Secure Abstraction Views for Scientific Workflow Provenance Querying

Artem Chebotko, *Member*, *IEEE*, Shiyong Lu, *Senior Member*, *IEEE*, Seunghan Chang, Farshad Fotouhi, *Member*, *IEEE*, and Ping Yang, *Member*, *IEEE*

Abstract—Provenance has become increasingly important in scientific workflows and services computing to capture the derivation history of a data product, including the original data sources, intermediate data products, and the steps that were applied to produce the data product. In many cases, both scientific results and the used protocol are sensitive and effective access control mechanisms are essential to protect their confidentiality. In this paper, we propose: 1) a formal scientific workflow provenance model as the basis for querying and access control for workflow provenance; 2) a security model for fine-grained access control for multilevel provenance and an algorithm for the derivation of a full security specification based on inheritance, overriding, and conflict resolution; 3) a formalization of the notion of security views and an algorithm for security view derivation; and 4) a formalization of the notion of secure abstraction views and an algorithm for its computation. A prototype called SECPROV has been developed, and experiments show the effectiveness and efficiency of our approach.

Index Terms—Scientific workflows, provenance, access control, security, abstraction, secure querying.

1 INTRODUCTION

Provenance management has become increasingly important in the areas of services computing [1], [2], [3] and scientific workflows [4], [5], [6]. Provenance data capture the derivation history of a data product, including the original data sources, intermediate data products, and the steps that were applied to produce the data product. Therefore, provenance captures the detailed protocol of a scientific experiment. In many cases, both scientific results and the used protocol are sensitive and effective access control mechanisms are essential to equip scientists with a fine-grained tool to release only partial provenance information (data products, dependencies, and parameters) to stakeholders or to the public. In a typical scenario, scientific workflows are often the intellectual property of a scientist, since the composition of various computational services into a workflow is crucial to obtaining interesting scientific results. Thus, a scientist might be willing to publish the scientific results as well as the source data used to obtain such results, but not the scientific workflow itself. In another scenario, a scientist might publish both the scientific results and source data, as well as the scientific workflow used to obtain the results, but might keep the parameter setting used for the workflow run as a secret. In a more general scenario, a scientist might release partial provenance information concerning scientific results, source data, scientific workflows, and parameter settings that are just enough to convince stakeholders, but hide certain provenance information to protect intellectual property. The above scenarios illustrate that a flexible and expressive fine-grained access control mechanism is necessary for scientific workflow provenance.

The importance and requirements of security have been well understood in business workflows [7], [8], [9]. However, while a traditional workflow access control protects the access to workflow tasks and data, a provenance access control protects the access to data products (source, intermediate, and final) as well as the dependencies among them. Another dimension of complexity for scientific workflow provenance is its multilevel semantics. Since scientific workflows may consist of composite tasks or subworkflows, provenance can be accessed at different abstraction levels (with composite tasks "folded" or "unfolded"). A user may be interested in viewing only provenance that is related to a composite task as a black box rather than viewing more detailed provenance of all the constituent tasks. These different abstraction views can hide or reveal provenance information based on a particular user requirements and preferences; thus, providing a convenient abstraction mechanism that enables a user to view only relevant information. We further illustrate how security and abstraction mechanisms interact with each other using a real-life example from the bacterial genome Intragenomic Gene Conversions (IGCs) project [10] in the following.

The crux of the IGC project is the intragenomic recombination analysis scientific workflow shown in Fig. 1; the workflow is simplified from our original

- *A. Chebotko is with the Department of Computer Science, University of Texas—Pan American, 1201 West University Dr., Edinburg, TX 78539. E-mail: artem@cs.panam.edu.*
- *S. Lu, S. Chang, and F. Fotouhi are with the Department of Computer Science, Wayne State University, 5057 Woodward Ave., Detroit, MI 48202. E-mail: {shiyong, chang, fotouhi}@wayne.edu.*
- *P. Yang is with the Department of Computer Science, State University of New York at Binghamton, T-6, Engineering Building, Binghamton, NY 13902. E-mail: pyang@cs.binghamton.edu.*

Manuscript received 5 May 2009; revised 2 Sept. 2009; accepted 27 May 2010; published online 25 Aug. 2010.
For information on obtaining reprints of this article, please send e-mail to: tsc@computer.org, and reference IEEECS Log Number TSC-2009-05-0116.
Digital Object Identifier no. 10.1109/TSC.2010.38.

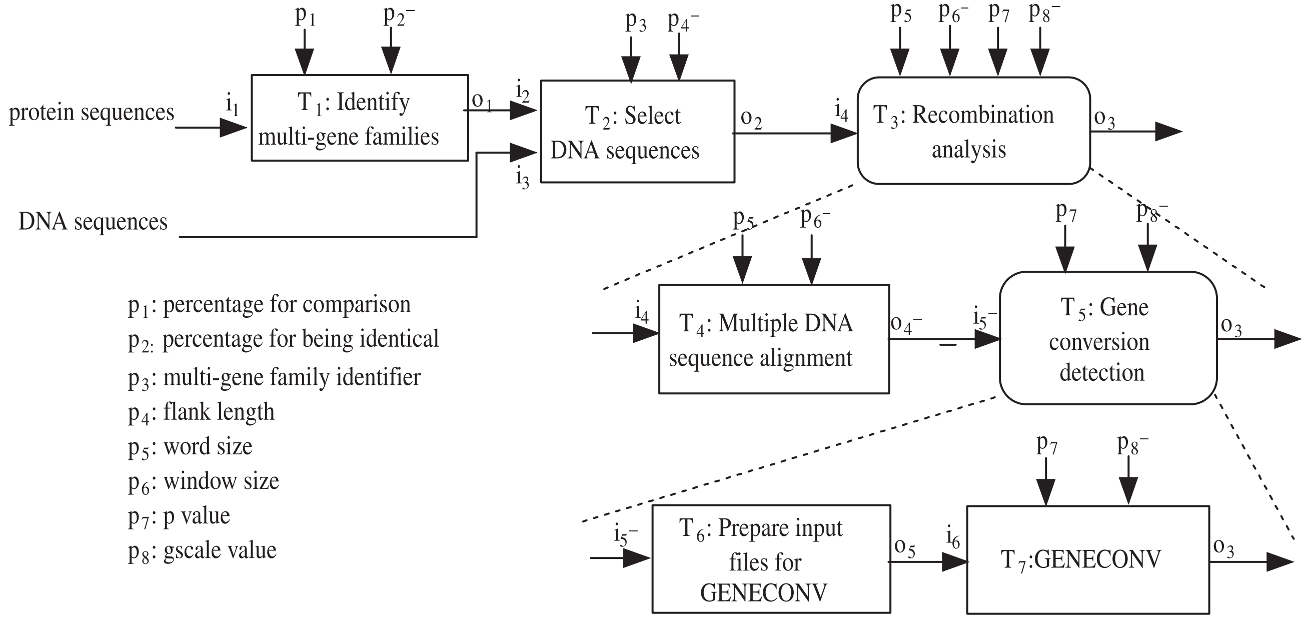
p1
p2−
p3
p4−
p5
p6−
p7
p8−
protein sequences
i1
T1: Identify multi-gene families
o1
i2
T2: Select DNA sequences
i3
o2
DNA sequences
i4
T3: Recombination analysis
o3
T4: Multiple DNA sequence alignment
o4−
i5−
T5: Gene conversion detection
T6: Prepare input files for GENECONV
o5
i6
T7:GENECONV
p1: percentage for comparison
p2: percentage for being identical
p3: multi-gene family identifier
p4: flank length
p5: word size
p6: window size
p7: p value
p8: gscale value

Fig. 1. An intragenomic recombination analysis scientific workflow.

workflow that consists of over 50 workflow tasks [10] and serves as a running example in this work. For a given genome, this workflow takes its protein sequences and identifies all its multigene families (T_1). A particular multigene family is then selected by the user and its associated DNA sequences are retrieved (T_2). Then, a recombination analysis is performed on the retrieved sequences (T_3), which consists of two steps: a multiple DNA sequence alignment step (T_4) and a gene conversion detection step (T_5); the latter is implemented by an off-the-shelf program GENECONV with an input data file preparation step (T_6). As shown in the figure, a scientific workflow consists of a set of workflow tasks, workflow inputs, workflow outputs, and data channels that connect them. Each task represents a computational or analytical step of a scientific process. A task has input ports and output ports that provide the communication interface to other tasks. Tasks are linked together into a workflow as a graph via data channels. During workflow execution, tasks communicate with each other by passing data via their ports through data channels. Finally, a task can have an arbitrary number of input parameters (special kind of input ports), which are used by a scientist to configure its dynamic execution behavior. In the workflow, $p_1, \ldots, p_8$ are input parameters whose meanings are described in the figure. The workflow is hierarchical: composite task T_3 consists of atomic task T_4 and composite task T_5, which, in turn, consists of atomic tasks T_6 and T_7.

This workflow can be executed many times for different genomes or for the same genome but with different

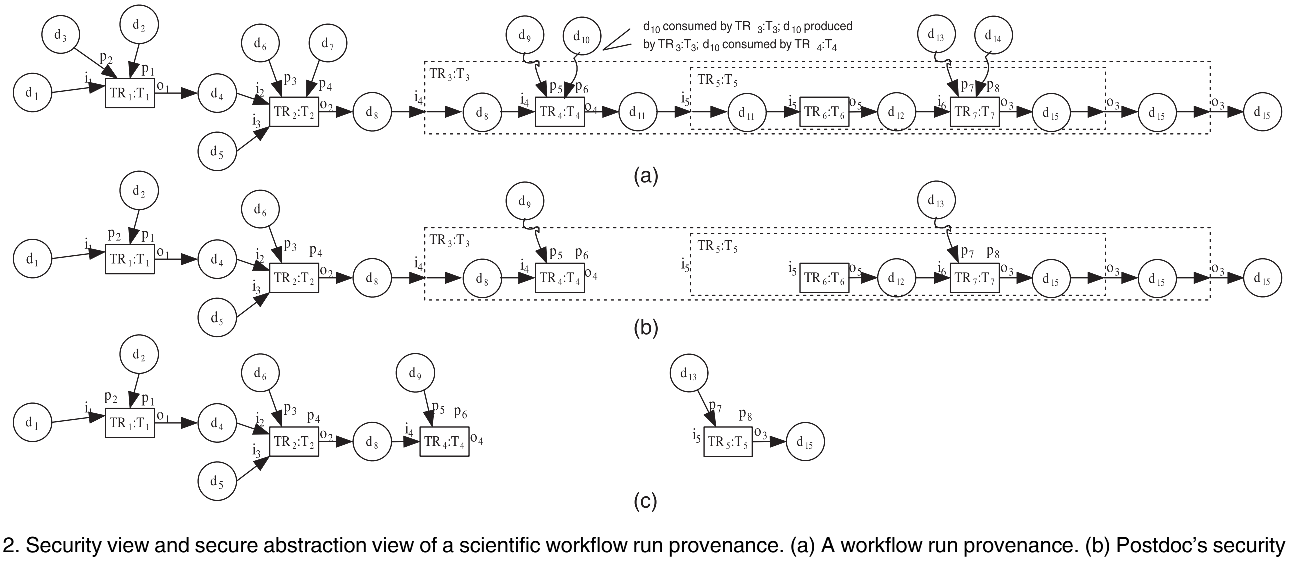

Fig. 2. Security view and secure abstraction view of a scientific workflow run provenance. (a) A workflow run provenance. (b) Postdoc's security view of the workflow run provenance. (c) Postdoc's one secure abstraction view of the workflow run provenance.

parameter settings, resulting in vast amounts of data products and provenance information. Fig. 2a shows a sample scientific workflow run of the workflow in Fig. 1. There are two kinds of nodes: circles represent data products (labeled with data product identifiers) and rectangles represent workflow task runs (labeled with task run and corresponding task identifiers). An edge from a data product to a task run represents a *consume* relationship, while an edge from a workflow task to a data product represents a *produce* relationship. Fig. 2a shows the most detailed workflow run provenance information that will be recorded by a scientific workflow provenance system.

There are several cases related to provenance security that we collected from a domain scientist involved in the project. First, the scientist might publish both the source data used for recombination analysis as well as discovered recombination patterns to the public, but not the recombination analysis workflow itself in order to maintain competitiveness among peers. Second, the scientist might release most parts of the workflow, but protect a critical workflow step, say parameter settings of GENECONV. Therefore, the public will not know which parameter values to use to derive a particular recombination pattern with the GENECONV analysis tool. In this way, the scientist can show the workflow to a stakeholder to establish collaboration relationship, but not release the GENECONV step parameters to ensure intellectual property. Finally, the scientist can choose to protect the *wasDerivedFrom* dependency relationship of "pattern X was derived from bacterial genome data Y"; as a result, even though both the source data and recombination pattern X are released, the public cannot infer from which (among over 400) bacterial genome data a particular discovery X was derived.

Following our example, suppose both data products and their provenance information are sensitive and there is a role called *Postdoc* that can access everything except p_2, p_4, p_6, p_8, o_4, i_5, and the dependency induced by data channel from o_4 to i_5; this is specified by a "$-$" annotation on them. The "$-$" annotations on ports imply that data products consumed or produced by them should not be visible to a user. On the other hand, the "$-$" annotation on the data channel does not allow a user to see that data product d_{12} was derived from data product d_8 (see Fig. 2a). Therefore, *Postdoc* can only see a security view of the provenance—a reduced view of the provenance based on the security restrictions imposed on this role. The security view of the provenance *Postdoc* can access, as shown in Fig. 2b in which data products d_3, d_7, d_{10}, d_{14}, d_{11}, and d_{11}'s incoming and outgoing edges are eliminated. Finally, since a user typically browses a workflow run provenance at a particular abstraction level at a time, she will see a secure abstraction view—a reduced view of the provenance where composite tasks can be seen as black boxes and security restrictions are enforced. A secure abstraction view of the above provenance is shown in Fig. 2c for *Postdoc*, in which $TR_5 : T_5$ is viewed as a black box but $TR_3 : T_3$ is viewed as a composition of $TR_4 : T_4$ and $TR_5 : T_5$.

The main contributions of this paper are

1. a formal scientific workflow provenance model as the basis for querying and access control for workflow provenance;
2. a security model for fine-grained access control for multilevel provenance and an algorithm for the derivation of a full security specification based on inheritance, overriding, and conflict resolution;
3. a formalization of the notion of security views and an algorithm for security view derivation; and
4. a formalization of the notion of secure abstraction views and an algorithm for its computation.

A prototype called SECPROV has been developed, and experiments show the effectiveness and efficiency of our approach.

Organization. The rest of the paper is organized as follows: Section 2 discusses related work. Section 3 presents a formal model for scientific workflow provenance. In Section 4, we present an access control mechanism for scientific workflow provenance. Sections 5 and 6 present the notions of security view and secure abstraction view of provenance, respectively. The implementation details, performance study, and concluding remarks appear in Sections 7, 8, and 9, respectively.

2 Related Work

The importance and requirements of security have been well understood in business workflows [7], [8], [9]. Much work has been done in authentication [11], authorization [12], [13], [14], data privacy, and secure workflow models [15], [16], [17]. While process integrity is ensured by constrained planning [18], [19], [20], data confidentiality is often supported by integrating Role-Based Access Control [21] in the enactment system [22], [23], [15]. Security requirements can be either managed by the workflow system itself [24], [25], or enforced outside of the workflow engine [26], [27].

While execution logs are maintained in business workflows, a richer set of provenance information is collected and maintained in a scientific workflow management system [28], [29], [30], [31], [32] for the purpose of supporting scientific discovery reproducibility, result interpretation, and problem diagnosis [4], [5]. Provenance in scientific workflows has become increasingly important as evidenced by the recent provenance challenge workshops [33].

Although security issues for provenance have been recognized by a few researchers [34], [35], [36], these issues are still open problems. While several access control mechanisms have been proposed for business workflows [7], they are insufficient for scientific workflow provenance for the following reasons:

1. they do not support the restriction of access to the dependency relationships between data products in scientific workflow provenance;
2. they have not considered different levels of granularity for workflow provenance, including

workflows, tasks, ports, data channels, and their containment and inheritance relationships;

3. they have not considered the interaction of access control and abstraction; the latter is used for viewing provenance at different abstraction levels as scientific workflows can be hierarchical; and
4. they have not considered the so-called *data channel constraint* introduced by a scientific workflow specification, which requires that the two ports connected by a data channel must have the same accessibility mode, i.e., either a role can access both ports of a data channel or neither.

Our work uses inheritance to derive security specification for provenance at various levels, which is different from the projection inheritance used in business workflows [37]: 1) they enforce access control using private workflows, but provenance access control is not part of their framework; 2) their projection inheritance is used for the detection of anomalies (deadlocks and livelocks), while our inheritance is used for the derivation of security specification; and 3) their projection inheritance is defined between an interorganizational workflow and its private workflows, but our inheritance is defined between tasks and subtasks for the provenance produced by a hierarchical scientific workflow.

Provenance management is closely related to SOA. First, web services are one major category of building blocks for scientific workflows [38], and therefore, scientific workflow provenance must subsume provenance information that is captured at the service level, called *service provenance* [39], [40], [41]. Second, provenance management should become a major functionality of SOA, leading to the notion of *provenance-aware service-oriented architecture* [42], [43], [44]. Provenance captures the origins and routes of data via service invocation, interaction, and collaboration in an SOA system. Hence, provenance tracking is critical to ensure the security, reliability, and integrity of an SOA system [3], [43]. Finally, a provenance system can be SOA-based, leading to highly interoperable and discoverable provenance subsystems that can be reused in various SOA-based systems, including SOA-based scientific workflow management systems, such as VIEW [45]. Three representative SOA-based provenance systems are Karma [2], PReServ [46], [47], and RDFPROV [48]. RDFPROV is used as the provenance subsystem for the SOA-based VIEW system. The importance of provenance security in SOA is discussed in [3], [34]. In [3], a multilevel data provenance security framework for SOA is proposed; the security model that we propose in this work can serve as one of the building blocks for that higher level framework.

Our scientific workflow provenance model is closely related to the Open Provenance Model (OPM) [49], a general-purpose provenance model that is currently under active development. However, a security model for provenance is not part of OPM, and our security model can be easily adapted to OPM once it is standardized. Our notion of abstraction views is closely related to the notion of user views introduced in [50], [51]. However, user views do not support security annotation and enforcement at the task, port, and data channel level, and do not support security views and secure abstraction views that are introduced in this paper.

Finally, the notion of view is well known in databases [52], [53]. While both database view and provenance view provide a restricted access to underlying data, they significantly differ in two aspects. First, in databases, a view is defined based on a set of flat tables and consists of a stored query accessible as a virtual table composed of the result set of a query; for provenance, a view is defined based on a hierarchical provenance graph and consists of the specification of level of details for each provenance node. Therefore, we need to consider inheritance and conflict resolution for security specification for the provenance security model, while such concerns are not raised for database views. Second, security specification for database views is typically course-grained, either the whole view is accessible to a group of users or it is not. On the other hand, security specification for provenance views is fine-grained, it is possible that one part of an abstraction view is accessible while another part is not, and thus, leading to the notion of secure abstraction view. The notion of secure abstraction view has not been proposed in the database literature.

This work extends our conference paper [54] with the improved security model, algorithms for full security specification, security view, abstraction view derivation, algorithm time complexity analysis, and performance study.

3 Scientific Workflow Provenance Model

To define a security model for scientific workflow provenance, we need to formalize basic components of a scientific workflow, such as atomic task and composite task, that are used for security constraints specification. In addition, we need to have precise notions of task run and workflow run provenance to enforce a security specification. Even though OPM [49] is emerging as a community standard for modeling scientific workflow provenance, it is not sufficient for our purpose. In particular, OPM provides no definitions for a workflow or task, which are important to our security model as security annotations are specified on a workflow rather than on a provenance graph. Besides that, OPM is very dynamic and its notions may evolve. Therefore, we formalize a model for scientific workflow provenance by defining the notions of atomic task, composite task, task run, and workflow run provenance, which is sufficient for our security model, and discuss how it can accommodate OPM at the end of this section.

Definition 3.1 (Atomic task). *An* atomic task *T is a tuple (tid, $\mathcal{IP}$, $\mathcal{OP}$), where tid is T's unique identifier, $\mathcal{IP} = \{i_1, i_2, \ldots, i_m\}$ is the set of input ports of T, and $\mathcal{OP} = \{o_1, o_2, \ldots, o_n\}$ is the set of output ports of T.*

We use $T.i_j$ and $T.o_k$ (or simply i_j and o_k when it is clear from the context) to denote the input port i_j and the output port o_k of T, respectively. Given a set of tasks $\mathcal{T}$, we use

$\mathcal{T}.\mathcal{IP}$ and $\mathcal{T}.\mathcal{OP}$ to represent the union of sets of input and output ports of tasks in $\mathcal{T}$, respectively. An example of an atomic task is T_1 in Fig. 1.

Definition 3.2 (Composite task). *A* composite task *(or* subworkflow*) T is a tuple (tid, $\mathcal{IP}$, $\mathcal{OP}$, $\mathcal{T}$, $\mathcal{DC}_{in}$, $\mathcal{DC}_{out}$, $\mathcal{DC}_{mid}$), where tid is T's unique identifier, $\mathcal{IP} = \{i_1, i_2, \ldots, i_m\}$ is the set of input ports of T, $\mathcal{OP} = \{o_1, o_2, \ldots, o_n\}$ is the set of output ports of T, $\mathcal{T}$ is the set of connected constituent tasks of T, each of which is either atomic or composite, $\mathcal{DC}_{in} : \mathcal{IP} \rightarrow \mathcal{T}.\mathcal{IP}$ is an inverse-functional one-to-many mapping of T with each $(T.i_j, t_2.i_k) \in \mathcal{DC}_{in}$ representing the data channel from input port i_j of T to input port i_k of some task $t_2 \in \mathcal{T}$, $\mathcal{DC}_{out} : \mathcal{T}.\mathcal{OP} \rightarrow \mathcal{OP}$ is an inverse-functional one-to-many mapping of T with each $(t_1.o_j, T.o_k) \in \mathcal{DC}_{out}$ representing the data channel from output port o_j of some task $t_1 \in \mathcal{T}$ to output port o_k of T, and $\mathcal{DC}_{mid} : \mathcal{T}.\mathcal{OP} \rightarrow \mathcal{T}.\mathcal{IP}$ is an inverse-functional one-to-many mapping of T with each $(t_1.o_j, t_2.i_k) \in \mathcal{DC}_{mid}$ representing the data channel from output port o_j of some task $t_1 \in \mathcal{T}$ to input port i_k of some task $t_2 \in \mathcal{T}$. We denote $\mathcal{DC} = \mathcal{DC}_{in} \cup \mathcal{DC}_{out} \cup \mathcal{DC}_{mid}$.*

For any $t \in \mathcal{T}$, we have $T \supset t$, representing that T immediately contains t. For a given workflow W, a list of tasks $T_1, T_2, \ldots, T_n$ is called a *full containment path* of W iff $T_1, \ldots, T_{n-1}$ are composite and T_n is atomic, and we have $W = T_1 \supset T_2 \supset \cdots \supset T_n$. Our definition requires that all constituent tasks to be connected, but does not require each input or output port be an endpoint of a data channel. Such ports are used as input or output parameters that interact with users directly. An example of a composite task is T_3 in Fig. 1.

Each execution of a scientific workflow produces a *workflow run provenance*, which archives the derivation history of data products. A task might be used in several parts of a scientific workflow. Such a task might get executed multiple times in a particular workflow execution. Each execution of a task T is called a *task run* and is assigned with a unique task run identifier in the form of $TR_i : T$; see Fig. 2 for examples. We formalize the notion of workflow run provenance as follows:

Definition 3.3 (Workflow run provenance). *A* workflow run provenance *is a tuple (WR, W, $\mathcal{D}$, $\mathcal{TR}$, Consume, Produce), where WR is the unique identifier of the workflow run provenance corresponding to a particular execution of workflow W, $\mathcal{D}$ is the set of all the data products consumed or produced during the execution of the workflow, $\mathcal{TR}$ is the set of all the task runs occurring in the execution and each $tr \in \mathcal{TR}$ corresponds to a unique task T contained in W, Consume is the relationship set with each $(d, tr.p) \in Consume$ representing that port p of task run tr consumed data product d, and Produce is the relationship set with each $(tr.q, d) \in$ Produce representing that port q of task run tr produced data product d. Given two task runs, tr_1 of task T_1 and tr_2 of task T_2, we can have $(tr_1.q, d) \in Produce$ and $(d, tr_2.p) \in Consume$ only if $(T_1.q, T_2.p) \in W.\mathcal{DC}$.*

A workflow run provenance graph is acyclic. While data channels in $\mathcal{DC}_{in}$ and $\mathcal{DC}_{out}$ pass existing data products without changing them, the graph has distinguished nodes for such data products. For example, in Fig. 2a, two nodes with labels d_8 denote that both data products have the same value which can be stored only once. While the first d_8 is passed between $TR_2 : T_2.o_2$ and $TR_3 : T_3.i_4$, the second d_8 is passed between $TR_3 : T_3.i_4$ and $TR_4 : T_4.i_4$, which is explicitly shown in the figure.

Definition 3.4 (Data dependency). *Given two data products d_1 and d_2 ($d_1 \neq d_2$), we say that d_2 directly depends on d_1 iff there exists a task run tr such that $(d_1, tr.p) \in Consume$, and $(tr.q, d_2) \in Produce$. We say that d_2 depends on d_1 iff d_2 directly or transitively depends on d_1.*

Since provenance captures the history of a scientific experiment and records workflow execution trails that happened in the past, all input, intermediate, and final data products, parameters interactively supplied by a scientist, and data dependencies are precisely defined in a provenance graph. While some dependencies may be known only at workflow runtime, they appear to be static and immutable in recorded provenance. The presented workflow run provenance model captures provenance at various levels of abstraction and granularity: *Consume* and *Produce* dependency information are collected for all levels of a composite task or workflow, and provenance is collected for task runs, ports, and data channels (by *Consume* and *Produce*). Such a scientific workflow provenance model provides the basis for querying and access control of provenance at different levels of abstraction and granularity.

Finally, we show that the presented model has counterparts in OPM, such as a task run corresponds to a process, a data product corresponds to an artifact, an abstraction level corresponds to an account, and *consume*, *produce*, and *depends* relationships correspond to *used*, *wasGeneratedBy*, and *wasDerivedFrom* relationships, respectively. If the appropriate terminology is used, the provenance graph in Fig. 2a can be a valid provenance representation in OPM. Therefore, our workflow run provenance model can straightforwardly adapt to OPM. However, since OPM provides no means to model the scientific workflow in Fig. 1, which is required in our work for security specification, we introduce the corresponding definitions of atomic and composite tasks.

4 Security Model

In this section, we propose a Role-Based Access Control (RBAC) for scientific workflow run provenance. Using our access control, one can not only impose restriction on access to data products consumed and produced during a workflow execution, but also impose restriction on access to the dependency relationships among the data products. When a workflow is designed, a system security administrator provides a security specification for each role of users in the system, in which a role is typically a job function or a position in an organization (e.g., PI, CoPI, Postdoc, etc.). We propose three levels of security specification, namely, *task level*, *port level*, and *data channel level*.

4.1 Task-Level Security Specification

A task can be annotated with $+$ or $-$. An annotation of $+$ for task T specifies that all tasks, ports, and data channels contained in T are accessible unless a $-$ annotation is further specified or derived for them. An annotation of $-$ for task T specifies that all tasks, ports, and data channels contained in T are inaccessible under all circumstances (no overriding is allowed). The administrator specifies security annotations for all or some of the tasks in the following way: $<$*Task* t, *Role* r, and *security annotation* $a>$, where a can be either $+$ or $-$. We call a set of such security annotations as a *task-level security specification* and denote it as TL. Any task that has no security annotation in TL inherits the annotation of the nearest containing ancestor that has a security annotation. At the top level, the annotation of a whole workflow can be set to a default annotation, either $+$ or $-$; we use $+$ as the default annotation for a workflow in this paper. The annotation of a task can be calculated using function $getTaskSecAnnot$ as outlined in Algorithm 1.

Algorithm 1. Algorithm for calculating task sec. annotations

```
function getTaskSecAnnot
input: Task $t$, Role $r$, workflow specification $W$,
  security specification $TL$
output: Task security annotation $<t, r, a>$
if there exists $<t, r, a>$ in $TL$ /*annotation is
  explicit*/ then
    if $a = +$ and there exists task $T$ that contains $t$ and
      $<T, r, -\!>$ is in $TL$ /*+ for $t$ conflicts with − for $T$*/
    then return inconsistency report end if
    return $<t, r, a>$
end if
if $t$ is workflow $W$ then return $<t, r, +\!>$ end if
  /*default annotation*/
Let $t_p$ be a composite task that immediately contains
  $t$ ($t_p \supset t$)
$<t_p, r, a_p>$ = getTaskSecAnnot($t_p$, $r$, $W$, $TL$)
return $<t, r, a_p>$ /* inheritance from a parent in a task
  hierarchy */
end function
```

4.2 Port-Level Security Specification

A port can be annotated with $+$ or $-$. An annotation of $+$ or $-$ for port p of task T specifies that all the data products consumed or produced by $T.p$ from all workflow runs of the workflow are accessible or inaccessible, respectively. The administrator specifies security annotations for all or some of the ports in the following way: $<$*Port* p, *Role* r, and *security annotation* $a>$, where a can be either $+$ or $-$. We call a set of such security annotations as a *port-level security specification* and denote it as PL. Any port that has no security annotation in PL inherits the annotation of its owning task. The annotation of a port can be calculated using function $getPortSecAnnot$ outlined in Algorithm 2.

Algorithm 2. Algorithm for calculating port sec. annotations

```
function getPortSecAnnot
input: Port $p$, Role $r$, workflow specification $W$, sec.
  specifications $TL$ and $PL$
output: Port security annotation $<p, r, a>$
let $t$ be the owning task of $p$
$<t, r, a_t>$ = getTaskSecAnnot($t$, $r$, $W$, $TL$)
if there exists $<p, r, a>$ in $PL$ /*annotation is
  explicit*/ then
    if $a = +$ and $a_t = -$ then return inconsistency report
      /*+ for $p$ conflicts with − for $t$*/ end if
    return $<p, r, a>$
end if
return $<p, r, a_t>$ /*inheritance from a task that
  $p$ belongs to*/
end function
```

4.3 Data-Channel-Level Security Specification

A data channel can be annotated with $+$ or $-$. An annotation of $+$ or $-$ for data channel $dc = (T_1.q, T_2.p)$ specifies that the dependency between a data product produced by $T_1.q$ and a data product consumed by $T_2.p$ from an execution of the workflow is accessible or inaccessible, respectively. In our security model, we require that both ports of a data channel need to have the same accessibility. When both ports are accessible, the data channel must also be accessible. When both ports are inaccessible, the data channel (dependency) can be specified as either accessible or inaccessible. The administrator may specify security annotations for all or some of the data channels in the following way: $<$*Data channel* (p_1, p_2), *Role* r, and *security annotation* $a>$, where a can be either $+$ or $-$. We call a set of such security annotations as a *data-channel-level security specification* and denote it as DL. The annotation of a data channel dc that has no security specification in DL can be derived as follows: 1) if both ports are accessible or inaccessible, then dc is accessible or inaccessible, respectively; and 2) if one port is accessible while the other is not, then the specification is not consistent. Other derivation rules can be used in practice to produce variants of our security model. In summary, the annotation of a data channel can be calculated using function $getDataChannelSecAnnot$ outlined in Algorithm 3.

Algorithm 3. Algorithm for calculating data channel security annotations

```
function getDataChannelSecAnnot
input: Data channel $(p_1, p_2)$, Role $r$, workflow $W$, spec.
  $TL$, $PL$, and $DL$
output: Data channel security annotation
  $<(p_1, p_2), r, a>$
$<p_1, r, a_1>$ = getPortSecAnnot ($p_1$, $r$, $W$, $TL$, $PL$)
$<p_2, r, a_2>$ = getPortSecAnnot($p_2$, $r$, $W$, $TL$, $PL$)
if $a_1 \neq a_2$ then return inconsistency report /*two ports
  of a data channel should have the same annotations*/
  end if
if there exists $<(p_1, p_2), r, a>$ in $DL$ /*ann. is
  explicit*/ then
```

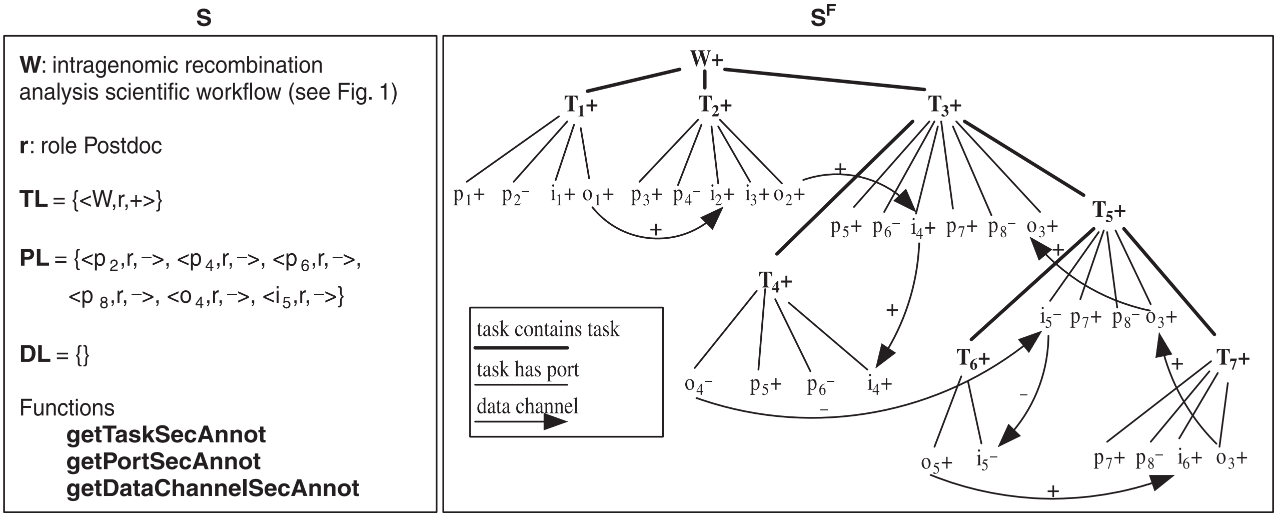

Fig. 3. Security specification S and full security specification S^F for intragenomic recombination analysis scientific workflow and role *Postdoc*.

```
        let $t_1$ ($t_2$) be the owning task of $p_1$ ($p_2$)
        let $t$ be the nearest containing ancestor of both $t_1$
          and $t_2$
        $<t, r, a_t> =$ getTaskSecAnnot($t$, $r$, $W$, $TL$)
        if ($a = +$ and $a_t = -$) or ($a = -$ and $a_1 = +$) then
            return inconsistency report /*+ for $(p_1, p_2)$
              conflicts with $-$ for $t$; $-$ for $(p_1, p_2)$ conflicts with
              $+$ for $p_1$ or $p_2$*/
        end if
        return $<(p_1, p_2), r, a>$
    end if
    return $<(p_1, p_2), r, a_1>$ /* inheritance from ports */
end function
```

4.4 Full Security Specification

The three defined functions, *getTaskSecAnnot*, *getPortSecAnnot*, and *getDataChannelSecAnnot*, have quadratic theoretical worst-case time complexity when sets are implemented using hash tables; however, in practice, the algorithms are very efficient as the set lookup operation takes constant time on average. These functions answer the question of how full security specifications for a workflow W can be derived from partial security specifications TL, PL, and DL for W. In the following, we refer a tuple (W, TL, PL, and DL) as a *security specification* S. Furthermore, we denote *full security specification* that contains explicit security annotations for all tasks, ports, and data channels in a given workflow as S^F, and S^F can be easily derived from S in $O(n^3)$ time as outlined in Algorithm 4. A slight modification of this algorithm to process a task hierarchy in level-order and "remember" task annotations, rather than recalculating them, can achieve quadratic performance.

Algorithm 4. Algorithm for calculating full sec. specifications

```
function calculateFullSecuritySpecification
input: Role $r$, workflow spec. $W$, sec. specs $TL$, $PL$,
  and $DL$
output: Full security specification $S^F$
let $S^F$ be an empty set
for each task $t$ in $W$ do
    $<t, r, a> =$ getTaskSecAnnot ($t$, $r$, $W$, $TL$)
    add $<t, r, a>$ to $S^F$ end for
for each port $p$ in $W$ do
    $<p, r, a> =$ getPortSecAnnot ($p$, $r$, $W$, $TL$, $PL$)
    add $<p, r, a>$ to $S^F$ end for
for each data channel $(p_1, p_2)$ in $W$ do
    $<(p_1, p_2), r, a> =$ getDataChannelSecAnnot
  (($p_1$, $p_2$), $r$, $W$, $TL$, $PL$, $DL$)
    add $<(p_1, p_2), r, a>$ to $S^F$ end for
return S $^F$
end function
```

In Fig. 3, we show security specification S and full security specification S^F for our sample scientific workflow W (see Fig. 1) and role *Postdoc*. S^F is represented as a graph and is computed by calling the corresponding functions on each task, port, and data channel of W.

4.5 Consistent Security Specification and Inference Problems

To check the consistency of a security specification, we define it formally below.

Definition 4.1 (Consistent security specification). *A security specification (W, TL, PL, DL) is consistent if and only if*

1. *there is no task, port, and data channel in W, such that it has an annotation or derived annotation of both $+$ and $-$;*
2. *for any task T, if T is annotated with $-$, then there does not exist any task T' contained in T, such that T', or some port or data channel of T' is annotated with $+$;*
3. *the two ports associated with each data channel $(T_1.q, T_2.p)$ must have the same accessibility; and*
4. *for each data channel $(T_1.q, T_2.p)$, if both end ports are accessible, then the data channel must also be accessible.*

The first constraint ensures that no ambiguity for accessibility will rise in our security model. The second constraint enforces the semantics of a $-$ annotation for a task T: all tasks contained in T and their ports and data channels are inaccessible under all circumstances. The third constraint, called *data channel constraint*, restricts that the two ports connected by any data channel must have the same security annotations. The rationale for this constraint is to avoid unintentional permission of accessing a sensitive data product in the situation in which the data product is accessible via one port (with $+$ annotation) and inaccessible via another port (with $-$ annotation) of some data channel. The fourth constraint ensures that if one needs to enforce the inaccessibility of a data channel, then the inaccessibility of both ports needs to be enforced to prevent an inference of the dependency by comparing data products accessible from the two ports.

While we define these four constraints in our security model as default constraints, sometimes, it may be interesting to relax some of them or add new ones. For example, eliminating the data channel constraint will allow annotating one port of a data channel as accessible and another port as inaccessible. In this situation, the data product that is passed via the channel is visible through the accessible port, and therefore, not protected. However, the corresponding produce or consume relationship is hidden from a user due to the inaccessible port, and thus, provenance information regarding which task run produced or consumed the data product is inaccessible. On the other hand, it may be desirable to add a new constraint, such as this one: if output of task T_1 is inaccessible, then output of task T_2 must also be inaccessible, because, for example, it is known that T_2 takes the output of T_1 and simply converts it into another format without the content modification, and therefore, revealing T_2's output is equivalent to revealing T_1's output. Overall, in the context of our work, adding or removing a constraint can only result in minor changes of Algorithms 1-3 that are related to security specification consistency checking. In the following, we always use the four constraints stated in Definition 4.1.

Our algorithms (Algorithms 1-4) are correct in the sense that only consistent security specifications will be allowed. First, for a task, either it is explicitly annotated with $+$ or $-$, or it will inherit the annotation of its nearest ancestor which has an explicit annotation. Moreover, constraint 2 is checked. As a result, the annotation of a task is always consistent. Second, for a port of a task T, either it is explicitly annotated with $+$ or $-$, or it will inherit the annotation of T. Therefore, the guarantee of the consistency of a task annotation, as well as the constraint 2 check, ensures the consistency of port annotation. Finally, for a data channel $(T_1.q, T_2.p)$, either it is explicitly annotated with $+$ or $-$, or it will inherit the accessibility of its end ports. In both cases, constraints 2-4 are checked to ensure the consistency of the specification.

Our sample security specification (see Fig. 3) for the intragenomic recombination analysis scientific workflow is consistent, since all the constraints hold for the derived full security specification. When our algorithms encounter an inconsistent security specification that violates one or more of the consistency constraints, the administrator is notified about the problem and is required to change the security specification.

To complete our security analysis, we discuss the following seven possible inference issues:

1. an output (input) of a task is inaccessible, but can be inferred if the same output (input) of its subtask is accessible;
2. an output (input) of a task is inaccessible, but can be inferred if the same output (input) of its parent task is accessible;
3. a data product is inaccessible, but can be inferred if it is inaccessible through one port of a data channel but accessible through the other port;
4. a data channel and its related data dependencies are inaccessible, but can be inferred if both ports of the data channel are accessible as they produce and consume the same data product;
5. an output of a task is inaccessible, but can be inferred by executing the task if the task is available and all its inputs are accessible;
6. an input of a task is inaccessible, but can be inferred by executing the inverse function of the task if such a function is available and all outputs are accessible; and
7. the functionality of a task is inaccessible, but can be inferred if all or some inputs and outputs of the task are accessible.

The first two inferences are only possible if the same data product is passed between a task and its subtask via a data channel with different security annotations on their ports, which is a special case of inference 3. Therefore, in a consistent security specification, inferences 1-3 are prevented by constraint 3 in Definition 4.1. Case 1 is impossible because of constraint 2. The condition for inference 4 cannot be true due to constraint 4. While the first four inferences are solely based on provenance information, inferences 5 and 6 involve execution of a task or its inverse, which is beyond the capability of a provenance system. Such inferences can be addressed with an additional access control for workflows and tasks. Finally, inference 7 is about task functionality rather than provenance, and hence, cannot be solved in our security model for provenance. In summary, our model can effectively prevent inferences 1-4. It is also straightforward to check conditions for inferences 5-7 to prompt an administrator to examine these issues.

4.6 RBAC Administration

Administration in the proposed RBAC can rely on existing and well-understood approaches, such as Administrative RBAC (ARBAC) [55], [56] and Scoped Administration of RBAC (SARBAC) [57]. Both administration models support administrative and normal roles, as well as role hierarchies. They specify which administrators can assign which roles to which users, which administrators can assign which permissions to which users, and which administrators can

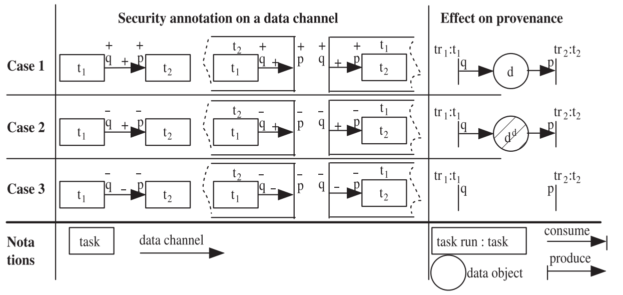

Fig. 4. Implication of security annotations of a data channel and its associated ports on provenance accessibility.

specify access control policies for which parts of a workflow. With a role hierarchy in place, a role, such as *Postdoc*, can be an implicit member of its junior role, say *Investigator*, and hence, has all permissions of the junior role. Security specifications can first be assigned to roles that do not have any junior roles. Afterward, the administrators can provide specifications for roles whose junior roles have already been considered. Such roles automatically inherit the permissions of their junior roles and such permissions cannot be removed unless the permissions are removed from their junior roles. The administrators can provide more permissions for such roles. While ARBAC and SARBAC are responsible for managing access control policies and granting access to subjects in the system, our RBAC is enforced with security views of provenance (and corresponding algorithms) that are discussed next.

5 Security Views of Provenance

Our approach for enforcing security specification for scientific workflow run provenance is based on the innovative notion of *security views*. A security view of provenance is a restricted view of a workflow run provenance consisting of all and only the information that users are authorized to access.

Before we formalize and incorporate the security view notion into our provenance model, consider the implication of security annotations of data channels and their associated ports on the accessibility of provenance. A consistent specification has three cases of security annotations for a data channel and its associated ports, as shown in Fig. 4; other cases lead to inconsistent specifications due to the mandatory data channel constraint. In the first case, both ports and the data channel are annotated with $+$, and therefore, corresponding data product and produce-consume relationships should be accessible. In the second case, both ports are inaccessible, while the data channel is accessible, which implies that the data product is not accessible but the produce-consume relationships are. As our model does not permit a task run to be connected directly to another task run, we replace the data product with a dummy data product d^d with a new unique ID to maintain the relationships without authorizing the access to the data product itself. Finally, in the third case, both the data channel and its associated ports are inaccessible, which implies that both the data product and produce-consume relationships are inaccessible. Therefore, the data product and its associated edges are deleted in the provenance that is returned to a user.

The security view of a workflow run provenance (WR, W, $\mathcal{D}$, $\mathcal{TR}$, *Consume*, and *Produce*) only includes a subset of data products in $\mathcal{D}$, as well as some dummy data products. Similarly, subsets of *Consume* and *Produce* are preserved and augmented with relationships for newly introduced dummy data products. In the following, we outline the security view definition.

Definition 5.1 (Security view). *A* security view *of a workflow run provenance is a tuple (WR, W, $\mathcal{D}'$, $\mathcal{TR}$, Consume$'$, Produce$'$) that is derived from a workflow run provenance (WR, W, $\mathcal{D}$, $\mathcal{TR}$, Consume, Produce) and a consistent full security specification S^F for a user role r, where*

- *$\mathcal{D}' = \mathcal{D}^a \cup \mathcal{D}^d$ is the set consisting of 1) all the data products $\mathcal{D}^a \subseteq \mathcal{D}$ consumed or produced by the workflow run and each $d \in \mathcal{D}^a$ is accessible to r via an accessible port p, i.e., it is true that $<p, r, +> \in S^F$ and $(d, tr.p) \in Consume$ or $(tr.p, d) \in Produce$ for some $tr \in (\mathcal{TR} \cup \{WR\})$, and 2) data products $\mathcal{D}^d$, where each $d \in \mathcal{D}^d$ is a dummy data product with a unique data product ID and each d corresponds to data product $d' \in \mathcal{D}$ that is consumed and produced by inaccessible ports connected by an accessible channel (see Case 2 in Fig. 4), i.e., $(d',\ tr_i.p) \in Consume$, $(tr_j.q,\ d') \in Produce$, (q, p) is a data channel, $<p, r, -> \in S^F$, $<q, r, -> \in S^F$, and $<(p, q), r, +> \in S^F$.*
- *$Consume' = Consume^a \cup Consume^d$ is the relationship set consisting of 1) set $Consume^a$ which is the projection of* Consume *over $\mathcal{D}^a$ and 2) set $Consume^d$ which is the projection of* Consume *over all the data products that have corresponding dummy data products in $\mathcal{D}^d$ and data products in $Consume^d$ are substituted with their dummy versions.*
- *$Produce' = Produce^a \cup Produce^d$ is the relationship set consisting of 1) set $Produce^a$ which is the projection of* Produce *over $\mathcal{D}^a$ and 2) set $Produce^d$ which is the projection of* Produce *over all the data products that have corresponding dummy data products in $\mathcal{D}^d$ and data products in $Produce^d$ are substituted with their dummy versions.*

The security view of a workflow run provenance can be calculated using function deriveSecurityView outlined in Algorithm 5. The algorithm iterates over relationship sets $Consume$ and $Produce$ to construct new sets $\mathcal{D}'$, $Consume'$, and $Produce'$ for the security view. Intuitively, the algorithm checks security annotations on ports and data channels to identify the cases described in Fig. 4 and construct the three new sets accordingly. Handling of Cases 1-3 are marked in the algorithm comments, where no information is added to the security view for Case 3. Situations in which a data product is produced (consumed), but never consumed (produced), are handled in lines 8 and 25, respectively. The algorithm takes $O(n^3)$ time when the S^F set is implemented via a hash table.

Algorithm 5. Algorithm for deriving security views

```
1:  function deriveSecurityView
2:  input: Role $r$, consistent full sec. specification $S^F$,
        workflow run provenance ($WR$, $W$, $\mathcal{D}$, $\mathcal{TR}$, $Consume$,
        $Produce$)
3:  output: Security view ($WR$, $W$, $\mathcal{D}'$, $\mathcal{TR}$, $Consume'$,
        $Produce'$)
4:  let $\mathcal{D}'$, $Consume'$, and $Produce'$ be empty sets
5:  for each $(tr.q, d)$ in $Produce$ do
6:      if $<q, r, +>$ is in $S^F$ /*Case 1*/ then
7:          add $d$ to $\mathcal{D}'$
8:          add $(tr.q, d)$ to $Produce'$
9:      else
10:         let $d^d$ be a dummy data product to represent $d$
11:         for each $(d, tr'.p)$ in $Consume$ do
12:             if $<(q, p), r, +>$ is in $S^F$ /*Case 2*/ then
13:                 add $d^d$ to $\mathcal{D}'$
14:                 add $(tr.q, d^d)$ to $Produce'$
15:                 add $(d^d, tr'.p)$ to $Consume'$
16:             else
17:                 /*Do nothing; data product and relationships
                    are not part of the view*/ /*Case 3*/
18:             end if
19:         end for
20:     end if
21: end for
22: for each $(d, tr.p)$ in $Consume$ do
23:     if $<p, r, +>$ is in $S^F$ /*Case 1*/ then
24:         add $d$ to $\mathcal{D}'$
25:         add $(d, tr.p)$ to $Consume'$
26:     end if
27: end for
28: return ($WR$, $W$, $\mathcal{D}'$, $\mathcal{TR}$, $Consume'$, $Produce'$)
29: end function
```

A sample security view derived using deriveSecurityView for the intragenomic recombination analysis scientific workflow run provenance and *Postdoc*'s security specification (see Fig. 3) is shown in Fig. 2b. For example, data product d_{11} and its corresponding produce and consume relationships are not part of the security view because the test conditions in lines 6, 12, and 23 fail, such as $<o_4, Postdoc, +> \notin S^F$, $<(o_4, i_5), Postdoc, +> \notin S^F$, and $<i_5, Postdoc, +> \notin S^F$. Similarly, data products d_3, d_7, d_{10}, and d_{14}, along with their relationships, are eliminated from the view because they are produced (line 6) or consumed (line 23) by inaccessible ports p_2, p_4, p_6, and p_8, respectively. On the other hand, data products and their relationships recorded via accessible ports are retained in the security view, as shown in Fig. 2b.

6 Secure Abstraction Views of Provenance

Hierarchical structure of scientific workflows, expressed via composite tasks or subworkflows, can serve as an important mechanism for abstraction. While exploring a workflow run provenance, a user may be interested in data products that have been produced or consumed by only certain composite task runs instead of looking into a more detailed view where provenance of their constituent task runs is revealed. Therefore, an abstraction mechanism is important to enable a user to focus on only relevant provenance information. In this section, we outline the notion of abstraction views and introduce a framework that integrates abstraction views and security views into secure abstraction views, such that a user can examine provenance at different abstraction levels while respecting the security specification prescribed for her.

We define an *abstraction view specification* for a scientific workflow and an *abstraction view* of a workflow run provenance in the following:

Definition 6.1 (Abstraction view specification). *Given a workflow W, let $\tau(W)$ be the set of constituent tasks of W at all levels, including W itself. An* abstraction view specification *is a Boolean function $\mathcal{A} : \tau(W) \rightarrow \{F, T\}$ such that for each full containment path $T_1, T_2, \ldots, T_n$ of W, $\mathcal{A}(T_n) = F$ and if for some i $(1 \leq i \leq n-1)$, $\mathcal{A}(T_i) = T$, and $\mathcal{A}(T_{i+1}) = F$, then $\mathcal{A}(T_1) = \mathcal{A}(T_2) = \cdots = \mathcal{A}(T_i) = T$ and $\mathcal{A}(T_{i+1}) = \cdots = \mathcal{A}(T_n) = F$.*

Intuitively, $\mathcal{A}(t) = F$ indicates that task t is abstracted as if it was an atomic task with its internal composition details hidden from a user. $\mathcal{A}(t) = T$ indicates that task t is viewed as a composition of its constituent tasks. An abstraction view specification identifies a particular level of composition details of a workflow that a user can see; $\mathcal{A}(t)$ is always assigned F when t is an atomic task since it has no internal composition details to be seen. Provenance of a task run is relevant (visible) in an abstraction view iff one of the following two conditions for the corresponding task t in W holds: 1) $\mathcal{A}(t) = F$ and $t = W$ (t is the whole workflow) or 2) $\mathcal{A}(t) = F$, t has immediately containing (parent) composite task $t_p \supset t$, and $\mathcal{A}(t_p) = T$.

For example, an abstraction view specification for the workflow W in Fig. 1 that views T_3 as an atomic task and W as the composition of T_1, T_2, and T_3 can be formalized as

$$\mathcal{A}_1 = \{W \rightarrow T, T_1 \rightarrow F, T_2 \rightarrow F, T_3 \rightarrow F, T_4 \rightarrow F, \\ T_5 \rightarrow F, T_6 \rightarrow F, T_7 \rightarrow F\}.$$

In the meanwhile, an abstraction view specification that views T_5 as an atomic task and W as the composition of T_1, T_2, T_4, and T_5 can be formalized as

$$\mathcal{A}_2 = \{W \rightarrow T, T_1 \rightarrow F, T_2 \rightarrow F, T_3 \rightarrow T, T_4 \rightarrow F, \\ T_5 \rightarrow F, T_6 \rightarrow F, T_7 \rightarrow F\}.$$

The implications of abstraction view specification on the provenance information are elaborated in the following definition:

Definition 6.2 (Abstraction view). *An* abstraction view *of a workflow run provenance is a tuple (WR, W, $\mathcal{D}'$, $\mathcal{TR}'$, $Consume'$, $Produce'$) that is derived from a workflow run* *provenance (WR, W, $\mathcal{D}$, $\mathcal{TR}$, Consume, Produce) and an abstraction view specification $\mathcal{A}$ for workflow W, where*

- *$\mathcal{TR}' \subseteq \mathcal{TR}$, such that $tr \in \mathcal{TR}'$ iff $tr \in \mathcal{TR}$, tr executes a task t in W, $\mathcal{A}(t) = F$, and ($t = W$ or ($t_p \supset t$ and $\mathcal{A}(t_p) = T$)).*
- *$\mathcal{D}' \subseteq \mathcal{D}$, such that $d \in \mathcal{D}'$ iff $d \in \mathcal{D}$, d is produced or consumed by a task run $tr \in \mathcal{TR}'$.*
- *$Consume'$ and $Produce'$ are the projections of Consume and Produce, respectively, over $\mathcal{D}'$ and $\mathcal{TR}'$, such that $(d, tr.p) \in Consume$ $((tr.q, d) \in Produce)$ iff $d \in \mathcal{D}'$, $tr \in \mathcal{TR}'$, and $p \in tr.\mathcal{IP}$ $(q \in tr.\mathcal{OP})$.*

The abstraction view of a workflow run provenance can be calculated using function deriveAbstractionView outlined in Algorithm 6. The algorithm iterates over sets $\mathcal{TR}$, *Consume*, and *Produce* to construct new sets $\mathcal{TR}'$, $\mathcal{D}'$, $Consume'$, and $Produce'$ for the abstraction view, such that irrelevant provenance information is filtered out according to a given abstraction view specification $\mathcal{A}$. Conditions in lines 11 and 15 ensure that a task run is viewed as atomic. The algorithm can be implemented to have quadratic time complexity.

Algorithm 6. Algorithm for deriving abstraction views

```
1:  function deriveAbstractionView
2:  input: Abstraction view specification $\mathcal{A}$, workflow run
        provenance ($WR$, $W$, $\mathcal{D}$, $\mathcal{TR}$, Consume, Produce)
3:  output: Abstraction view ($WR$, $W$, $\mathcal{D}'$, $\mathcal{TR}'$, $Consume'$,
        $Produce'$)
4:  let $\mathcal{TR}'$, $\mathcal{D}'$, $Consume'$, and $Produce'$ be empty sets
5:  for each $tr$ in $\mathcal{TR}$ do
6:      let $t$ be a task in $W$ that $tr$ executes
7:      if $\mathcal{A}(t) = F$ and (($t = W$) or (for $t$'s parent $t_p$,
            $\mathcal{A}(t_p) = T$)) then
8:          add $tr$ to $\mathcal{TR}'$ /*task runs preserved in the view*/
9:      end if
10: end for
11: for each $(tr.q, d)$ in Produce, such that $tr$ is in $\mathcal{TR}'$ and
        $q$ is in $tr.\mathcal{OP}$ do
12:     add $d$ to $\mathcal{D}'$ /*preserved data products*/
13:     add $(tr.q, d)$ to $Produce'$ /*preserved produce
            relationships*/
14: end for
15: for each $(d, tr.p)$ in Consume, such that $tr$ is in $\mathcal{TR}'$ and
        $p$ is in $tr.\mathcal{IP}$ do
16:     add $d$ to $\mathcal{D}'$ /*preserved data products*/
17:     add $(d, tr.p)$ to $Consume'$/*preserved consume
            relationships*/
18: end for
19: return ($WR$, $W$, $\mathcal{D}'$, $\mathcal{TR}'$, $Consume'$, $Produce'$)
20: end function
```

Thus, both security and abstraction views are restricted views (aka filters) of a workflow run provenance that include restricted sets of data products, consume relationships, produce relationships, and so forth. To integrate these two kinds of views into our framework, we introduce a novel notion of *secure abstraction view* as follows:

Definition 6.3 (Secure abstraction view). *Let $sv_r^S(WR)$ and $av^{\mathcal{A}}(WR)$ denote operations that compute a security view of workflow run provenance WR for role r and an abstraction view of workflow run provenance WR for user u of role r, respectively, where a security specification S for r and an abstraction view specification $\mathcal{A}$ for u are given. A secure abstraction view of workflow run provenance WR for user u with role r is defined as $av^{\mathcal{A}}(sv_r^S(WR))$ or $sv_r^S(av^{\mathcal{A}}(WR))$.*

Operations $sv_r^S(WR)$ and $av^{\mathcal{A}}(WR)$ can be implemented with our proposed functions deriveSecurityView and deriveAbstractionView. The commutativity of these operations, i.e., $sv_r^S(av^{\mathcal{A}}(WR)) \equiv av^{\mathcal{A}}(sv_r^S(WR))$, should be evident, as briefly explained in the following. On the one hand, $av^{\mathcal{A}}(WR)$ returns a subgraph of a provenance graph WR without altering existing dataflows, such that if a data product in WR is consumed or produced, then the relationships can be only preserved or hidden in $av^{\mathcal{A}}(WR)$. Therefore, $sv_r^S(WR)$ and $sv_r^S(av^{\mathcal{A}}(WR))$ must have the same effect on dataflow-associated provenance relationships (see Fig. 4) that are common in WR and $av^{\mathcal{A}}(WR)$, and all the relationships in $av^{\mathcal{A}}(WR)$ are also in WR. On the other hand, $sv_r^S(WR)$ does not affect task runs in WR, such that $sv_r^S(WR)$ and WR have the same sets of task runs. Therefore, $av^{\mathcal{A}}(WR)$ and $av^{\mathcal{A}}(sv_r^S(WR))$ must filter out the same task runs, along with their consume and produce relationships.

A sample secure abstraction view of the intragenomic recombination analysis scientific workflow run provenance for *Postdoc*'s security specification (see Fig. 3) and the abstraction view specification $\mathcal{A}_2$ (see definition above) is shown in Fig. 2c. It can be computed by first applying deriveSecurityView (see Fig. 2b), and then, applying deriveAbstractionView, or vice versa. For example, to compute the provenance graph in Fig. 2c from the graph in Fig. 2b based on the abstraction view specification $\{W \rightarrow T,\ T_1 \rightarrow F,\ T_2 \rightarrow F,\ T_3 \rightarrow T,\ T_4 \rightarrow F,\ T_5 \rightarrow F,\ T_6 \rightarrow F,\ T_7 \rightarrow F\}$, the deriveAbstractionView function first filters out (line 7) task runs $TR_3 : T_3$, $TR_6 : T_6$, and $TR_7 : T_7$ that are not visible in the abstraction view. The rest, filtering out the corresponding produce and consume relationships, is straightforward such that only relationships that involve ports of the retained task runs are added to the view along with the data products (lines 12-13 and 16-17).

Finally, we outline three approaches to provenance querying with security views and abstraction views (see Fig. 5). In the first, the most natural one, a provenance query q is evaluated over a secure abstraction view $av^{\mathcal{A}}(sv_r^S(WR))$ or $sv_r^S(av^{\mathcal{A}}(WR))$ of provenance. In the second approach, q is evaluated over a workflow run provenance WR and the result is filtered out based on security and abstraction view specifications S and $\mathcal{A}$. In the last approach, q is rewritten into a "security and abstraction view aware" query q', and q' is evaluated over WR. For example, consider the following query q issued by a *Postdoc* user: return task runs that produced data product d_{15} (see Fig. 2). Using the first approach, we can retrieve $TR_5 : T_5$ (see Fig. 2c) directly as

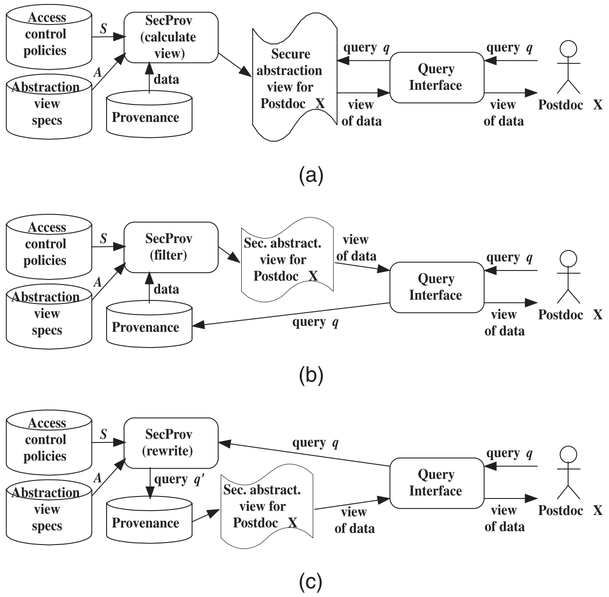

Fig. 5. Secure querying architecture. (a) Preprocessing. (b) Postprocessing. (c) Query rewriting.

the result of the query. Using the second approach, we can retrieve $TR_5 : T_5$, $TR_7 : T_7$, and $TR_3 : T_3$ (see Fig. 2a) and filter out $TR_7 : T_7$ and $TR_3 : T_3$, since the tasks T_7 and T_3 are not part of the *Postdoc*'s abstraction view specification. Finally, using the third approach, we can rewrite q to return task run *tr* that produced data product d_{15}, such that *tr* does not execute T_7 or T_3 and *tr*'s output port is accessible with respect to the security specification. While the first approach will be more efficient if a user is authorized to access only a small portion of the provenance, the second approach can take advantage of indexing techniques to speed up query processing, and the third approach will be the best choice if the security policy for a user will not change frequently as query rewriting can be reused for future access.

7 SECPROV PROTOTYPE AND SERVICES COMPUTING APPLICATIONS

We developed the SECPROV [58] prototype to validate the effectiveness of our approach to secure provenance querying with integrated security views and abstraction views. We used XSB Prolog to implement algorithms for security and abstraction views derivation and Java with the JGraph library to implement a GUI for assigning security and abstraction specifications. In Fig. 6, two screenshots of SECPROV are presented. In the upper one, an abstraction view specification is selected (on the left) based on the task hierarchy of a workflow, such that a task can be "folded" or "unfolded." A workflow (on the right) is annotated at the task, port, and data channel levels to create a security specification. The lower screenshot shows a secure abstraction view of a workflow run provenance (on the right). A user can select different abstraction levels from the left panel to examine different abstraction views of the same workflow run provenance.

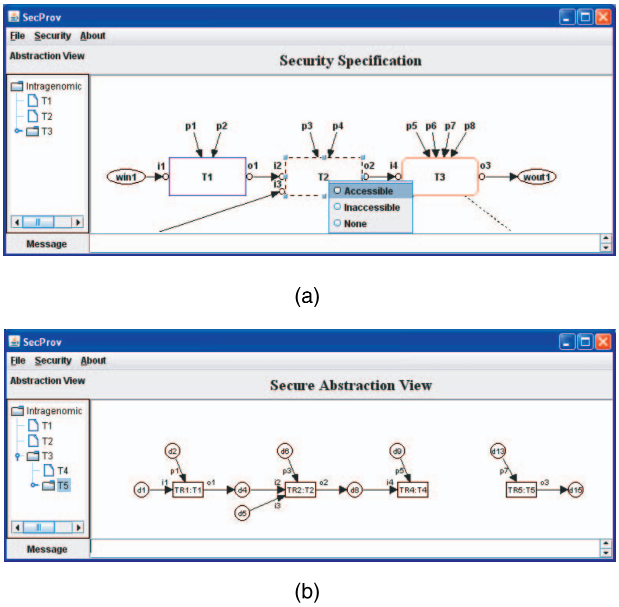

Fig. 6. Screenshots of SECPROV. (a) Abstraction view specification (on the left) and security specification (on the right). (b) Sample secure abstraction view of provenance.

Each abstraction view is *secure* (aka secure abstraction view) in the sense that only accessible provenance information is returned to the user according to the security policy specified for her at that abstraction level. Note that the returned view does not have to be a connected graph; in this case, the dependencies between the subgraphs are hidden from the user.

Our proposed security and abstraction mechanisms are applicable to provenance of any computing system that is composed of loosely coupled components. The most natural examples include workflows, SOA-based systems, and composite web services. While we present our solution in the context of scientific workflows, it applies to provenance-aware SOAs [42], [43] and composite web services [59] as well. In both SOAs and composite web services, constituent services can be composite, and thus, impose multilevel provenance semantics. Abstraction views provide a mechanism for a user to examine service provenance at a simplified abstraction level and secure abstraction views further restrict to expose only authorized provenance information. Our implementation of the secure abstraction views is currently used in VIEW [45], an SOA-based scientific workflow management system. VIEW consists of six loosely coupled autonomous distributed service subsystems, where the *Provenance Manager* is the key subsystem to manage and query scientific workflow provenance. In VIEW, a scientific workflow is composed from tasks, which are abstracted from task components, such as local executables, remote web services, and Grid services [38]. During workflow execution, provenance (including service provenance) is automatically collected and stored in the *Provenance Manager*. Secure abstraction views are used in VIEW to examine service provenance at different abstraction levels in a secure manner.

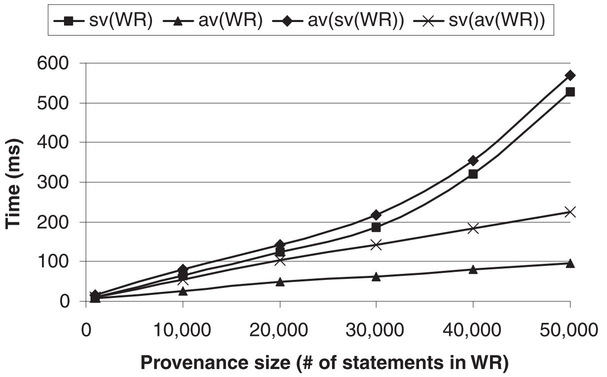

Fig. 7. Performance of algorithms deriveSecurityView and deriveAbstractionView to derive $sv_r^S(WR)$, $av^{\mathcal{A}}(WR)$, $av^{\mathcal{A}}(sv_r^S(WR))$, and $sv_r^S(av^{\mathcal{A}}(WR))$.

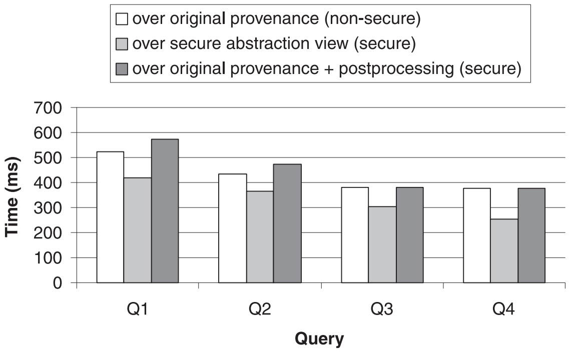

Fig. 8. Performance of provenance queries over original provenance (nonsecure results), its secure abstraction view (secure results), and original provenance with postprocessing to filter out results based on security and abstraction view specifications (secure results).

8 Performance Study

The performance study reported in this section was conducted on a PC with 3.00 GHz Intel Core 2 CPU, 4 GB RAM, and 750 GB disk space running MS Windows XP Professional. While functions deriveSecurityView and deriveAbstractionView were used to either preprocess (materialized secure abstraction views) or postprocess (dynamically computed secure abstraction views) provenance data, sample provenance queries were evaluated with our RDFPROV system [48] that stored workflow run provenance in a MySQL 5.0 CE RDBMS. Provenance was generated by our in-house VIEW workflow engine [45] in the Resource Description Framework (RDF) format according to our provenance model. The VIEW system provides a user-friendly GUI for the design and execution of scientific workflows, as well as the collection and management of scientific workflow provenance; more details on VIEW can be found in [45].

The performance of functions deriveSecurityView and deriveAbstractionView is presented in Fig. 7. We used a workflow with 12 tasks and two loops, which allowed us to generate workflow run provenance documents of different sizes. The provenance size was measured as the number of statements in a document (e.g., the total number of task runs, data products, and consume and produce relationships). Our abstraction and security view specifications were fixed for a user and user's role, respectively. The security view filtered out ≈20 percent of its input by restricting access to some relationships in one of the loops. The abstraction view filtered out ≈50 percent of its input by hiding provenance of the other loop. Overall, the functions showed good performance, while the abstraction view derivation showed to be faster than the security view derivation. As a result, the derivation of a secure abstraction view as $sv_r^S(av^{\mathcal{A}}(WR))$ showed to be more efficient than the derivation of the same view as $av^{\mathcal{A}}(sv_r^S(WR))$, since, in the first case, sv_r^S was applied to a smaller data set.

To evaluate the performance of provenance queries, we stored our provenance data set with 100 workflow runs of size 10,000 statements (triples) in each into the RDFPROV system using the SchemaMapping-T and DataMapping-T algorithms presented in [48], which resulted in a database instance with one million triples. Similarly, we stored the secure abstraction view of each workflow run provenance in this data set as a new database instance with ≈400,000 triples. In Fig. 8, we report evaluation times for four queries, where each query was evaluated over the database with: 1) the original data set with one million triples, returning nonsecure results, 2) the secure abstraction view data set, returning secure results, and 3) the original data set with postprocessing to filter out (similar to deriveSecurityView and deriveAbstractionView) nonsecure and nonrelevant results based on security and abstraction view specifications, returning secure results. For a particular workflow run, the queries returned:

Q1. complete provenance of the workflow run,
Q2. provenance of task runs that executed an iterative workflow task (task involved in a loop),
Q3. provenance of a task run that executed a noniterative workflow task, and
Q4. provenance of a task run that executed a workflow task, whose provenance is not visible in the abstraction view.

The queries were expressed in SPARQL as follows: The first query returned all accessible provenance for a workflow run with identifier wr50 or, in other words, all accessible RDF triples (*subject*, *predicate*, *object*) from graph wr50:

```
SELECT ?sub ?pre ?obj
FROM <wr50>
WHERE {?sub ?pre ?obj .}
```

The second query returned provenance of task runs that executed an iterative task with identifier t5 for the same workflow run wr50 or, in other words, all accessible RDF triple from graph wr50 whose subjects were task runs (particular bindings of variable ?tr) that executed task t5:

```
SELECT ?tr ?pre ?obj
FROM <wr50>
WHERE {?tr :executes :t5 .
       ?tr ?pre ?obj .}
```

The other two queries were similar to Q2, except for that task identifiers were different to correspond to a noniterative workflow task and a task that did not belong to user's abstraction view.

As reported in Fig. 8, while both approaches to secure provenance querying showed good performance, the approach which relied on materialized secure abstraction views showed to be faster than the one which relied on dynamic view calculation. Depending on the disk space constraint, the change frequency of security and abstraction view specifications, and the number of such specifications, one approach or the other can be given a preference.

9 Conclusions and Future Work

In this work, we studied the problem of protecting scientific workflow provenance, including both data products and their dependencies. First, we formalized scientific workflow provenance model that builds the basis for querying and access control. Second, we proposed a security model for fine-grained access control for multilevel provenance and an algorithm for the derivation of a full security specification based on inheritance, overriding, and conflict resolution. Third, we formalized the notion of security views of provenance to serve as the security enforcement mechanism and proposed an algorithm for security view derivation. Fourth, we formalized the notions of abstraction views and secure abstraction views, and outlined algorithms for their computation. Finally, we developed the SECPROV prototype to validate the effectiveness of our approach and conducted a performance study. In the future, we will consider

1. conducting security case studies using scientific workflows with more complex data patterns [60],
2. developing a context-aware access control for workflow provenance where access to the output of a task depends on the access to the input of the task,
3. integrating our access control with an access control for data products (e.g., XML documents [61]) to deal with the granularity of data, and
4. studying usability of the system.

References

[1] L. Zhang, J. Zhang, and H. Cai, *Services Computing.* Springer-Verlag, 2007.

[2] Y. Simmhan, B. Plale, and D. Gannon, "Karma2: Provenance Management for Data-Driven Workflows," *Int'l J. Web Services Research,* vol. 5, no. 2, pp. 1-22, 2008.

[3] W.T. Tsai, X. Wei, Y. Chen, R.A. Paul, J.Y. Chung, and D. Zhang, "Data Provenance in SOA: Security, Reliability, and Integrity," *Service Oriented Computing and Applications,* vol. 1, no. 4, pp. 223-247, 2007.

[4] Y. Simmhan, B. Plale, and D. Gannon, "A Survey of Data Provenance in E-Science," *SIGMOD Record,* vol. 34, no. 3, pp. 31-36, 2005.

[5] R. Bose and J. Frew, "Lineage Retrieval for Scientific Data Processing: A Survey," *ACM Computer Surveys,* vol. 37, no. 1, pp. 1-28, 2005.

[6] S. Miles, P.T. Groth, M. Branco, and L. Moreau, "The Requirements of Using Provenance in E-Science Experiments," *J. Grid Computing,* vol. 5, no. 1, pp. 1-25, 2007.

[7] V. Atluri and J. Warner, "Security for Workflow Systems," *Handbook of Database Security Applications and Trends,* pp. 213-230, Springer, 2007.

[8] R.A. Botha and J.H.P. Eloff, "Separation of Duties for Access Control Enforcement in Workflow Environments," *End-to-End Security,* vol. 40, no. 3, pp. 666-682, 2001.

[9] "Workflow Security Considerations," White Paper wFMC-TC-1019, Workflow Management Coalition, Feb. 1998.

[10] J. Alhiyafi, C. Sabesan, S. Lu, and J.L. Ram, "RECOMBFLOW: A Scientific Workflow Environment for Intragenomic Gene Conversion Analysis in Bacterial Genomes, Including the Pathogen Streptococcus Pyogenes," *Int'l J. Bioinformatics Research and Applications,* vol. 5, no. 1, pp. 1-19, 2009.

[11] R. Martinho, D. Domingos, and A. Rito-Silvas, "Supporting Authentication Requirements in Workflows," *Proc. Eighth Int'l Conf. Enterprise Information Systems: Databases and Information Systems Integration,* pp. 181-188, 2006.

[12] J. Warner and V. Atluri, "Inter-Instance Authorization Constraints for Secure Workflow Management," *Proc. 11th ACM Symp. Access Control Models and Technologies,* pp. 190-199, 2006.

[13] W. Huang and V. Atluri, "Analysing the Safety of Workflow Authorization Models," *Proc. IFIP TC11 WG 11.3 12th Int'l Working Conf. Database Security XII,* pp. 43-57, 1999.

[14] S. Wu, A. Sheth, J. Miller, and Z. Luo, "Authorization and Access Control of Application Data in Workflow Systems," *J. Intelligent Information Systems,* vol. 18, no. 1, pp. 71-94, 2002.

[15] P. Hung and K. Karlapalem, "A Secure Workflow Model," *Proc. Australasian Information Security Workshop Conf. ACSW Frontiers,* 2003.

[16] S. Kandala and R. Sandhu, "Secure Role-Based Workflow Models," *Proc. 15th Ann. Working Conf. Database and Application Security,* pp. 45-58, 2001.

[17] V. Atluri, W. Huang, and E. Bertino, "A Semantic-Based Execution Model for Multilevel Secure Workflows," *J. Computer Security,* vol. 8, no. 1, pp. 3-41, 2000.

[18] E. Bertino, E. Ferrari, and V. Atluri, "The Specification and Enforcement of Authorization Constraints in Workflow Management Systems," *ACM Trans. Information and System Security,* vol. 2, no. 1, pp. 65-104, 1999.

[19] J. Wainer, P. Barthelmess, and A. Kumar, "W_RBAC—A Workflow Security Model Incorporating Controlled Overriding of Constraints," *Int'l J. Cooperative Information Systems,* vol. 12, no. 4, pp. 455-485, 2003.

[20] H. Davulcu, M. Kifer, L. Pokorny, C. Ramakrishnan, I. Ramakrishnan, and S. Dawson, "Modeling and Analysis of Interactions in Virtual Enterprises," *Proc. Ninth Int'l Workshop Research Issues on Data Eng.,* 1999.

[21] R. Sandhu, E. Coyne, H. Feinstein, and C. Youman, "Role-Based Access Control Models," *Computer,* vol. 29, no. 2, pp. 38-47, Feb. 1996.

[22] M. Kang, J. Park, and J. Froscher, "Access Control Mechanisms for Inter-Organizational Workflow," *Proc. Sixth ACM Symp. Access Control Models and Technologies,* pp. 66-74, 2001.

[23] E. Gudes, M. Olivier, and R. Riet, "Modelling, Specifying and Implementing Workflow Security in Cyberspace," *J. Computer Security,* vol. 7, no. 4, pp. 287-315, 1999.

[24] W. Huang and V. Atluri, "SecureFlow: A Secure Web-Enabled Workflow Management System," *Proc. Fourth ACM Workshop Role-Based Access Control,* pp. 83-94, 1999.

[25] D. Long, J. Baker, and F. Fung, "A Prototype Secure Workflow Server," *Proc. 15th Ann. Computer Security Applications Conf.,* pp. 129-133, 1999.

[26] M. zur Muehlen and M. Rosemann, "Workflow-Based Process Monitoring and Controlling—Technical and Organizational Issues," *Proc. 33rd Ann. Hawaii Int'l Conf. System Sciences,* 2000.

[27] H. Chivers and J. McDermid, "Refactoring Service-Based Systems: How to Avoid Trusting a Workflow Service," *Concurrency and Computation: Practice and Experience,* vol. 18, no. 10, pp. 1255-1275, 2006.

[28] B. Ludäscher, I. Altintas, C. Berkley, D. Higgins, E. Jaeger, M. Jones, E.A. Lee, J. Tao, and Y. Zhao, "Scientific Workflow Management and the Kepler System," *Concurrency and Computation: Practice and Experience,* vol. 18, no. 10, pp. 1039-1065, 2006.

[29] T.M. Oinn et al., "Taverna: Lessons in Creating a Workflow Environment for the Life Sciences," *Concurrency and Computation: Practice and Experience,* vol. 18, no. 10, pp. 1067-1100, 2006.

[30] J. Freire, C.T. Silva, S.P. Callahan, E. Santos, C.E. Scheidegger, and H.T. Vo, "Managing Rapidly-Evolving Scientific Workflows," *Proc. Int'l Provenance and Annotation Workshop (IPAW)*, 2006.

[31] E. Deelman, G. Singh, M.-H. Su, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, K. Vahi, G.B. Berriman, J. Good, A. Laity, J.C. Jacob, and D.S. Katz, "Pegasus: A Framework for Mapping Complex Scientific Workflows onto Distributed Systems," *Scientific Programming J.*, vol. 13, no. 3, pp. 219-237, 2005.

[32] Y. Zhao, M. Hategan, B. Clifford, I. Foster, G. von Laszewski, I. Raicu, T. Stef-Praun, and M. Wilde, "Swift: Fast, Reliable, Loosely Coupled Parallel Computation," *Proc. Int'l Workshop Scientific Workflows (SWF) in conjunction with Int'l Conf. Services Computing (SCC)*, 2007.

[33] "The Provenance Challenge Series," http://twiki.ipaw.info/bin/view/Challenge, 2010.

[34] V. Tan, P. Groth, S. Miles, S. Jiang, S. Munroe, S. Tsasakou, and L. Moreau, "Security Issues in a SOA-Based Provenance System," *Proc. Third Int'l Provenance and Annotation Workshop*, 2006.

[35] U. Braun and A. Shinna, "A Security Model for Provenance," Technical Report TR-04-06, Harvard Univ., Jan. 2006.

[36] S.B. Davidson and J. Freire, "Provenance and Scientific Workflows: Challenges and Opportunities," *Proc. SIGMOD*, pp. 1345-1350, 2008.

[37] W. van der Aalst, "Inheritance of Interorganizational Workflows: How to Agree or Disagree without Loosing Control?" *Information Technology and Management J.*, vol. 2, no. 3, pp. 195-231, 2002.

[38] C. Lin, S. Lu, X. Fei, D. Pai, and J. Hua, "A Task Abstraction and Mapping Approach to the Shimming Problem in Scientific Workflows," *Proc. Int'l Conf. Services Computing (SCC)*, pp. 284-291, 2009.

[39] A. Michlmayr, F. Rosenberg, P. Leitner, and S. Dustdar, "Service Provenance in QoS-Aware Web Service Runtimes," *Proc. Int'l Conf. Web Services (ICWS)*, pp. 115-122, 2009.

[40] K. Xu, Y. Wang, and C. Wu, "Service Provenance Based Abstraction of Grid Application Knowledge," *Proc. Second Int'l Conf. Semantics, Knowledge, and Grid*, pp. 50-53, 2006.

[41] P.T. Groth, S. Miles, and L. Moreau, "A Model of Process Documentation to Determine Provenance in Mash-Ups," *ACM Trans. Internet Technology*, vol. 9, no. 1, 2009.

[42] Provenance Aware Service Oriented Architecture (PASOA) Project, http://www.pasoa.org, 2010.

[43] W.T. Tsai, X. Wei, D. Zhang, R. Paul, Y. Chen, and J.Y. Chung, "A New SOA Data-Provenance Framework," *Proc. Eighth Int'l Symp. Autonomous Decentralized Systems*, pp. 105-112, 2007.

[44] S.M.S. Cruz, P.M. Barros, P.M. Bisch, M.L.M. Campos, and M. Mattoso, "Provenance Services for Distributed Workflows," *Proc. Int'l Symp. Cluster Computing and the Grid (CCGRID)*, pp. 526-533, 2008.

[45] C. Lin, S. Lu, X. Fei, A. Chebotko, D. Pai, Z. Lai, F. Fotouhi, and J. Hua, "A Reference Architecture for Scientific Workflow Management Systems and the VIEW SOA Solution," *IEEE Trans. Services Computing*, vol. 2, no. 1, pp. 79-92, Jan.-Mar. 2009.

[46] P. Groth, S. Miles, W. Fang, S.C. Wong, K.-P. Zauner, and L. Moreau, "Recording and Using Provenance in a Protein Compressibility Experiment," *Proc. Int'l Symp. High Performance Distributed Computing (HPDC)*, 2005.

[47] P. Groth, S. Jiang, S. Miles, S. Munroe, V. Tan, S. Tsasakou, and L. Moreau, "An Architecture for Provenance Systems Executive Summary," technical report, Univ. of Southampton, Feb. 2006.

[48] A. Chebotko, S. Lu, X. Fei, and F. Fotouhi, "RDFProv: A Relational RDF Store for Querying and Managing Scientific Workflow Provenance," to be published in *Data and Knowledge Eng.*, vol. 69, no. 8, pp. 836-865, 2010.

[49] Open Provenance Model, http://openprovenance.org, 2010.

[50] O. Biton, S. Cohen-Boulakia, S. Davidson, and C. Hara, "Querying and Managing Provenance through User Views in Scientific Workflows," *Proc. 24th IEEE Int'l Conf. Data Eng. (ICDE)*, pp. 1072-1081, 2008.

[51] O. Biton, S.B. Davidson, S. Khanna, and S. Roy, "Optimizing User Views for Workflows," *Proc. 12th Int'l Conf. Database Theory (ICDT)*, pp. 310-323, 2009.

[52] M. Kifer, A. Bernstein, and P.M. Lewis, *Database Systems: An Application Oriented Approach.* Addison-Wesley, 2006.

[53] R. Elmasri and S.B. Navathe, *Fundamentals of Database Systems.* Addison-Wesley, 2004.

[54] A. Chebotko, S. Chang, S. Lu, F. Fotouhi, and P. Yang, "Scientific Workflow Provenance Querying with Security Views," *Proc. Int'l Conf. Web-Age Information Management*, pp. 349-356, 2008.

[55] R.S. Sandhu and Q. Munawer, "The ARBAC99 Model for Administration of Roles," *Proc. 15th Ann. Computer Security Applications Conference (ACSAC)*, pp. 229-238, 1999.

[56] S. Oh and R.S. Sandhu, "A Model for Role Administration Using Organization Structure," *Proc. Seventh ACM Symp. Access Control Models and Technologies*, pp. 155-162, 2002.

[57] J. Crampton and G. Loizou, "Administrative Scope: A Foundation for Role-Based Administrative Models," *ACM Trans. Information and System Security*, vol. 6, no. 2, pp. 201-231, 2003.

[58] SecProv, http://www.cs.panam.edu/~artem/SecProv.zip, 2010.

[59] J. Wei, L. Singaravelu, and C. Pu, "Guarding Sensitive Information Streams through the Jungle of Composite Web Services," *Proc. Int'l Conf. Web Services (ICWS)*, pp. 455-462, 2007.

[60] N. Russell, A. ter Hofstede, D. Edmond, and W. van der Aalst, "Workflow Data Patterns," Technical Report FIT-TR-2004-01, Queensland Univ. of Technology, 2004.

[61] E. Damiani, S. De Capitani di Vimercati, S. Paraboschi, and P. Samarati, "A Fine-Grained Access Control System for XML Documents," *ACM Trans. Information and System Security*, vol. 5, no. 2, pp. 169-202, 2002.

Artem Chebotko received the PhD degree in computer science from Wayne State University in 2008. He is currently an assistant professor in the Department of Computer Science, University of Texas—Pan American. His research interests include scientific workflows, services computing, and semantic web data management. He has published more than 30 refereed research papers and currently serves as a program committee member of several international conferences and workshops on scientific workflows, services computing, and semantic web. He is a member of the IEEE.

Shiyong Lu received the PhD degree from the State University of New York at Stony Brook in 2002. He is currently an associate professor in the Department of Computer Science, Wayne State University, and the director of the Scientific Workflow Research Laboratory. His research interests include scientific workflows and databases. He has more than 90 refereed publications in the above areas. He is the founder and currently a cochair of the IEEE International Workshop on Scientific Workflows and an editorial board member for the *International Journal of Healthcare Information Systems and Informatics* and the *International Journal of Semantic Web and Information Systems*. He serves as a program committee member for several top-tier IEEE conferences including SCC and ICWS. He is a senior member of the IEEE.

Seunghan Chang received the PhD degree in computer science from Wayne State University in 2008. His research interests include workflow and XML data security. He is currently an active officer of the Republic of Korea Armed Forces.

Farshad Fotouhi received the PhD degree in computer science from Michigan State University in 1988. In August 1988, he joined the faculty of computer science at Wayne State University, where he is currently a professor and the chair of the department. His research interests include databases, query optimization, and multimedia systems. He has published more than 100 papers in refereed journals and conference proceedings and served as a program committee member of various database-related conferences. He is on the editorial boards of *IEEE Multimedia* and the *International Journal on Semantic Web and Information Systems*. He is a member of the IEEE.

Ping Yang received the BS degree from Sun Yat-Sen University, the ME degree from the Chinese Academy of Sciences, and the PhD degree from Stony Brook University. She is an assistant professor in the Computer Science Department at the State University of New York at Binghamton. Her research areas are information and systems security, privacy, and formal methods. She received the Most Practical Paper Award from the Seventh International Symposium on Practical Aspects of Declarative Languages in 2005. She is a member of the IEEE.