mechanism checks its relevant information locally and remotely and indicates if it can be enforced and what is its maximal enforceable attribute set. The set of unenforceable attributes and the unenforceable rules are removed from the rule set. Since this algorithm works as the first option we discussed above, we also provide the algorithm for the second option. Therefore, whenever the inspected rule cannot be totally enforced, we modify rules to make it enforceable. We discuss the possible ways to define minimality and what is the minimal impact on the current set of rules. We present the first algorithm below, and then we describe the second algorithm. In fact, the second algorithm just introduces extra steps into the first one when inspecting a given rule, and all the other steps are the same. Thus, we discuss only these additional steps for the second algorithm.

C. Finding enforceable information

When examining a rule $[A_t, J_t, P_t]$, we call such a rule r_t as *Target Rule*, the attribute set A_t as *Target Set*, the join path J_t as *Target Join Path*, and the party P_t in the rule as *Target Party*. All the other parties are *Remote Parties*. To check the enforceability of r_t, we first find the relevant information that can be obtained locally at P_t. If this is not enough, we check the information from remote parties.

Rules can be enforced by performing consistent operations over the information that is already enforceable. In addition, it is always the case that information from short join paths is put together to enforce a rule of longer join path. Therefore, we propose the algorithm to work in a bottom-up way in the order of join path length, and it begins with rules on basic relations (length of 1 rules). As the mechanism works bottom-up, when examining a target rule with join path of length n, we can assume that all the rules on join paths with shorter lengths have already been examined, and only the maximal enforceable attribute sets of the rules are preserved.

Since the first task is to identify relevant information locally, we check the rules relevant to r_t at P_t. We call a join path as a **Sub-Join Path** of J_t if it is a join path which contains a proper set of relations of JR_t. Rules not on the sub-join paths are not relevant to r_t since any composition with these rules will contain information more than what r_t authorizes. At party P_t, a joinable plan that is on a sub-join path of J_t is a **Relevant Plan**, and a rule defined on a sub-join path of J_t is a **Relevant Rule** to the target rule. Parties that have rules defined on the equivalent join path of J_t are called J_t**-cooperative parties**, and information on J_t is allowed to be exchanged only among these parties by data transmission operations. For instance, P_E and P_S are J_{13}-cooperative parties since $J_{13} \cong J_7$. We assume that each inspected rule is represented by an enforcement plan. When inspecting the target rule, we consider using these plans to enforce it. We say "join among rules" below, which means their enforcement plans.

The checking process iterates in the order of the join path lengths beginning with the rules defined on the basic relations on various parties. These rules can be totally enforced as the data owners sending their data to the authorized parties. From then on, the algorithm checks for rules defined on longer join paths. At the same time examining the rules, the algorithm also builds a graph structure. Each node in such structure is a rule with its maximal enforceable attribute set. The nodes in the graph are put in different levels based on their join path lengths. Two nodes on the same party are connected if one is the relevant rule of the other. Among different parties, nodes can be connected if they have the equivalent join paths. Such a structure captures the relevance and cooperation relationships among the enforceable rules. Figure 3 shows the built structure for our running example. The different parties are separated vertically. The bold boxes show the basic relations owned by different parties. The algorithm starts the iteration with the rules on basic relations r_1, r_2, r_4, r_{10}, and so on.

As the algorithm iteratively checks all the rules, when a target rule r_t is examined, the algorithm first checks whether the join path J_t can be enforced using relevant rules on P_t. After that, all the rules with equivalent join path of J_t are checked respectively at J_t-cooperative parties. Then the algorithm checks the possible enforcement by exchanging information among these parties. In figure 3, on the level of join path length 2, the algorithm checks the rules with the order of r_3, r_{12}, r_5, r_{16}, r_{11} because $J_3 \cong J_{12}$ and $J_5 \cong J_{16}$. J_t-cooperative parties such as P_W and P_S on J_3 will check

the remote enforcement between r_3 and r_{12}, which will be described later.

To check local enforceability, the algorithm finds its local relevant rules in the currently built graph structure since all its relevant rules have already been examined and added to the graph. It only checks with the top level relevant rules in the current graph, where top level rules are the nodes not connected to any higher level nodes (rules with longer join paths) in the currently built graph during the bottom-up procedure. For example, in figure 3, when the algorithm examines r_{13} on P_S, only r_{11}, r_{12} are top level rules. And when checking r_8, r_7 and r_5 are top level rules since r_6 is not enforceable. Here, we take advantage of the upwards closed property of the rules, so that the top level rules cover all possible join results among the lower level rules. If these top level rules cannot be composed to enforce the J_t, there is no need to check lower level rules. When examining r_{13}, there is no need to consider the join between r_9 and r_{10}. Among the rules in the graph on P_t, a relevant rule r_r of r_t can be efficiently decided, if $JR_r \subset JR_t$.

The following step is to check whether the join path J_t can be enforced locally by performing joins among these top level relevant rules. The algorithm basically checks each pair of these rules. We check it pairwise because if a pair of them can join, the result must be able to enforce J_t. Otherwise, there must exist another relevant rule of r_t authorizing the join result, and such a rule is on higher level of the pair of rules being inspected, which is contradict to the fact that the pair of rules are top level rules. When checking whether a pair of rules (r_s, r_r) can join, the algorithm first tests their relation sets to see if $JR_s \bigcap JR_r = \emptyset$. If these two join paths have overlapped relations, they can join over the key attribute of the overlap part, and J_t can always be enforced. Otherwise, we need to further check the attributes of two rules to see if they have the required join attribute in common. If J_t can be locally enforced, we mark the target rule as **local enforceable** rule and add it to the graph by connecting it with top level relevant rules. Otherwise, it has to wait and see if J_t can be enforced on other parties. For instance, when checking r_3 in our example, it has top level relevant rules r_1 and r_2, since there is no overlapped relation for the pair of rules, the algorithm checks whether join attribute pid can be found in both rules. On the other hand, when checking the pair r_{11} and r_{12}, as E is the overlapped relation, the join path J_{13} can be locally enforced. r_{17} does not has a valid join pair, and it is not locally enforceable. Once a pair is found to enforce the join path, the algorithm proceeds to next steps.

Meanwhile, the algorithm also computes the union of the attributes from top level relevant rules regardless of the enforceability of J_t. The resulting attribute set A_r includes all attributes that can be obtained from party P_t if J_t can be enforced. It is always the case that $A_r \subseteq A_t$ as rules are upwards closed. If A_r not equals to A_t, we call the set of attributes $A_t \setminus A_r$ as **missing attribute set** A_m. The attributes in A_m are potentially obtainable from the J_t-cooperative parties. In the example, the attribute $delivery$ in r_8 cannot be found in its top level rules r_7 and r_5, and it is a missing attribute after the local checking.

Algorithm 1 Rule Enforcement Checking Algorithm

Require: All given authorization rule set R on all parties
Ensure: Find enforceable rules and build graph
Mark rules with length 1 as total enforceable rules
Get the maximal length of join path length N
for Join path of length 2 to N **do**
  for Each join path J_t length equal to i **do**
    $AJ_t \leftarrow \emptyset$, the set of shared attributes on J_t
    for Each party P_t has a rule r_t on J_t **do**
      Obtain the set of top level relevant rules R_v
      Add the node and connections to R_v in graph
      $A_v \leftarrow$ the union of attributes in R_v
      Missing attribute set $A_m \leftarrow A_t$
      for Each pair of relevant rule (r_s, r_r) **do**
        if The pair can locally enforce J_t **then**
          $A_m \leftarrow A_m \setminus A_v$ and break
      if $A_m \neq \emptyset$ **then**
        Put r_t with A_m into the Queue of J_t
      $AJ_t \leftarrow AJ_t \bigcup A_v$
    for Each rule r_t in the Queue of J_t **do**
      if J_t can be enforced on some party **then**
        Add connections among J_t-cooperative parties in graph
        $A_m \leftarrow A_m \setminus AJ_t$
        if $A_m \neq \emptyset$ **then**
          Replace A_t with $A_t \setminus A_m$ in graph
      else
        r_t cannot be enforced, remove rules on J_t from graph
  Join path length i++

Next, the algorithm checks the remote information that a party can use to enforce a rule, and only J_t-cooperative parties are checked. As the previous steps of the algorithm tell which parties can locally enforce the join path J_t, if there exists any party that can enforce J_t, then all the J_t-cooperative parties can have joinable plans for their rules on J_t. Thus, the party P_t is able to get attributes from all its J_t-cooperative parties to enforce r_t. For instance, r_{17} is not locally enforceable, but J_8 can be enforced with a joinable plan at P_E. Thus, we can add a data transmission operation to such plan, and r_{17} also has a joinable plan. This plan can join with r_{16}, so that attributes $issue, assistant$ in r_{17} can be enforced. Consquently, these attributes in r_8 can also be enforced. Therefore, we take the union of the attribute sets from all J_t-cooperative parties to check if r_t can be totally enforced. If the missing attribute set $A_m \subset A_{r_1} \bigcup A_{r_2} ... A_{r_k}$ (where A_{r_i} is the relevant attribute set of a J_t-cooperative party P_i), then r_t can be totally enforced. Otherwise, A_m is updated by removing the attributes appear in any A_{r_i}. In such case, r_t has a maximal enforceable attribute set on J_t without the attributes in A_m. The node r_t in the graph structure is presented with the attribute set $A_t \setminus A_m$. Meanwhile, **connection edges** are added among the J_t-cooperative rules in the graph structure. For example, attribute $delivery$ of r_8 also cannot be found in its J_t-cooperative party P_C, so it cannot be enforced. r_8 in the graph is represented with the attribute set without $delivery$. We use bold font in figure 3 to indicate this attribute is not enforceable. Also, since join path J_6 cannot be enforced at any party, r_6 is not enforceable, and it will not be included in the graph structure. In figure 3, we use the dashed box to show r_6 is removed. The local enforceable rules are marked with "L". The detailed algorithm is described in Algorithm 1.

In algorithm 1, each rule will be examined at most twice, with one local enforceability check and another one in check-

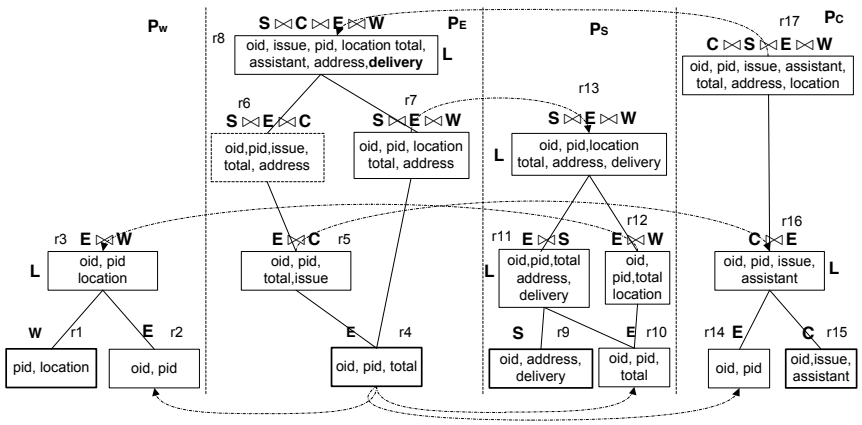

Fig. 3. Graph structure built for the example

ing the queue of J_t. In the step of local enforcement checking, only the top level relevant rules on party P_t are checked. Suppose that the total number of rules is N_t, the maximal number of relevant rules of a rule is N_o, and checking join condition takes constant C. Then the worst case complexity for algorithm 1 is $O(N_t * N_o^2 * C)$, where N_o is usually very small. In addition, this algorithm can be used as a pre-compute step once all the rules are given.

Theorem 1: The **Rule Enforcement Checking Algorithm** finds all enforceable information.

Proof: As all the information can be obtained on join results comes from the basic relations, the algorithm works in bottom-up manner to capture the operation results. If the join path of a rule cannot be enforced, then all the rules on this join path cannot be enforced and can be discarded. The algorithm first finds a way to enforce the join path of the rule r_t. The check on local relevant rules explores all possible ways to compose useful information on P_t. Since the only other information can be used to enforce r_t must come from J_t-cooperative parties, the algorithm also considers all the attributes that r_t can get from them. There is no other way to enforce more attribute for r_t. ■

V. Augmenting Authorizations for Enforceability

In this section, we discuss the mechanism which modifies the rule configuration by granting more privileges to enforce the unenforceable rules. We can add such steps into the previous algorithm after concluding that the inspected rule cannot be totally enforced. Such a rule could be either partially enforceable or its join path is not enforceable at all.

In the former case, according to Lemma 1, the missing attributes are all non-key attributes in the underlining basic relations. Therefore, there is no need to add new rules, and expanding the attribute sets of existing rules can make the rule totally enforceable. For the latter case, new rules must be added to make the join path enforceable. The problem of deciding new rules to add is extremely complex for a number of reasons. First, an addition could violate the "upwards closed" property that we are assuming, and thus run the algorithm to fix this. This topic is discussed in [13] and is beyond the scope of this paper. Second, an automated addition of a rule may be undesirable, and we may need to look for some manually assisted or guided process. Third, it would desired to define some notion of minimality in adding new rules which is nontrivial. In view of these challenges, we reserve this case for future work.

Thus, our goal is to grant more attributes to existing rules so as to turn partially enforceable rules into totally enforceable rules while maintaining minimal impact on the existing rules. Here we define the impact as the amount of new information granted to the parties, which can be measured in two slightly different ways as explained below.

We first consider the minimal impact in terms of the number of attributes being added to the different rules. Referring to the graph we built, we can turn the problem of minimal attribute addition into a shortest path problem with path length measured by the number of edges. We do it for each attribute M_i in the missing attribute set respectively. A shortest path from the target rule r_t to a rule with missing attributes M_i gives the minimal number of rules. For this, we perform a breadth first search starting from r_t where each visited node records its parent node. Once a rule has M_i is found, the breadth first search stops and the path between the two nodes is selected. For each node on the selected path, the attribute M_i is added to the corresponding rule.

Since the goal is to make the attribute M_i enforceable in r_t, the algorithm only checks the rules on the sub-paths of J_t and include the relation that M_i comes from. When r_t is inspected, it is at the top level of the graph, so the search is performed top-down. Each next visited node must have the join path length no longer than the current node. It is because that a plan has longer join path cannot be used in a plan for a shorter join path by valid operations, and the information of M_i cannot be transmitted from a higher level node to a lower one. All these selection conditions make the search more efficient than the general case. For instance, since rule r_8 has missing attribute $delivery$, the search begins from r_8. The search gives the shortest path r_8, r_7, r_{13}, and r_8 becomes totally enforceable by adding $delivery$ to r_7.

However, because of the upwards closed property of the rules, once M_i is released to a party on a rule with shortest

Algorithm 2 Rule Enforcement Algorithm with Minimal Parties

Require: A partially enforceable rule r_t with missing attribute set A_m
Ensure: r_t is totally enforced with modified rules R'

```
1: Get the graph structure G up to r_t
2: for Each M_i in A_m do
3:     Create a queue Q
4:     Enqueue r_t onto Q
5:     while Q is not empty do
6:         r_i ← Q.dequeue()
7:         if r_i has attribute M_i then
8:             if P_i is unvisited then
9:                 Mark P_i visited, record with r_i
10:        for Each edge e that includes r_i do
11:            r_n ← e.opposite(r_i)
12:            if r_n is unvisited && J_n is a sub-path of J_t &&
               JR_n ≤ JR_i && J_n includes the relation of M_i then
13:                Mark r_n visited, record r_i as its parent
14:                Enqueue r_n onto Q
15:    for Each visited party P_i except P_t do
16:        Get r_i associated with P_i
17:        Count the number of parties on path from r_i to r_t
18:        Keep the minimal party P_m
19:    Get r_m associated with P_m
20:    while r_m ≠ r_t do
21:        r_m ← parent(r_m)
22:        Add M_i to r_m
```

join path, all the rules that this rule is relevant to should also have M_i added to them. Since rules with longer join paths have not been examined by Alg. 1 yet, it is difficult to know the exact number of attributes that will ultimately be added. That is, the above algorithm ensures minimality only in terms of initial addition, not the ultimate one. It has the worst case complexity the same as breadth first search which is related to the number of nodes and edges.

On the other hand, we can think that once an attribute is granted to a party, this party will have the privilege to access such attribute. Therefore, no matter how many rules on this party are expanded with this attribute, we only count them as one attribute release. In this sense, our algorithm does accomplish its stated goal.

The detailed algorithm is described in Alg 2. Different from the previous algorithm which stops when a rule having the attribute M_i is found, we do the breadth first search for all the qualified rules. Whenever a new rule r_n is visited, the algorithm checks if the party P_n has been visited before. If this is the first time P_n being visited, the algorithm records the rule r_n associated with the party P_n indicating a shortest path from P_n to P_t is found. After the breath first search, the parties that have the associated rules are examined. For each found shortest path from P_n to P_t, the algorithm counts the number of different parties along such path. At last, the path with the minimal number of parties will be selected as the best way to modify the rules, and rules on this path are expanded with the attribute M_i. All these modified rules are recorded. Similar to the previous discussion, when Alg. 1 examines rules on longer join paths, if a modified rule is relevant to the rule being inspected, such a rule is also modified so as to include the missing attributes into its attribute set. The algorithm has the complexity of $O((|E|+|V|)*|A_m|)$, where $|E|$ is the number of the edges and $|V|$ is the number of nodes in the relevant subgraph, and $|A_m|$ is the number of missing attributes.

In the example, r_8 has the missing attribute $delivery$. After the breadth first search, only party P_S is visited, and the associated rule is r_{13}. Following the path from r_{13} to r_8, only r_7 need to be updated with the attribute $delivery$. The result is the same as the previous one.

VI. Conclusions and future work

In previous research work, a flexible data authorization model has been proposed to meet the security requirements for collaboration computing among different data owners in a distributed environment. Since authorization rules are made based on business requirements, it is possible that some rules cannot be enforced among the cooperative parties. In this work, we first propose an algorithm to check the enforceability of the given rules among cooperative parties, and also the mechanisms to make rules totally enforceable by modifying the rule configurations.

For the future works, we will study the problem of enforcing the join paths by adding rules to the exisint configurations. In addition, we will study the problem where a trusted third party is available. In such scenario, the problems of how to enforce the rules without modifications and what are the optimal ways to enforce the rules will be studied.

References

[1] 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. In *CIDR 2005*, pages 186–199.

[2] R. Agrawal, D. Asonov, M. Kantarcioglu, and Y. Li. Sovereign joins. In , *ICDE 2006, 3-8 April 2006, Atlanta, GA, USA*, page 26, 2006.

[3] A. V. Aho, C. Beeri, and J. D. Ullman. The theory of joins in relational databases. *ACM Transactions on Database Systems*, 4(3):297–314, Sept. 1979.

[4] E. Al-Shaer, A. El-Atawy, and T. Samak. Automated pseudo-live testing of firewall configuration enforcement. *IEEE Journal on Selected Areas in Communications*, 27(3):302–314, 2009.

[5] P. A. Bernstein, N. Goodman, E. Wong, C. L. Reeve, and J. B. Rothnie, Jr. Query processing in a system for distributed databases (SDD-1). *ACM Transactions on Database Systems*, 6(4):602–625, Dec. 1981.

[6] A. Calì and D. Martinenghi. Querying data under access limitations. In *ICDE 2008, April 7-12, 2008, Cancún, México*, pages 50–59, 2008.

[7] 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.

[8] S. De Capitani di Vimercati, S. Foresti, S. Jajodia, S. Paraboschi, and P. Samarati. Assessing query privileges via safe and efficient permission composition. In *CCS 2008, Virginia, USA, October 27-31, 2008*.

[9] S. De Capitani di Vimercati, S. Foresti, S. Jajodia, S. Paraboschi, and P. Samarati. Controlled information sharing in collaborative distributed query processing. In *ICDCS 2008*, Beijing, China, June 2008.

[10] J. Goldstein and P. Larson. Optimizing queries using materialized views: A practical, scalable solution. In *SIGMOD 2001*, pages 331–342.

[11] A. Y. Halevy. Answering queries using views: A survey. *VLDB Journal*, 10(4):270–294, 2001.

[12] D. Kossmann. The state of the art in distributed query processing. *ACM Comput. Survey*, 32(4):422–469, 2000.

[13] M. Le, K. Kant, and S. Jajodia. Adaption for policy changes in cloud cooperative data access. *submitted for publication, available at http://csis.gmu.edu/pub/policy.pdf*.

[14] C. Li. Computing complete answers to queries in the presence of limited access patterns. *VLDB Journal*, 12(3):211–227, 2003.

[15] R. Pottinger and A. Y. Halevy. Minicon: A scalable algorithm for answering queries using views. *VLDB Joural*, 10(2-3):182–198, 2001.

[16] A. Wool. A quantitative study of firewall configuration errors. *IEEE Computer*, 37(6):62–67, 2004.