Analysis of Scientific Workflow Provenance Access Control Policies

Ruiqi Luo*, Ping Yang*, Shiyong Lu† and Mikhail Gofman*
*Computer Science, State University of New York at Binghamton, Binghamton, NY, USA
Email: {rluo1,pyang,mgofman1}@binghamton.edu
†Department of Computer Science, Wayne State University, Detroit, Michigan, USA
Email: shiyong@cs.wayne.edu

***Abstract*—Provenance has become an important concept for services computing in general, and for scientific workflows in particular. Provenance often contains confidential data and dependencies whose access needs to be protected. Provenance access control policies control who can access which provenance information. Correct specification of provenance access control policies is critical to ensure system security. However, due to the sheer size of provenance, it is often difficult to comprehend the full effects of an access control policy by manual inspection alone due to complex multi-step dependencies and their interactions. In this paper, we present automated analysis algorithms and complexity results for three provenance analysis problems. We have also developed incremental strategies for these algorithms for evolving provenance and access control policies.**

I. Introduction

Provenance, which captures the derivation history of a data product, has become an important concept for services computing [1], [2], [3] in general, and for scientific workflows [4], [5], [6], [7] in particular. Provenance is essential for scientific workflows to support reproducibility of scientific discovery, result interpretation, and problem diagnosis [8], [9]. However, provenance collected from scientific workflows may contain confidential information. Due to the highly competitive nature of scientific research, it is important to ensure that sensitive provenance information can be accessed by and propagated to only authorized parties before the scientific results are ready for public release.

Access control is a security mechanism that restricts access to system resources to only authorized users. Existing access control mechanisms for business and scientific workflows (e.g. [10], [11], [12]) support secure execution of workflow tasks, but not secure access of their dependencies. Dependencies are confidential data in projects in which the protection of the derivation history of a scientific result is critical.

While we have previously developed a role-based access control mechanisms for scientific workflow provenance [5], how to ensure the correctness and understand the full effects of a provenance access control policy is still an open problem. This is challenging since in large scientific workflows, the size of the access control policies could also be large and it is often difficult to comprehend the full effects of a provenance access control policy by manual inspection alone due to complex multi-step dependencies and their interactions. In this paper, we present automated algorithms for analyzing provenance access control policies, which help administrators detect potential flaws in the policies. We define and solve three analysis problems: (1) the provenance access control policy existence problem, which checks whether there exists an access control policy that conforms to desirable dependency constraints; (2) the dependency satisfiability problem, which checks whether a given provenance access control policy conforms to desirable dependency constraints; and (3) the provenance completion problem, which checks whether a set of users together will be able to access all the dependencies in the provenance.

Our contributions are summarized below.

- We have developed algorithms for solving the above three analysis problems.
- We have shown that the three analysis problems are NP-complete and developed polynomial algorithms for solving special cases of these problems.
- We have developed incremental analysis algorithms for evolving provenance and access control policies. These algorithms incrementally update the analysis results by reusing the information obtained from the previous analysis, and hence are expected to be faster than the original algorithms.

II. Preliminaries

A. Scientific Workflow Provenance

Figure 1 shows a scientific workflow w, which models the scientific process of performing intragenomic gene conversion analysis in bacterial genomes. Each workflow task represents a computational or analytical step of a scientific process. This workflow takes as input the protein sequences of a given genome and identifies all its multi-gene families (task t_1). A particular multi-gene family is then selected by the user and its associated DNA sequences are retrieved (task t_2). Next, a recombination analysis is performed on the retrieved sequences (task t_3), which consists of two steps: a multiple DNA sequence alignment step (task t_4) and a gene conversion detection step (task t_5); the latter is implemented by the off-the-shelf program GENECONV (task t_7) with

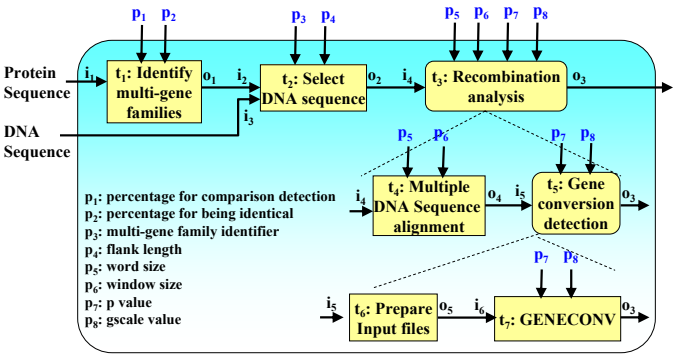

Figure 1. A hierarchical scientific workflow w.

an input data file preparation step (task t_6). Each task in this workflow has input ports ($i_1 - i_6$) and output ports ($o_1 - o_3$) that provide the communication interface to other tasks. Tasks are linked together via data channels. There are 8 parameters $p_1 - p_8$ in this workflow, which are used to configure its dynamic execution behavior. This workflow can be executed many times for different genomes or for the same genome using different parameter settings; Every execution produces a *workflow run provenance*.

Figure 2 gives a sample workflow run provenance of w, which is represented in a notation similar to the one used in the Open Provenance Model (OPM) [13]. Circles represent parameter values and data products, rectangles represent task runs that are labeled with task run identifier r_k and task identifier t_i, octagons represent users performing the tasks, and edges represent dependency relationships. Edge $d_i(p_i) \longleftarrow r_1 : t_j$ represents the *Used* dependency, which specifies that data product d_i (p_i) is the input to task t_j in workflow run r_1 and edge $r_1 : t_j \longleftarrow d_i$ represents the *WasGeneratedBy* dependency, which specifies that data product d_i is the output of task t_j in workflow run r_1. Edge $u \overset{WasControlledBy}{\longleftarrow} r_1 : t_j$ specifies that task t_j was executed by user u in workflow run r_1.

B. Role Based Access Control

Role based access control [14] has been widely used for restricting access to resources. In role based access control, users are assigned to roles and roles are assigned to permissions. Formally, an RBAC policy is defined as a tuple (U, R, P, UA, PA) where

- U, R, and P are finite sets of users, roles, and permissions, respectively.
- $UA \subseteq U \times R$ is a set of user-role assignments. $(u, r) \in UA$ specifies that user u is a member of role r. For example, $(alice, student) \in UA$ specifies that alice is a member of the student role.
- $PA \subseteq R \times P$ is the permission-role relation. $(r, p) \in PA$ specifies that role r has granted permission to access p. For example, $(student, dp) \in PA$ specifies that every member of the student role has permission to access dependency dp.

This paper is based on the role based access control for scientific workflow provenance proposed in [5] that controls which role can access which provenance component.

III. Dependency Analysis

Correct understanding of a provenance access control policy and its associated dependency constraints is critical for assuring provenance security. However, due to the sheer size of provenance, it is often hard to comprehend the full effects of a provenance access control policy by simple manual inspection alone. Dependency analysis help administrators understand a policy and its associated dependency constraints better. In this section, we consider two types of dependency analysis problems: (1) *provenance access control policy existence analysis*, which checks whether there exists an RBAC policy that satisfies a dependency constraint; and (2) *dependency satisfiability analysis*, which checks whether a given RBAC policy satisfies a dependency constraint. Below, we first define dependency and dependency constraint.

Definition 1 (Dependency): We say that a data product d_2 is derived from a data product d_1 in provenance P in one step (one-step dependency), denoted as $d_1 \rightarrow d_2$, if there exists a task t such that $d_1 \overset{Used}{\longleftarrow} t \in P$ and $t \overset{wasGeneratedBy}{\longleftarrow} d_2 \in P$. We use $d_1 \rightarrow^* d_n$ to denote that d_n is derived from d_1 by one or more steps.

Definition 2 (Provenance Dependency Graph): A provenance dependency graph G_P for provenance P is a directed acyclic graph $< V, E >$, where each node in V represents a data product in P and each edge in E represents a one-step dependency. A role provenance dependency graph $G_P(r)$ is a subgraph $< V', E' >$ of G_P where $(r, d) \in PA$ for all $d \in E'$.

Definition 3 (Dependency Constraint): A dependency constraint C is of the form $(c_{11} \vee \ldots \vee c_{1n}) \wedge \ldots \wedge (c_{m1} \vee \ldots \vee c_{mk})$, where c_{ij} is either $allow(r, d_1 \rightarrow^* d_2)$, which specifies that role r must be allowed to access dependency $d_1 \rightarrow^* d_2$, or $disallow(r, d_1 \rightarrow^* d_2)$, which specifies that role r must not be allowed to access dependency $d_1 \rightarrow^* d_2$.

Definition 4 (Dependency Satisfiability): We say that an RBAC policy satisfies dependency constraint $allow(r, d_1 \rightarrow^* d_2)$ if d_2 is derivable from d_1 in $G_P(r)$ in one or more steps. We say that an RBAC policy satisfies dependency constraint $disallow(r, d_1 \rightarrow^* d_2)$ if d_2 is not derivable from d_1 in $G_P(r)$.

In this paper, we consider RBAC with cardinality constraint of the form $cardinality(r, dset, k)$ $(n > k)$, which specifies that a user who is a member of role r can access at most k one-step dependencies in set $dset$. This means that, if a user of role r has accessed k dependencies in $dset$, then the user cannot access the rest of dependencies in $dset$. This constraint prevents a user who is a member of role r from accessing more than k dependencies in $dset$ even if the user

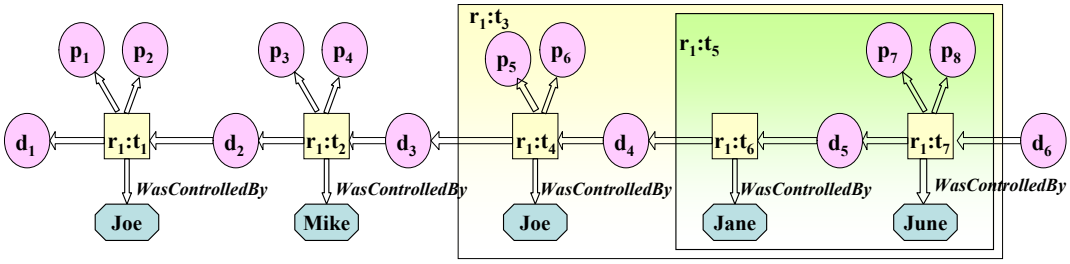

Figure 2. A sample provenance generated from the workflow in Figure 1.

is also a member of another role that does not have such a constraint.

A. Provenance Access Control Policy Existence Analysis

The provenance access control policy existence analysis (PE) asks, given provenance P and a dependency constraint C, does there exist an RBAC policy for P that satisfies C? If there does not exist an RBAC policy that satisfies C, then adding cardinality constraints to the policy will not help the policy satisfy C. As a result, we need to consider only RBAC without the cardinality constraint for PE.

Solving PE is non-trivial, because $allow/disallow$ constraints may interact with each other in unintended ways and trying to satisfy one $allow/disallow$ constraint may result in the violation of another constraint. As an example, consider dependences between data products $d_1, \ldots, d_5$ shown in Figure 3 and the dependency constraint that is the conjunction of the following $allow/disallow$ constraints: (1) $disallow(r, d_5 \rightarrow^* d_4)$; (2) $disallow(r, d_3 \rightarrow^* d_4)$; and (3) $allow(r, d_1 \rightarrow^* d_4)$. To satisfy (1), r should not be allowed to access either $d_5 \rightarrow d_2$ or $d_2 \rightarrow d_4$. To satisfy (2), r should not be allowed to access $d_3 \rightarrow d_4$. Assume that we do not allow r to access $d_2 \rightarrow d_4$ and $d_3 \rightarrow d_4$, then (3) cannot be satisfied. However, if we do not allow r to access $d_5 \rightarrow d_2$ at the first place, all constraints can be satisfied. Clearly, it is inefficient to try all combinations of dependencies that should not be granted to r in order to satisfy the constraint.

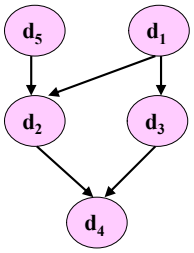

Figure 3. Example: PE analysis.

Below, we show that PE for RBAC without cardinality constraint is NP-complete. First, we prove that the satisfiability problem for a 3-CNF formula, where each clause contains either all positive or negative literals, is NP-complete; we call such a formula 3-CNF-SAME formula.

Lemma 1: The satisfiability problem for a 3-CNF-SAME formula is NP-complete.

Proof: Given a 3-CNF-SAME formula F and an assignment A, we can check whether A satisfies F in polynomial time. Thus the problem is in NP.

Next, we show that the problem is NP-hard by providing a polynomial-time reduction from the 3-CNF satisfiability problem to the problem. Let $F = F_1 \wedge \ldots \wedge F_n$ be a 3-CNF formula. First, we replace every negative literal $\bar{l}_i$ in F with l'_i and append $(l_i \vee l'_i \vee false) \wedge (\bar{l}_i \vee \bar{l}'_i \vee \overline{true})$ to the formula. After the reduction, every clause contains either all positive literals or all negative literals. The worst-case complexity of the reduction is $O(|F|)$.

Let F' be the transformed formula. Below, we show that F is satisfiable iff F' is satisfiable. First, we prove "only if". Assume that F is true under a set of assignments $A = \{(l_1, v_1), \ldots, (l_m, v_m)\}$. For every $(l_i, true) \in A$, we add $(l_i, true)$ and $(l'_i, false)$ to A'. For every $(l_i, false) \in A$, we add $(l_i, false)$ and $(l'_i, true)$ to A'. It is easy to see that F' is true under A'. Next, we prove "if". Assume that F' is true under a set of assignments A'. Then from the reduction, F is true under A'. Thus the problem is NP-hard. □

Theorem 2: PE for RBAC that does not contain the cardinality constraint is NP-Complete.

Proof: When the RBAC policy ψ does not contain the cardinality constraint, it takes polynomial time to check whether the policy satisfies the dependency constraint. Therefore, the problem is in NP.

Next, we show that PE is NP-hard by reducing the satisfiability problem for 3-CNF-SAME formula, to this problem. Let $F = F_1 \wedge F_2 \wedge \ldots \wedge F_n$ be a 3-CNF-SAME formula. For every literal l in F, we create two nodes $sn(l)$ and $dn(l)$, and add edge $sn(l) \rightarrow dn(l)$ to G_P. We then construct the dependency constraint $C = C_1 \wedge \ldots \wedge C_n$ as follows. For every clause $F_i = l_{i1} \vee l_{i2} \vee l_{i3}$, we construct $C_i = \bigvee_{1 \le j \le 3} allow(r, sn(l_{ij}) \rightarrow^* dn(l_{ij}))$. For every clause $F_i = \bar{l}_{i1} \vee \bar{l}_{i2} \vee \bar{l}_{i3}$, we construct $C_i = \bigvee_{1 \le j \le 3} disallow(r, sn(l_{ij}) \rightarrow^* dn(l_{ij}))$. The corresponding PE problem pe is: does there exist an RBAC policy for P that satisfies C.

Finally, we show that F is satisfiable iff pe has a solution.

Proof for "only if": Assume that F is true under assignments $A = \{(l_1, v_1), \ldots, (l_m, v_m)\}$. Then dependencies $\{sn(l_1) \rightarrow dn(l_1), \ldots, sn(l_m) \rightarrow dn(l_m)\}$ are added to G_P. Next, we show that $\psi = \{(r, sn(l_i) \rightarrow dn(j_i)) \in PA \mid v_i = true\}$ satisfies C. Suppose that ψ does not satisfy C. Then ψ does not satisfy C_i for some $1 \le i \le n$. If $C_i = \bigvee_{1 \le j \le 3} allow(r, sn(l_{ij}) \rightarrow *dn(l_{ij}))$, then r

does not have permission to access $sn(l_{ij}) \rightarrow dn(l_{ij})$ for all $1 \leq j \leq 3$. This means that all l_{ij} are $false$ and hence F_i is false. Thus F is not true under A, which is a contradiction. The case where C_i contains $disallow$ can be proven similarly.

Proof for "if": Assume that ψ satisfies C. l_i is assigned $true$ if $(r, sn(l_i) \rightarrow dn(l_i)) \in PA$; otherwise, l_i is assigned $false$. We now show that F is true under the above assignments. Suppose that this is not the case, then there exists an F_i that is false under the above assignments. If $F_i = l_{i1} \vee l_{i2} \vee l_{i3}$, then all l_{ij}s are false. This means that r does not have permission to access all $sn(l_{ij}) \rightarrow dn(l_{ij})$ and hence C_i is false. Thus C is not satisfied, which is a contradiction. The case where F_i contains only negative literals can be proven similarly.

As an example, consider $F = (l_1 \vee l_2 \vee l_3) \wedge (\bar{l}_1 \vee \bar{l}_2 \vee \bar{l}_4)$. After the reduction, four dependencies $sn(l_1) \rightarrow dn(l_1)$, $sn(l_2) \rightarrow dn(l_2)$, $sn(l_3) \rightarrow dn(l_3)$, $sn(l_4) \rightarrow dn(l_4)$ are added to G_P. The first conjunct in F is reduced to constraint $\bigvee_{1 \leq i \leq 3} allow(r, sn(l_i) \rightarrow^* dn(l_i))$. The second conjunct is reduced to constraint $disallow(r, sn(l_1) \rightarrow^* dn(l_1)) \vee disallow(r, sn(l_2) \rightarrow^* dn(l_2)) \vee disallow(r, sn(l_4) \rightarrow^* dn(l_4))$. F is true under assignments $\{(l_1, true), (l_2, false), (l_3, true), (l_4, false)\}$. The corresponding RBAC policy that satisfies the above constraint is $(r, sn(l_1) \rightarrow sn(l_1)) \in PA$, $(r, sn(l_3) \rightarrow dn(l_3)) \in PA$. □

Below, we present an algorithm for solving PE. The algorithm reduces PE to the satisfiability problem of propositional formulas; PE has a solution iff the corresponding formula is satisfiable. The pseudocode of the algorithm is given in Algorithm 1. Function $comp\_paths(r, d_1, d_2)$ computes a set of all paths from d_1 to d_2 in $G_P(r)$ and $lit(r, e)$ returns the literal corresponding to edge e in $G_P(r)$.

Below, we use the example in Figure 3 to illustrate our algorithm. Let $lit(d_1 \rightarrow d_2) = l_{r,1}$, $lit(d_1 \rightarrow d_3) = l_{r,2}$, $lit(d_5 \rightarrow d_2) = l_{r,3}$, $lit(d_2 \rightarrow d_4) = l_{r,4}$, and $lit(d_3 \rightarrow d_4) = l_{r,5}$. First, the algorithm processes $disallow(r, d_5 \rightarrow^* d_4)$. It computes the set of all paths $\{\{d_5 \rightarrow d_2, d_2 \rightarrow d_4\}\}$ from d_5 to d_4 and constructs formula $\bar{l}_{r,3} \vee \bar{l}_{r,4}$. Next, the algorithm processes $disallow(r, d_3 \rightarrow^* d_4)$ and constructs formula $\bar{l}_{r,5}$. Finally, the algorithm processes $allow(r, d_1 \rightarrow^* d_4)$ and constructs formula $(l_{r,1} \wedge l_{r,4}) \vee (l_{r,2} \wedge l_{r,5})$. The conjunction of the above formula is true under the assignments: $\{(l_{r,5}, false), (l_{r,3}, false), (l_{r,1}, true), (l_{r,4}, true), (l_{r,2}, false)\}$. The corresponding RBAC policy is: $(r, d_1 \rightarrow d_2) \in PA$, $(r, d_2 \rightarrow d_4) \in PA$.

Let $|Od|$ be the maximum outgoing degree of nodes in G_P and $|Depth|$ be the maximum depth of G_P. The maximum number of paths in G_P is $|Od|^{|Depth|}$. Thus, the worst-case complexity of Algorithm 1 is $O(|C|(|Od|^{|Depth|})$.

We have also identified the following cases under which PE can be solved in polynomial time. Due to space constraints, the proofs are not given in the paper.

Theorem 3: PE for dependency constraint that contains only $allow$ or $disallow$ can be solved in polynomial time.

Theorem 4: PE for dependency constraint that does not contain $\wedge$ can be solved in polynomial time.

Algorithm 1 Provenance access control policy existence analysis algorithm.

```
1: Procedure dep_analysis(C, P)
2: formula = ∅
3: if C = allow(r, d1 →* d2) then
4:    path_set = comp_paths(r, d1, d2)
5:    for all p ∈ path_set do
6:       clause = ∅
7:       for all e ∈ p do
8:          clause = clause ∧ lit(r, e)
9:       end for
10:      formula = formula ∨ clause
11:   end for
12:   return formula
13: end if
14: if C = dislow(r, d1 →* d2) then
15:   path_set = comp_paths(r, d1, d2)
16:   for all p ∈ path_set do
17:      clause = ∅
18:      for all e ∈ p do
19:         clause = clause ∨ ¬lit(r, e)
20:      end for
21:      formula = formula ∧ clause
22:   end for
23:   return formula
24: end if
25: if C = C1 ∨ C2 then
26:   return dep_analysis(C1, P) ∨ dep_analysis(C2, P)
27: end if
28: if C = C1 ∧ C2 then
29:   return dep_analysis(C1, P) ∧ dep_analysis(C2, P)
30: end if
```

B. Dependency Satisfiability Analysis

Dependency satisfiability analysis (DS) asks, given provenance P, an RBAC policy ψ for P, and a dependency constraint C, does ψ satisfy C? If ψ does not contain the cardinality constraint, DS can be easily solved by reducing it to the reachability analysis problem of dependency graphs for roles in C. The worst-case time complexity of the algorithm is $O(|C||G_P|)$.

With cardinality constraints, the problem becomes NP-complete, as shown below.

Theorem 5: DS is NP-Complete.

Proof: Given a set of role-dependency assignments Rd that conforms to RBAC policy ψ, it takes polynomial time to check if Rd satisfies the dependency constraint C. Therefore, the problem is in NP.

Next, we show that DS is NP-hard by providing a polynomial time reduction from the satisfiability problem of 3-CNF-SAME to the problem. Without loss of generality, we assume that three literals in the same clause are pairwise different. Let $F = F_1 \wedge \ldots \wedge F_n$ be a 3-CNF-SAME formula where $F_i = l_{i1} \wedge l_{i2} \wedge l_{i3}$ or $\bar{l}_{i1} \wedge \bar{l}_{i2} \wedge \bar{l}_{i3}$.

For every clause F_i that contains all positive literals, we append $allow(r, S \rightarrow^* T_i)$ to dependency constraint C as a conjunct, and for every l_{ij} in F_i, we add $(r, S \rightarrow node(l_{ij})) \in PA$ and $(r, node(l_{ij}) \rightarrow T_i) \in PA$ to ψ. For every clause F_i that contains all negative literals, we add $cardinality(r, \{S \rightarrow node(l_{i1}), S \rightarrow node(l_{i2}), S \rightarrow node(l_{i3})\}, 2)$ to ψ. The corresponding DS problem ds is: does policy ψ satisfy the dependency constraint C? The worst-case complexity of the reduction is $O(|F|)$.

Finally, we show that F is satisfiable iff ds has a solution.

Proof for "only if": Assume that F is true under assignments $\{(l_1, v_1), \ldots, (l_m, v_m)\}$. For every F_i containing all positive literals, if l_{ij} is true, then $(r, S \rightarrow node(l_{ij})) \in PA$ and $(r, node(l_{ij}) \rightarrow T_i) \in PA$ satisfy $allow(r, S \rightarrow^* T_i)$. For every clause F_i that contains all negative literals, at least one of l_{ij} is $false$. Correspondingly, r does not have permission to access one of the following dependencies: $S \rightarrow node(l_{i1})$, $S \rightarrow node(l_{i2})$, and $S \rightarrow node(l_{i3})$. Therefore, the cardinality constraint is satisfied. Thus ds has a solution.

Proof for "if": If ds has a solution, then ψ satisfies the dependency constraint C. This means that, for every F_i that contains only positive literals, r can access at least one of the dependencies $S \rightarrow node(l_{i1})$, $S \rightarrow node(l_{i2})$, and $S \rightarrow node(l_{i3})$. In this case, the corresponding literal in F_i is assigned $true$ and hence F_i is $true$. Since all cardinality constraints are satisfied, for every F_i containing only negative literals, r cannot access at least one of the dependencies $S \rightarrow node(l_{i1})$, $S \rightarrow node(l_{i2})$, and $S \rightarrow node(l_{l3})$. In this case, the corresponding literal is assigned $false$ and hence F_i is true. Note that a literal cannot be assigned both $true$ and $false$. Otherwise, assume that l_{ij} is assigned both $true$ and $false$. Then $S \rightarrow node(l_{ij})$ is both accessible and not accessible by r, which is a contradiction. Therefore, F is true and hence DS is NP-hard.

As an example, consider the 3-CNF-SAME formula $F = (l_1 \vee l_2 \vee l_3) \wedge (\bar{l}_1 \vee \bar{l}_2 \vee \bar{l}_3)$. First, we process the first clause, add $allow(r, S \rightarrow^* T_1)$ to C, and add $(r, S \rightarrow node(l_{1j})) \in PA$ and $(r, node(l_{1j}) \rightarrow T_1) \in PA$ for $1 \leq j \leq 3$ to ψ. We then process the second clause and add $cardinality(r, \{S \rightarrow node(l_{11}), S \rightarrow node(l_{12}), S \rightarrow node(l_{13})\}, 2)$ to ψ. F is true under assignments $\{(l_1, true), (l_2, true), (l_3, false)\}$. Correspondingly, ψ grants role r permission to access dependencies $S \rightarrow node(l_1)$, $node(l_1) \rightarrow T_1$, $S \rightarrow node(l_2)$, and $node(l_1) \rightarrow T_1$, which satisfies C. □

A naive algorithm for solving DS is given below. First, for every $disallow(r, d_1 \rightarrow^* d_2)$ in C_i, if there does not exist a path from d_1 to d_2 in $G_P(r)$, then C_i is replaced with $true$; otherwise, $disallow(r, d_1 \rightarrow^* d_2)$ is replaced with $false$. For every $allow(r, d_1 \rightarrow^* d_2)$ in C_i, if there does not exist a path from d_1 to d_2 in $G_P(r)$, C_i is replaced with $false$. Let C' be the resulting constraint. Next, we compute all possible combinations of paths that need to present in order to satisfy C'. If one combination of paths satisfies the cardinality constraint, then the algorithm returns true; otherwise false. The worst-case complexity of the algorithm is $O(|C|(|Od|^{|Depth|})$ where $|Od|$ is the maximum outgoing degree of nodes in G_P and $|Depth|$ is the maximum depth of G_P.

As an example, consider the dependency graph in Figure 3, dependency constraint $(allow(r, d_1 \rightarrow^* d_4) \vee allow(r, d_5 \rightarrow^* d_4)) \wedge allow(r, d_1 \rightarrow^* d_2)$, and cardinality constraint $cardinality(r, \{d_1 \rightarrow d_3, d_3 \rightarrow d_4, d_1 \rightarrow d_2\}, 2)$. Assume that $(r, d) \in PA$ for every dependency d in the graph. The following combinations of paths satisfy the dependency constraint: (a) $\{d_1 \rightarrow d_3, d_3 \rightarrow d_4, d_1 \rightarrow d_2\}$, (b) $\{d_1 \rightarrow d_2, d_2 \rightarrow d_4\}$, (c) $\{d_1 \rightarrow d_2, d_5 \rightarrow d_2, d_2 \rightarrow d_4\}$, but only (b) and (c) satisfy the cardinality constraint. Therefore the algorithm returns $true$.

IV. Provenance Completion Problem

Provenance completion problem (PC) asks, given an RBAC policy ψ and provenance metadata P, does there exist a set of users $U' \subseteq U$ that together can access all dependencies $D = \{dp_1, \cdots, dp_m\}$ in P under ψ? The problem is true iff there exists a set of user-dependency assignments $\{(u_1, dp_1), \ldots, (u_n, dp_m)\}$ such that $u_i \in U'$ and the assignments conforms to ψ.

A naive algorithm for solving the problem is given below, which is a brute-force approach that consists of two phases. In the first phase, for every dependency $dp \in D$, if there exists a user $u \in U'$ such that $(u, r) \in UA$ and $(r, dp) \in PA$, and there does not exist $cardinality(r, dset, k) \in \psi$ where $dp \in dset$, then the algorithm assigns u to dp. Otherwise, dp is added to a working set S. In the second phase, we try all possible combinations of assignments of users in U' to S to search for a solution in which each dependency in S is assigned to some user in U' and no cardinality constraint is violated. The procedure terminates when such a solution is found or all combinations are exhausted. The worst-case time complexity of the algorithm is $O(|U'|^{|S|})$.

Theorem 6: PC is NP-complete.

Proof: Given a set of user-dependency assignments $A \subseteq U' \times D$, we can verify whether every dependency in D has been assigned to a user in U' and if the assignment conforms to the RBAC policy in polynomial time. Therefore, the problem is in NP.

Next, we show that PC is NP-hard by providing a polynomial time reduction from the 3-CNF satisfiability problem to the problem. Let $F = F_1 \wedge F_2 \wedge \ldots \wedge F_n$ be a 3-CNF formula. The reduction is given below. Each clause F_j is mapped to a dependency dp_j. Each literal l_i is mapped to a user-role assignment $(u_i, r_i) \in UA$. If l_i appears (either positively or negatively) in F_j, $(r_i, dp_j) \in PA$ is added to ψ. If l_i appears positively in F_{j1} and negatively in F_{j2}, a cardinality constraint $cardinality(r_i, \{dp_{j1}, dp_{j2}\}, 1)$ is added to ψ. Let k be the number of literals in F. The

corresponding provenance completion problem pc is: does there exist a set of users $U' \subseteq \{u_1, \ldots, u_k\}$ that together can access all dependencies $\{dp_1, \ldots, dp_n\}$ under the RBAC policy ψ.

Below, we show that F is satisfiable iff pc has a solution.

Proof for "only if": Assume that F is true under the set of assignments $\{(l_1, v_1), \ldots, (l_k, v_k)\}$. If $v_i = true$, l_i appears positively in F_j, and dp_j has not been assigned to any user, then assign dp_j to u_i. if $v_i = false$, l_i appears negatively in F_j, and dp_j has not been assigned to any user, then assign dp_j to u_i. Since F is true, all F_is are true and hence all dp_is are assigned to some user. Next, we prove that the above assignment does not violate the cardinality constraint. Assume that $cardinality(r, \{dp_{j1}, dp_{j2}\}, 1)$ is violated. Then there exists $u \in U'$ such that $(u, r) \in UA$ and u is assigned to both dp_{j1} and dp_{j2}. From the reduction, one of the following holds: (1) l_i appears positively in F_{j1} and negatively in F_{j2}; or (2) l_i appears negatively in F_{j1} and positively in F_{j2}. In both cases, l_i is assigned both values $true$ and $false$, which is a contradiction. Therefore, pc has a solution.

Proof for "if": Assume that pc has a solution $\{(u_1, dp_1), \ldots, (u_m, dp_n)\}$. We iterate from dp_1 to dp_n and apply one of the following rules to compute a set of assignments under which F is true: (1) If u_i is assigned dp_j, l_i appears positively in F_j, and l_i has not been assigned any value, then l_i is assigned $true$. (2) If u_i is assigned dp_j, p_i appears negatively in F_j, and l_i has not been assigned any value, then l_i is assigned $false$. Since every dependency is assigned to some user, the above rules guarantee that all F_is are true. The cardinality constraint ensures that no literals can be assigned to both $true$ and $false$. Therefore F is true under the above assignments. ■

As an example, consider the 3-CNF formula $(l_1 \vee \bar{l}_2 \vee l_3) \wedge (l_2 \vee l_3 \vee l_4) \wedge (l_2 \vee l_3 \vee l_5)$. The following RBAC policy is generated: $(u_1, r_1) \in UA$, $(u_2, r_2) \in UA$, $(u_3, r_3) \in UA$, $(u_4, r_4) \in UA$, $(u_5, r_5) \in UA$ $(r_1, dp_1) \in PA$, $(r_2, dp_1) \in PA$, $(r_3, dp_1) \in PA$, $(r_2, dp_2) \in PA$, $(r_3, dp_2) \in PA$, $(r_4, dp_2) \in PA$, $(r_2, dp_3) \in PA$, $(r_3, dp_3) \in PA$, $(r_5, dp_3) \in PA$, $cardinality(r_2, \{dp_1, dp_2\}, 1)$, $cardinality(r_2, \{dp_1, dp_3\}, 1)$. The corresponding PC is: does there exist a set of users $U' \subseteq \{u_1, u_2, u_3, u_4, u_5\}$ that together can access all dependencies $\{dp_1, dp_2, dp_3\}$ under the RBAC policy ψ? The formula is satisfied under assignment $\{(l_2, false), (l_3, true), (l_5, true)\}$. The solution for the corresponding PC is $\{(u_2, dp_1), (u_3, dp_2), (u_5, dp_3)\}$.

Below, we provide a polynomial-time algorithm for solving the following two special cases: (1) each user is constrained at most once, i.e., for every user $u \in U'$, there do not exist $cardinality(r_1, dset_1, k_1) \in \psi$ and $cardinality(r_2, dset_2, k_2) \in \psi$, such that $(u, r_1) \in UA$ and $(u, r_2) \in UA$; or (2) if a user $u \in U'$ is a member of roles in the set of constraints $cardinality(r_1, dset_1, k_1), \ldots, cardinality(r_n, dset_2, k_n)$, then $dset_i \cap dset_j = \emptyset$ for all $1 \leq i, j \leq n$.

Let Con be a set of all cardinality constraints in ψ. The algorithm is given below. First, we perform the first stage of the naive algorithm. Next, for every $cardinality(r, dset, k) \in Con$, if the constraint satisfies $|\{s \mid s \in S \wedge s \in dset\}| \leq k$, we assign a user who is a member of role r to all dependencies in $dset$, remove such dependencies from S, and remove the constraint from Con. We then construct a bipartite graph G from U' and S as follows. Each vertice in G corresponds to one dependency in S or one user in U' that is a member of a role in the cardinality constraint in Con. Let V_u represent the vertice corresponding to user u and V_{dp} represent the vertice corresponding to dependency dp. For every $cardinality(r, pdet, k) \in Con$ and every u who is a member of role r, we duplicate V_u $k-1$ times, which results in nodes $V_u^1, \ldots, V_u^{(k-1)}$. Next, we add edges from $V_u, V_u^1, \ldots, V_u^{(k-1)}$ to all vertices corresponding to dependencies in $dset$ that can be accessed by r. Finally, we apply the maximum bipartite matching algorithm to compute the maximum matching between vertices representing users and vertices representing dependencies. The problem is true if the size of the maximum matching is equal to $|S|$, which means that all dependencies in S can be assigned to some users in U'. The number of times a user vertice is duplicated ensures that the maximum matching also does not violate the cardinality constraint. Note that this algorithm can also be applied to the case where all constraints in Con updated after the first stage satisfy one of the two conditions.

The worst-case complexity for the first stage is $O(|Con||U'||R||D|)$. The worst-case complexity of maximum matching algorithm for a graph $G =< V, E >$ is $O(|V|^{(1/2)}|E|)$. In our algorithm, the maximum number of vertices and edges are $|U'||S| + |S|$ and $|U'||S|^2$, respectively. Therefore the worst-case complexity of the above algorithm is $O((|Con||U'||R||D| + (|U'||S| + |S|)^{(1/2)}(|U'||S|^2))$.

As an example, consider the following RBAC policy:
$(u_1, r_1) \in UA$, $(u_2, r_2) \in UA$, $(u_3, r_3) \in UA$, $(u_3, r_4) \in UA$, $(u_4, r_4) \in UA$, $(u_5, r_5) \in UA$ $(r_1, dp_1) \in PA$, $(r_1, dp_2) \in PA$, $(r_1, dp_3) \in PA$, $(r_2, dp_3) \in PA$, $(r_2, dp_4) \in PA$, $(r_3, dp_4) \in PA$, $(r_3, dp_5) \in PA$ $(r_3, dp_6) \in PA$, $(r_4, dp_7) \in PA$, $(r_4, dp_8) \in PA$, $(r_4, dp_9) \in PA$, $(r_5, dp_9) \in PA$, $(r_5, dp_{10}) \in PA$, $cardinality(r_1, \{dp_1, dp_2, dp_3\}, 2)$, $cardinality(r_3, \{dp_4, dp_5, dp_6\}, 2)$, $cardinality(r_4, \{dp_7, dp_8, dp_9\}, 2)$. $cardinality(r_5, \{dp_9, dp_{10}\}, 1)$.

Our algorithm works as follows. First, u_2 is assigned to dp_3 and dp_4, because u_2 is not a member of roles in any cardinality constraint. Other dependencies are added to S. We then remove dp_3 from $cardinality(r_1, \{dp_1, dp_2, dp_3\}, 2)$ and remove dp_4 from $cardinality(r_3, \{dp_4, dp_5, dp_6\}, 2)$. As a result, the number of dependencies in these

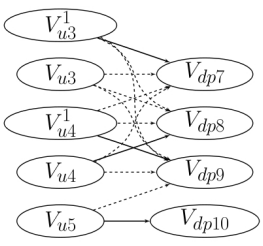

Figure 4. Example: algorithm for two special cases of PC

two constraints is 2 and hence these two constraints are satisfied. We then assign u_1 to dp_1 and dp_2, assign u_3 to dp_5 and dp_6, remove these dependencies from S, and remove the two constraints from Con. Next, we process $cardinality(r_4, \{dp_7, dp_8, dp_9\}, 2)$ and $cardinality(r_5, \{dp_9, dp_{10}\}, 1)$, and construct a bipartite graph as shown in Figure 4. Since u_3 and u_4 are members of role r_4, u_3 and u_4 are duplicated once, which results in two new vertices u_3^1 and u_4^1. We then add edges from these vertices to V_{dp_7}, V_{dp_8} and V_{dp_9}. Since u_5 is a member of r_5, we add edges from V_{u_5} to V_{dp_9} and $V_{dp_{10}}$. Finally, the maximum matching algorithm is applied to compute a solution for the problem. In Figure 4, edges are represented using dotted/solid lines, and edges represented using solid lines are the maximum matching computed. The algorithm returns true as the size of maximum matching is equal to $|S|$. The solution is $\{(u_1, dp_1), (u_1, dp_2), (u_2, dp_3), (u_2, dp_4), (u_3, dp_5), (u_3, dp_6), (u_3, dp_7), (u_4, dp_8), (u_4, dp_9), (u_5, dp_{10})\}$.

V. Incremental Analysis Algorithms

The provenance access control policy and dependency constraints may evolve over time in order to fix flaws and cope with changing requirements of organizations. Changes to the policy or the provenance may invalidate the analysis result. It would be inefficient to perform reanalysis from scratch every time a change occurs. To address this issue, we present efficient algorithms that reuse the previous analysis result to incrementally perform reanalysis. Due to space constraints, we present only algorithms for the provenance completion problem (PC) and consider only operations for adding user-role relation, permission role relation, and the cardinality constraint. Other operations and incremental algorithms for DS and PE can be handled similarly.

To enable incremental analysis, we store the set of user-dependency assignments A_1 and the workset S computed in the first phase. If the previous result is true, we also store the set of user-dependency assignments A_2 computed in the second phase.

Add $cardinality(r, dset, k)$ to ψ:: If $(u, r) \notin UA$ for all $u \in U'$, the algorithm simply returns the previous result. If the previous result is false, then adding $cardinality(r, dset, k)$ does not change the result, but may invalidate some assignments in A_1. Our algorithm updates A_1 as follows: for every $dp \in dset$, if $(u, dp) \in A_1$ and $(u, r) \in UA$, then the algorithm tries to find another user $u' \in U'$ such that $(u', r') \in UA$ and $(r', dp) \in PA$, and there does not exist $cardinality(r', dset', k) \in \psi$ where $dp \in dset'$. If such a user exists, the algorithm replaces (u, dp) with (u', dp). In other cases, the algorithm removes (u, dp) from A_1 and adds dp to the workset S. If the previous result is true, then adding $cardinality(r, dset, k)$ may change the result. In this case, the algorithm updates A_1 and S as given above, and performs the second phase using the updated A_1 and S.

Add $(r, dp) \in PA$: If (1) $(u, r) \notin UA$ for all $u \in U'$ or (2) the previous result is true, then adding $(r, dp) \in PA$ does not change the result. In case (2), the assignments in A_1 may change. The algorithm updates A_1 as follows: if $dp \in S$, there exists a user $u \in U'$ such that $(u, r) \in UA$, and there does not exist $cardinality(r, dset, k) \in \psi$ where $dp \in dset$, then the algorithm adds (u, dp) to A_1, removes dp from S, and removes the assignment for dp from A_2. If the previous result is false, the algorithm updates A_1 and S as given above, and performs the second phase.

Add $(u, r) \in UA$: If (1) $u \neq u'$ for all $u' \in U'$ or (2) the previous result is true, then adding $(u, r) \in UA$ does not change the result. In case (2), if there does not exist $(u', r) \in UA$ such that $u' \in U'$, then the algorithm updates A_1 as follows. For every $dp \in S$, if $(r, dp) \in PA$ and there does not exist $cardinality(r, dset, k) \in \psi$ where $dp \in dset$, then the algorithm adds (u, dp) to A_1, removes dp from S, and performs the second phase. If the previous result is false, then adding (u, r) may change the result. In this case, the algorithm updates A_1 as given above and performs the second phase.

VI. Related Work

Provenance access control:: Recently, security issues for provenance have been identified by several researchers. Braun et al. argued that a new security model is needed for protecting provenance [15] and presented a security model for provenance, in which security requirements are modeled as a multi-level system [16]. Tan et al. [17] and Tsai et al. [18] discussed security issues in SOA-based provenance systems. Artem et al. [5] proposed a role-based access control mechanism for scientific workflow provenance. However, none of these work considered formal security analysis of provenance access control policies.

Provenance dependency analysis:: Analysis of access control policies has been long recognized as an important problem (e.g. [19], [20], [21], [22], [23]), which checks whether an access control policy conforms to given security properties (e.g. reachability, availability, containment). Cheney et al. [24] proposed a semantic characterization of dependency provenance, showed that minimal dependency provenance is not computable, and presented provenance analysis techniques. Sun et al. [25] proved that the problem of identifying and correcting unsound workflow views with minimal changes is NP-hard. They have also developed

polynomial time algorithms for correcting unsound views to meet two local optimality conditions. However, to the best of our knowledge, PE and DS have not been considered by other researchers.

Provenance completion problem:: When the cardinality constraint is not considered, the provenance completion problem can be solved using algorithms for workflow satisfiability analysis [26], [27], [28]. However, none of them considered the cardinality constraint defined in this paper.

VII. Conclusion and future work

In this paper, we have presented algorithms and complexity results for three provenance analysis problems and developed incremental algorithms for analyzing evolving provenance and RBAC policies.

In the future, we will explore the possibility of developing *fixed-parameter tractable* algorithms for solving these problems, i.e., algorithms that are exponential in the size of small factors but are polynomial in the size of large factors. In addition, we will consider several subclasses of problem instances by imposing structural restrictions on dependency constraints. For example, when the dependency constraints do not contain $\vee$, it is not clear if PE is still NP-complete.

Acknowledgements This work was supported in part by NSF Grant CNS-0855204.

References

[1] W. Ding, J. Wang, and Y. Han, "Vipen: A model supporting knowledge provenance for exploratory service composition," in *Proc. of the IEEE International Conference on Services Computing (SCC 2010)*, 2010, pp. 265–272.

[2] A. Michlmayr, F. Rosenberg, P. Leitner, and S. Dustdar, "Service provenance in QoS-aware web service runtimes," in *IEEE International Conference on Web Services*, 2009.

[3] Y. L. Simmhan, B. Plale, and D. Gannon, "A framework for collecting provenance in data-centric scientific workflows," in *Proc. of the IEEE International Conference on Web Services (ICWS 2006)*, 2006, pp. 427–436.

[4] J. Zhang, D. Kuc, and S. Lu, "Confucius: A tool supporting collaborative scientific workflow composition," *IEEE Transactions on Services Computing (TSC)*, 2012, in press.

[5] A. Chebotko, S. Lu, S. Chang, F. Fotouhi, and P. Yang, "Secure abstraction views for scientific workflow provenance querying," *IEEE T. Services Computing*, vol. 3, no. 4, pp. 322–337, 2010.

[6] J. Abraham, P. Brazier, A. Chebotko, J. Navarro, and A. Piazza, "Distributed storage and querying techniques for a semantic web of scientific workflow provenance," in *Proc. of the IEEE International Conference on Services Computing (SCC)*, 2010, pp. 178–185.

[7] C. Lim, S. Lu, A. Chebotko, and F. Fotouhi, "Prospective and retrospective provenance collection in scientific workflow environments," in *Proc. of the IEEE International Conference on Services Computing (SCC)*, 2010, pp. 449–456.

[8] Y. Simmhan, B. Plale, and D. Gannon, "A survey of data provenance in e-Science," *SIGMOD Record*, vol. 34, no. 3, pp. 31–36, Sept. 2005.

[9] R. Bose and J. Frew, "Lineage retrieval for scientific data processing: a survey," *ACM Computing Surveys*, vol. 37, no. 1, pp. 1–28, 2005.

[10] E. Bertino, E. Ferrari, and V. Atluri, "A flexible model supporting the specification and enforcement of role-based authorization in workflow management systems." in *ACM Workshop on Role-Based Access Control*, 1997, pp. 1–12.

[11] M. H. Kang, J. S. Park, and J. N. Froscher, "Access control mechanisms for inter-organizational workflow," in *ACM symposium on Access control models and technologies*, 2001, pp. 66–74.

[12] S. Kandala and R. Sandhu, "Secure role-based workflow models," in *Annual Working Conference on Database and Ap plication Security*, 2001, pp. 45–58.

[13] *Open Provenance Model*, http://www.openprovenance.org/.

[14] R. Sandhu, D. Ferraiolo, and R. K. D, "The NIST model for role based access control: Towards a unified standard," in *5th ACM Workshop on Role Based Access Control*, 2000.

[15] U. Braun, A. Shinnar, and M. Seltzer, "Securing provenance," in *Proceedings of the 3rd conference on Hot topics in secuity*, 2008, pp. 1–5.

[16] U. Braun and A. Shinna, "A security model for provenance," Harvard University, Tech. Rep. TR-04-06, 2006.

[17] V. Tan, P. Groth, S. Miles, S. Jiang, S. Munroe, S. Tsasakou, and L. Moreau, "Security issues in a SOA-based provenance system," in *Proc. of the third International Provenance and Annotation Workshop*, 2006.

[18] W. Tsai, X. Wei, Y. Chen, R. Paul, J. Chung, and D. Zhang, "Data provenance in soa: security, reliability, and integrity," *Service Oriented Computing and Applications*, vol. 1, no. 4, pp. 223–247, December 2007.

[19] M. A. Harrison, W. L. Ruzzo, and J. D. Ullman, "Protection in operating systems," *Communications of the ACM*, vol. 19, no. 8, pp. 461–471, 1976.

[20] R. Sandhu, "Transaction control expressions for separation of duties," in *Proc. of the Fourth Computer Security Applications Conference*, 1988, pp. 282–286.

[21] S. Stoller, P. Yang, C. R. Ramakrishnan, and M. Gofman, "Efficient policy analysis for administrative role based access control," in *ACM Conference on Computer and Communication Security (CCS)*. ACM Press, 2007, pp. 445–455.

[22] M. Gofman, R. Luo, and P. Yang, "User-role reachability analysis of evolving administrative role based access control," in *15th European Symposium on Research in Computer Security (ESORICS)*, 2010.

[23] A. Sasturkar, P. Yang, S. D. Stoller, and C. Ramakrishnan, "Policy analysis for administrative role based access control," *Theoretical Computer Science*, vol. 412(44), pp. 6208–6234, 2011.

[24] J. Cheney, A. Ahmed, and U. Acar, "Provenance as dependency analysis," in *Proceedings of the 11th International Conference on Database Programming Languages*, 2007.

[25] P. Sun, Z. Liu, S. Davidson, and Y. Chen, "Detecting and resolving unsound workflow views for correct provenance analysis," in *Proceedings of the 35th SIGMOD International Conference on Management of Data*, 2009.

[26] Q. Wang and N. Li, "Satisfiability and resiliency in workflow authorization systems," *ACM Trans. Inf. Syst. Secur.*, vol. 13, pp. 40:1–40:35, December 2010.

[27] J. Crampton and H. Khambhammettu, "Delegation and satisfiability in workflow systems," in *ACM symposium on Access control models and technologies*, 2008, pp. 31–40.

[28] J. Crampton, "A reference monitor for workflow systems with constrained task execution," in *ACM symposium on Access control models and technologies*, 2005, pp. 38–47.