# On Provenance and Privacy

Susan B. Davidson, Sanjeev Khanna, Sudeepa Roy, Julia Stoyanovich, Val Tannen
University of Pennsylvania, Philadelphia, USA
{susan, sanjeev, sudeepa, jstoy, val}@cis.upenn.edu

Yi Chen
Arizona State University, Tempe, USA
yi@asu.edu

## ABSTRACT

Provenance in scientific workflows is a double-edged sword. On the one hand, recording information about the module executions used to produce a data item, as well as the parameter settings and intermediate data items passed between module executions, enables transparency and reproducibility of results. On the other hand, a scientific workflow often contains private or confidential data and uses proprietary modules. Hence, providing exact answers to provenance queries over all executions of the workflow may reveal private information. In this paper we discuss privacy concerns in scientific workflows – data, module, and structural privacy - and frame several natural questions: (i) Can we formally analyze data, module, and structural privacy, giving provable privacy guarantees for an unlimited/bounded number of provenance queries? (ii) How can we answer search and structural queries over repositories of workflow specifications and their executions, providing as much information as possible to the user while still guaranteeing privacy? We then highlight some recent work in this area and point to several directions for future work.

## Categories and Subject Descriptors

H.2.0 [**Database Management**]: General—*Security, integrity and protection*; H.3.3 [**Information Systems**]: Information Storage and Retrieval—*Information Search and Retrieval*

## General Terms

Design, Theory

## Keywords

Provenance, Privacy, Scientific Workflows

## 1. INTRODUCTION

Provenance in scientific workflows is of increasing interest, as evidenced by several recent workshops, tutorials, and surveys on the topic. A number of tools for capturing provenance have been developed in workflow systems such as myGrid/Taverna [33], Kepler [9], and VisTrails [20], and a standard for provenance representation called the Open Provenance Model (OPM) [30] has been designed. By maintaining information about the sequence of module executions used to produce a data item, as well as the parameter settings and intermediate data items passed between module executions, the validity and reliability of data can be better understood and results be made reproducible.

Currently, users of a workflow system can see a repository of workflow *specifications* (as with myExperiment [32] and some of the systems cited above), and possibly also the initial inputs and final outputs of workflow executions. However, we envision a future in which extensive workflow *execution* records will be added to these repositories of workflow specifications; we will call such repositories *Workflow Provenance repositories* (`WP` repositories). `WP` repositories could be used in several ways: Scientists who wish to perform new analyses may use *keyword search* to find specifications of interest to reuse or modify. They may also search executions associated with a specification to understand the meaning of the workflow, or to correct/debug an erroneous specification. Finding erroneous or suspect data, a user may then wish to ask *structural queries* over provenance information to determine what downstream data might have been affected, or to understand how the process failed that led to creating the data.

However, workflow authors or owners may wish to keep some information confidential in a `WP` repository. For example, intermediate *data* within an execution may contain sensitive information, such as a social security number, a medical record, or financial information about an individual. Although users with the appropriate access level may be allowed to see such confidential data, making it available to all users, even for scientific purposes, is an unacceptable breach of privacy. Beyond data privacy, a *module* itself may be proprietary, and hiding its description may not be enough: users without the appropriate access level should not be able to infer its behavior if they are allowed to see the inputs and outputs of the module. Finally, details of how certain modules in the workflow are connected may be proprietary, and so showing how data is passed between modules may reveal too much of the *structure* of the workflow. *There is thus an inherent tradeoff between the utility of the information provided in response to a search/query and the privacy guarantees that authors/owners desire.*

We illustrate these three types of privacy using an example from life sciences, a domain in which privacy concerns are particularly acute. Consider a personalized disease susceptibility workflow in Figure 1 (see [37] for details). *Data privacy* requires that the genetic disorders a patient is susceptible to (the output of $M1$) should not be revealed to users without the required access privilege. *Module privacy* with respect to $M1$ requires that the functionality of the module – that is, the mappings between inputs and outputs – is not revealed to users without the required access privilege. Assuming that $M1$ implements a function $f_1$, no adversarial user should be able to correctly deduce the output $f_1$(`SNP`, `ethnicity`) with high probability for any `SNP`[1] and `ethnicity` input pair. From a patient's perspective, this is important because they do not want someone who happens to have access to their `SNP` and `ethnicity` information to be able to determine what `disorders` they are susceptible to. From the module owner's perspective, they do not want the module to be simulated by someone who observes some of the input-output relationships. Finally, *structural privacy* in this example might mean that users without the required access privilege should not know whether or not `lifestyle` was used to calculate the `disorders` output by $M1$.

Note the difference between module and data privacy: For module privacy, we may reveal `disorders` as long as we do not know the values of both `SNP` and `ethnicity` such that `disorders` $=$ $f_1$(`SNP`, `ethnicity`). However, for data privacy `disorders` can never be revealed.

As recently noted in [36], "You are better off designing in security and privacy ... from the start, rather than trying to add them later."[2] Accordingly, we believe that privacy guarantees should be *integrated* with the design of `WP` repositories for scientific workflow systems.

**Organization.** We give a model of workflow specifications and their executions in Section 2. In Section 3 we enumerate privacy concerns, discuss related work, and highlight our initial results in module privacy. We turn to keyword search and queries on `WP` repositories in Section 4, and discuss how privacy concerns affects search and query results. Section 5 concludes and points to future work.

## 2. MODEL

A workflow *specification* is typically represented by a directed acyclic graph, with nodes denoting modules and edges indicating dataflow between modules. Modules can be labeled with names, keywords, and descriptions; the description may include the name and type of input/output data. Workflow specifications may be *hierarchical*, in the sense that a module may be *composite* and itself contain a workflow. Composite modules are frequently used to simplify workflow design and allow component reuse. Workflows that do not have composite modules are referred as *simple workflows*.

For example, the workflow in Figure 1 estimates disease susceptibility based on genome-wide SNP array data. The top-most level of the workflow is given by the dotted box labeled $W1$. The input to $W1$ is a set of SNPs, ethnicity infor-

[1]Single Nucleotide Polymorphism (SNP), a DNA sequence variation occurring when a single nucleotide in the genome differs between members of a species or paired chromosomes in an individual.

[2]While the context for this statement was the use of full body scanning in airports (where the privacy issues are obvious), it is equally valid in `WP` repositories!

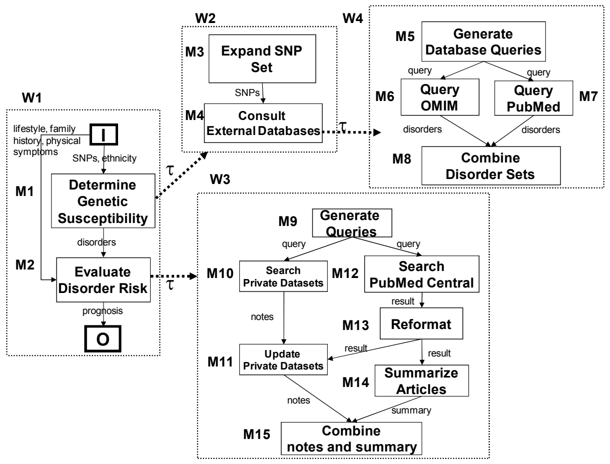

Figure 1: Disease Susceptibility Workflow Specification

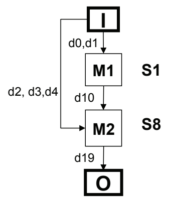

Figure 2: View of Provenance Graph

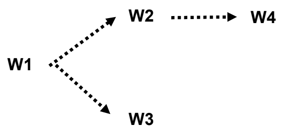

Figure 3: Expansion Hierarchy

mation, lifestyle, family history, and physical symptoms, and is indicated by a special node labeled $I$. The output for the workflow is a prognosis, indicated by a special node labeled $O$. The first module in $W1$ is named $M1$ with description "Determine Genetic Susceptibility". $M1$ determines a set of disorders the patient is genetically susceptible to based on the input SNPs and ethnicity information. The second module, named $M2$ with description "Evaluate Disorder Risk", refines the set of disorders for which the patient is at risk, based on lifestyle, family history, and physical symptoms.

Figure 1 also contains $\tau$-labeled edges that give the definitions of composite modules, which we call *expansions*. For example, $M1$ is defined by the workflow $W2$, $M2$ by the workflow $W3$, and $M4$ by the workflow $W4$. Hence $W2$ and $W3$ are *subworkflows* of $W1$, and $W4$ is a subworkflow of $W2$. The $\tau$ expansions (subworkflow relationships) naturally yield an *expansion hierarchy* as shown in Figure 3.

Prefixes of the expansion hierarchy can be used to define views of a workflow specification. A *prefix* of an expansion hierarchy $H$ is a tree obtained from $H$ by deleting some of its subtrees, and is denoted by the set of nodes contained in the prefix. For example, $\{W1, W2\}$ is a valid prefix for the expansion hierarchy in Figure 3 but $\{W1, W4\}$ is not. Given a prefix, the *view* that it defines is given by expanding the root workflow so that composite modules in the prefix are replaced by their expansions. For example, the prefix $\{W1, W2, W4\}$ determines a view of the specification in Figure 1 that is the simple workflow obtained from $W1$ by replacing $M1$ with $W2$ and $M4$ with $W4$ (see Figure 5). The prefix $\{W1, W2, W3, W4\}$ is the full expansion, and yields

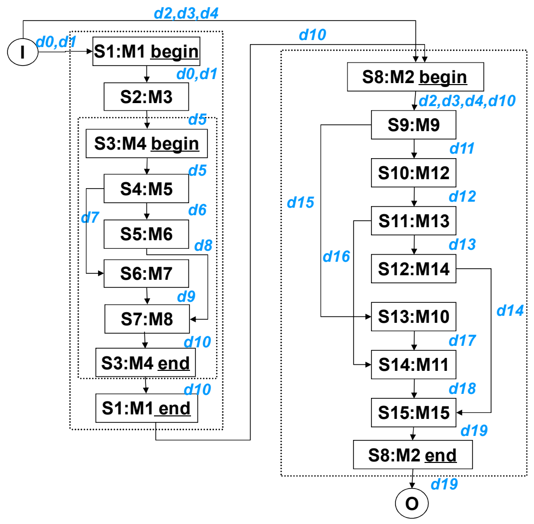

Figure 4: Disease Susceptibility Workflow Execution

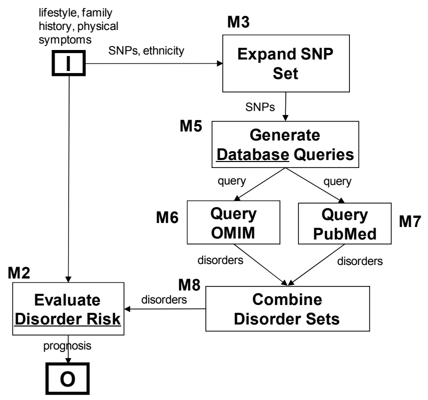

Figure 5: Result of "Database, Disorder Risk" using prefix $\{W1, W2, W4\}$

a workflow with module names $I, O, M3$, and $M5 - M15$, and whose edges include one from $M3$ to $M5$ and another from $M8$ to $M9$. We will shortly discuss the benefit of views.

A workflow specification describes the possible run-time *executions*. Executions are modeled similarly to simple workflow graphs, but additionally associate a unique process id with a module execution, and data with edges. When execution reaches a composite module, it continues in the corresponding subworkflow and eventually returns (like a procedure call). For example, an execution of the workflow specification in Figure 1 is shown in Figure 4. In this example, for clarity we show the process id appended to the name of the module being executed, e.g., S1:M1. Each composite module execution is represented by two nodes, the first standing for its activation and the second for its completion, e.g., S1:M1-begin and S1:M1-end.

In an execution, data flows over the edges. We assume that each data item (an object with a value) is the output of exactly one module execution and has a unique id. We therefore annotate each edge $M \rightarrow N$ in the execution with the set of data items that flow as the output of $M$ to the input of $N$. For example, in Figure 4 the set $\{d0, d1\}$ flows from I to S1:M1.

The *provenance* of a data item $d$ in an execution $E$ is therefore the subgraph induced by the set of paths from the start node to the end node of $E$ that produced $d$ as output. In the sequel, we blur the distinction between the provenance of data items and the executions that produce them.

As introduced in [7], we can use views to simplify what is seen of an execution. Using the view defined by prefix $\{W1\}$, the execution of Figure 4 would be simplified to that in Figure 2. Views can also be used to define *access control* to address privacy concerns. Specifically, we can define a user's access privilege as the finest grained view that s/he can access, called an *access view*. We will return to the issue of access views, keyword search and structural querying in Section 4 after describing privacy concerns.

## 3. PRIVACY

Privacy concerns are tied to the workflow components: data, modules, and the structure of a workflow. In *data privacy*, private intermediate data, that is, data flowing between modules in a workflow execution, must not be revealed to a user. In *module privacy*, we wish to ensure that the functionality of the module is hidden from the users in a strong sense, namely, for any input to the module, a user should not be able to guess with any degree of certainty the output of the module. In *structural privacy*, we wish to ensure that details of how a particular data product has been generated in an execution of the workflow are kept private from a user, and portions of the provenance graph must not be revealed.

Broadly speaking, the fundamental question to be addressed is: *How do we provide provable guarantees on the privacy of components in a workflow while maximizing utility with respect to provenance queries?* In doing so, we must understand 1) how to measure privacy; 2) what information can be hidden; 3) how to measure utility; and 4) how to efficiently find solutions that *simultaneously* provide provably good guarantees on privacy and utility. Note that all privacy guarantees must hold over repeated executions of a workflow with varied inputs.

After briefly discussing issues of data privacy in our setting, we describe our initial results for module privacy (see [13, 14, 15] for details) as well as initial ideas on structural privacy. We close by describing related work.

### 3.1 Data Privacy

In scientific workflows, provenance is used for ensuring the repeatability of experiments and verifiability of results. For example, if the output of a workflow execution is believed to be incorrect, the user will need to trace through the intermediate data and parameter settings used by modules to determine the source of the error. Unlike statistical databases, aggregate queries are not used, and the assumption is that the values of data shown to the user are *exact* rather than approximate. Existing techniques for preserving data privacy in statistical and relational databases (see last subsection) therefore do not directly extend to this setting. Access control techniques to hide a data value must also be used in conjunction with module privacy techniques to ensure that hidden data is not "leaked" through visible data and modules whose behavior can be simulated (*public* modules). For example, hiding a data value that is the output of a public reformatting module does not ensure its privacy.

We therefore start by focusing on module privacy.

## 3.2 Module Privacy

To simplify the discussion, we assume that all modules are atomic (i.e., not composite), yielding a workflow with a single node in the expansion hierarchy (the workflow itself). We return to the question of hierarchy later. Overloading terminology slightly, we will also use $I$ to denote the set of input data to a module, and $O$ to denote the set of output data for a module.

In a typical scenario, some of the modules in a workflow are private while others are *public*, i.e., their function and behavior is assumed to be known. Our initial work [13, 14] addresses a restricted case in which *all* modules are private and the user has no apriori knowledge of their functionality.[3] Although requiring all modules to be private is a special case, the privacy issues are already complex and yield significant insight into handling the general case in which some modules may be public.

It is easy to see that if information about all intermediate data is repeatedly given for multiple executions of a workflow on different initial inputs, then partial or complete functionality of modules may be revealed. Our mechanism to achieve privacy of modules is therefore to *hide a carefully chosen subset of intermediate data*, thereby limiting the amount of provenance data shown to the user. We assume that users can see initial input and final output data as well as all connections or edges between modules in the workflow; only the *values* of selected intermediate data are hidden, and are hidden in *all* executions of the workflow. Hiding is accomplished by access control.

**Standalone Module Privacy.** Informally, we say that a module $M$ in isolation (a *standalone module*) is $\Gamma$-*private* for some parameter $\Gamma \geq 1$ w.r.t. the information visible to the user if, given any input $I$ to the module, the user cannot guess the correct output $O = f(I)$ with probability greater than $\frac{1}{\Gamma}$. $\Gamma$-privacy is similar to the concept of $\ell$-diversity [28] but extended to modules in a workflow setting.

Note that we can equivalently think of $f$ as a table $R$ over a set of attributes $A = I \cup O$ that satisfies the functional dependency $I \rightarrow O$, and we use this in our formalization of $\Gamma$-privacy.

$\Gamma$-privacy is formalized using a notion of consistent functions: Two functions are said to be *consistent* w.r.t. a set $V$ of visible attributes (data items) if their tables are the same when projected over $V$. Given a set of visible attributes for $M$, for each input $I$ the set of consistent functions must map $I$ to at least $\Gamma$ different output values.

As a simple example, consider a module $M$ whose functionality is shown in the table $R_1^1$ in Figure 6a. The attributes under $I$ ($a_1$, $a_2$) indicate the data input to $M$ and the attributes under $O$ ($a_3$, $a_4$, $a_5$) represent the data output from $M$; hence, each row in the table represents an execution of $M$. If $a_2$ and $a_4$ are hidden in the displayed provenance ($V = \{a_1, a_3, a_5\}$), then $\Gamma$-privacy is achieved for $\Gamma$=4, since there are four possible outputs for each input, shown by the set of consistent functions represented in Figure 6. For example, input (0,0) can be mapped to outputs (0,0,1), (0,1,1), (1,0,0) or (1,1,0). The same holds for inputs (0,1), (1,0) and (1,1). We therefore call the set $\{a_2, a_4\}$ a *safe subset* for $\Gamma = 4$.

[3]The *functionality* of a module with an underlying function $f$ refers to the input-output behavior of the module, i.e., the $(I, O = f(I))$ pairs for all possible inputs $I$ to the module, and *not* the underlying algorithm to compute the function $f$.

| $I$ | | $O$ | | |
|---|---|---|---|---|
| $a_1$ | $a_2$ | $a_3$ | $a_4$ | $a_5$ |
| 0 | 0 | 0 | 0 | 1 |
| 0 | 1 | 1 | 0 | 0 |
| 1 | 0 | 1 | 0 | 0 |
| 1 | 1 | 1 | 0 | 1 |

(a) $R_1^1$

| $I$ | | $O$ | | |
|---|---|---|---|---|
| $a_1$ | $a_2$ | $a_3$ | $a_4$ | $a_5$ |
| 0 | 0 | 0 | 1 | 1 |
| 0 | 1 | 1 | 1 | 0 |
| 1 | 0 | 1 | 0 | 0 |
| 1 | 1 | 1 | 0 | 1 |

(b) $R_1^2$

| $I$ | | $O$ | | |
|---|---|---|---|---|
| $a_1$ | $a_2$ | $a_3$ | $a_4$ | $a_5$ |
| 0 | 0 | 1 | 0 | 0 |
| 0 | 1 | 0 | 0 | 1 |
| 1 | 0 | 1 | 0 | 0 |
| 1 | 1 | 1 | 0 | 1 |

(c) $R_1^3$

| $I$ | | $O$ | | |
|---|---|---|---|---|
| $a_1$ | $a_2$ | $a_3$ | $a_4$ | $a_5$ |
| 0 | 0 | 1 | 1 | 0 |
| 0 | 1 | 0 | 1 | 1 |
| 1 | 0 | 1 | 0 | 0 |
| 1 | 1 | 1 | 0 | 1 |

(d) $R_1^4$

Figure 6: Consistent functions w.r.t. $V = \{a_1, a_3, a_5\}$

Other combinations of inputs/outputs may also be safe subsets for $\Gamma$=4 for $M$, for example, $\{a_4 , a_5\}$ (or any two of the output attributes).[4] In general, the *standalone privacy requirement* of each module in a workflow is specified as a *requirements list*, i.e., a list of safe subsets for some specified $\Gamma$. Since some of the input/output data items may be more valuable than others to the user, we allow the user to associate a *cost* with each data item, leading to the following natural optimization problem: Given a requirements list for $M$ and a cost $\mathsf{c}(a)$ associated with each input/output data item $a$, find a safe subset $V$ s.t. the cost of the hidden attributes, $\mathsf{c}(\overline{V}) = \sum_{a \in \overline{V}} \mathsf{c}(a)$, is minimized.

Although in the worse case finding a minimal-cost safe attribute set for a module may take time that is exponential in the number $k$ of attributes [13], $k$ is typically not large (often less than 10, see [32]), so the computation is still feasible. Knowledge, on the part of the module designers, about the module's behavior and safe attribute sets may also be exploited to speed up the computation. Furthermore, a given module is often used in many workflows. For example, sequence alignment modules, like BLAST or FASTA, are used in many different biological workflows. Since the safe subsets for individual modules can be used as building blocks for attaining privacy for the full workflow, the effort invested in deriving safe subsets for a module may therefore be amortized over all uses.

**In-network Module Privacy.** The *in-network* $\Gamma$-privacy of a module is defined similarly to standalone $\Gamma$-privacy: For *any* input to *any* module in the workflow, the user should not be able to guess the correct output of the module with probability greater than $\frac{1}{\Gamma}$. Thus, the notion of in-network privacy of a module is inherently linked to the topology of the network representing the workflow as well as the functionality of the modules – and the problem becomes considerably more complex. For example, *data sharing* may occur, in which output data from one module is input to more than one downstream module.

In [13, 14] we show that, although a sequence of consistent functions for the network is always consistent for the individual standalone modules with respect to the same set of hidden data, the converse is not true. In particular, a

[4]Note that $\{a_1, a_2\}$ is only a safe subset for $\Gamma$=3, since each input can only be mapped to (0,0,1), (1,0,0) or (1,0,1).

sequence of consistent functions for a set of standalone modules with respect to a set of hidden data does not guarantee that the same sequence is consistent in a network setting, when the same set of data is hidden. In general, it appears that in-network privacy of modules requires us to consider the input-output behavior of *all* modules given *all* possible initial inputs, which may be computationally infeasible.

However, we ultimately show in [13, 14] that in-network privacy can indeed be captured in terms of the local standalone privacy requirements of individual modules. The resulting *secure-view* optimization problem can be stated as follows: *Given a privacy parameter $\Gamma \geq 1$, what is the minimum cost of intermediate data that needs to be hidden to guarantee that all modules in the workflow have in-network $\Gamma$-privacy?* Our work in [14] shows that the problem is NP-hard even in a very restricted setting, motivating poly-time *approximation algorithms* for the secure-view problem.

**General Workflows.** We then show in [13] that ensuring standalone privacy of private modules in workflows involving both public and private modules (*general* workflows) no longer guarantees in-network privacy. However, by making some of the public modules private (called *privatization*) we can attain workflow privacy of all private modules in the workflow. Since privatization has a cost, the optimization problem becomes much harder: even without data sharing the problem is $\Omega(\log n)$-hard to approximate.

**Module Privacy in Hierarchical Workflows.** We now revisit module privacy in hierarchical workflows. Here we assume that any module, whether composite or atomic, could be private. If a private module is composite, then expanding it might reveal something of its functionality. Expansions of private composite modules are therefore disallowed, yielding a (maximal) access view consisting of all prefixes of the expansion hierarchy terminating with either an atomic or a private composite module. As described earlier, input/output data would have to be hidden for the private composite module to achieve the desired level of privacy.

## 3.3 Structural Privacy

We now describe some initial ideas on structural privacy. Recall that the provenance of a data item is defined to be the subgraph in an execution consisting of all execution paths originating at the start node and terminating at the module that produces the data item. The goal of structural privacy is to keep private the information that some module $M$ contributes to the generation of a data item $d$, output by another module $M'$. For instance, in the execution of the workflow $W3$, we may wish to hide the fact that the reformatted data from PubMed Central (module $M13$) contributes to updating of the private DB, and hence to the output of module $M11$.

One possible solution is to delete edges and vertices so as to eliminate all $M \rightsquigarrow M'$ paths. However, by doing so, we may hide not only the fact that $M \rightsquigarrow M'$, but additional provenance information that does not need be hidden, say, the fact that $M \rightsquigarrow M''$ for some $M'' \neq M$. Hiding the provenance information $M \rightsquigarrow M''$ represents lost *utility* for the public pair $(M, M'')$. Broadly speaking, our goal is to study the question of how to hide private structural information while maximizing utility, or minimizing lost utility.

We can formalize the structural privacy problem as follows. Given an execution $G(V, E)$, along with a collection $P \subseteq V \times V$ of ordered pairs of vertices, create a new execution $G'(V', E')$ that hides reachability for all pairs in $P$ but preserves the reachability information for a maximum number of node pairs in $Q \stackrel{\text{def}}{=} (V \times V) \setminus P$. Here, $P$ and $Q$ respectively denote the sets of private and public module pairs. In general, the transformation from $G$ to $G'$ may create a $u \rightsquigarrow v$ path in $G'$ for some public pair $(u, v) \in Q$ for which $u \not\rightsquigarrow v$ in $G$, or may show that $u \not\rightsquigarrow v$ in $G'$ when $u \rightsquigarrow v$ in $G$. In either case, we lose utility for the pair $(u, v)$ in the transformation. We may view the goal of preserving reachability for pairs in $Q$ as a *completeness* property, and the goal of preserving non-reachability for pairs in $Q$ as a *soundness* property. Then the utility lost by a solution $G'$ can be measured in terms of the number of pairs in $Q$ for which the completeness or soundness properties are violated.

There are two natural approaches to create a structural-privacy preserving graph $G'$. The first approach, mentioned earlier, is to simply *delete* a subset of edges and vertices in $G'$ so as to ensure that no pair in $P$ satisfies reachability in $G'$. Clearly, this approach is guaranteed to generate *sound* solutions since $G'$ is always a subgraph (projection) of $G$. A second approach is to *cluster* modules so that for any pair $(u, v)$ in $P$, $u$ and $v$ belong to a new composite module whose expansion is disallowed in any access view, and thus reachability between $u$ and $v$ is no longer externally visible. In contrast to deletion, clustering tends to preserve completeness at the expense of violating soundness. A composite module that also preserves soundness (and thus has high utility) is called a *sound module*.

As a concrete example, consider an execution of the workflow $W3$ in Figure 1, and let $P = \{(M13, M11)\}$. If we use deletion, then the only possible solution is to delete the edge $(M13, M11)$ from $W3$. This solution, however, also deletes the $M12 \rightsquigarrow M11$ path, thus losing utility for the pair $(M12, M11)$. In contrast, the clustering solution merges modules $M13$ and $M11$ into a single composite module, say, $M16$, whereby all edges in $G$ coming to $M13$ or $M11$ now enter the module $M'$, while all edges leaving $M13$ or $M11$ now leave from the module $M16$. This solution preserves reachability for all non-private pairs, but violates soundness by creating some new paths. For instance, we now have a $M10 \rightsquigarrow M14$ path that did not exist in the original workflow, thus losing the utility for the pair $(M10, M14)$.

We studied the problem of creating a set of complete and sound composite modules in a workflow in a related but different setting [8, 38]. For instance, in [8], we consider the problem of creating composite modules that preserve pairwise reachability for a given subset of *relevant* modules. Here the completeness and soundness are hard constraints, and the utility is measured in terms of the reduction achieved in the size of the workflow. This is different from our setting, where the privacy requirements may necessitate some violation in completeness and soundness, and the focus is on containing the extent of these violations.

**Structural Privacy in Hierarchical Workflows.** Clustering modules within the same subworkflow will introduce new nodes in the expansion hierarchy for a workflow. For example, clustering $M11$ and $M13$ into a composite module $M16$, whose expansion is a new subworkflow $W5$, would place $W5$ as a child of $W3$ in the expansion hierarchy in Figure 3. While the expansion of $M16$ would be disabled, the fact that it contains $M11$ and $M13$ would be known, perhaps by allowing $M16$ to "inherit" the keywords/title/description of the contained modules. Clustering should respect the hierarchy: nodes in different subworkflows (such as $M2$ and $M10$, or $M3$ and $M6$) should not be placed in the same cluster. When modules from two different sub-workflows are chosen to be clustered, then the parent of the least common ancestor of the two sub-workflows would be disabled for expansion (involving a significant loss of utility).

## 3.4 Related Work

*Access control* is heavily used in relational databases, see, for example, a discussion in [34] (Chapter 21). More recently, data privacy has been considered in hierarchical data models such as XML [6, 12, 19]. There, an access control policy is typically specified, assigning access levels to users, or groups of users, with respect to XML elements or attribute values [12]. Such a policy may then be used to compute a *security view* over an XML document [19]. We plan to evaluate the applicability of these techniques in our setting.

*Workflow privacy* has been considered in [11, 21, 22]. In [11], the authors discuss a framework to output a *partial* view of a workflow that conforms to a given set of access permissions on the connections between modules and data on input/output ports. The problem of ensuring the *lawful use* of data according to specified privacy policies has been considered in [21, 22]. The focus of the work is a policy language for specifying relationships among data and module sets, and their properties relevant to privacy. Although all these papers address workflow privacy, the privacy notions are somewhat informal and no guarantees on the quality of the solution are provided in terms of privacy and utility.

*Secure provenance* for workflows has been studied in [10, 23, 27]. The goal is to ensure that provenance information has not been forged or corrupted, and a variety of cryptographic and trusted computing techniques are proposed. In contrast, we assume that provenance information has not been corrupted, and focus on ensuring module privacy.

In [29], the authors study information disclosure in data exchange, where, given a set of public views, the goal is to decide if they reveal any information about a private view. This does not directly apply to our problem, where the private elements are the $(\mathbf{x}, m(\mathbf{x}))$ relations. For example, if all $\mathbf{x}$ values are shown without showing any of the $m(\mathbf{x})$ values for a module $m$, then information is revealed in their setting but not in our setting.[5]

*Privacy-preserving data mining* has received considerable attention (see surveys [1, 41]). The goal is to hide individual data attributes while retaining the suitability of data for mining patterns. For example, the technique of *anonymizing data* makes each record indistinguishable from a large enough set of other records in certain identifying attributes [2, 28, 39]. Privacy preserving approaches were studied for *social networks* [4, 35], *auditing queries* [31], and in other contexts. Our notion of *standalone* module privacy is close to that of $\ell$-diversity [28], in which the values of *non-sensitive attributes* are generalized so that, for every such generalization, there are at least $\ell$ different values of *sensitive attributes*. We extend this work in two ways: First, we place modules (relations) in a network of modules, which significantly complicates the problem; Second, we analyze the complexity of attaining standalone as well as workflow privacy of modules.

[5]In contrast, it can be shown that showing all $m(\mathbf{x})$ values while hiding the $\mathbf{x}$'s, may reveal information in our setting.

Another widely used technique is that of *data perturbation*, where some noise, usually random, is added to the the output of a query or to the underlying database. This technique is often used in *statistical databases*, where a query computes some aggregate function over the dataset [16], and the goal is to preserve the privacy of data elements. In contrast, in our setting the private elements are $(\mathbf{x}, m(\mathbf{x}))$ pairs for a private module $m$, while queries are select-project-join style queries over the provenance relation, rather than aggregate queries.

Privacy in *statistical databases* is typically quantified using *differential privacy*, which requires that the output distribution is *almost* invariant to the inclusion of any particular record (see surveys [17, 18] and the references therein). It is an interesting future direction to see if this notion can be adapted to the domain of workflow provenance.

## 4. INTEGRATING PRIVACY WITH SEARCH AND QUERY

As discussed earlier, users will access `WP` repositories using either keyword search or structured queries. When answering these queries, we must guarantee that the data, module and structural privacy requirements are respected while maximizing the amount of information provided in response to queries. We start by describing initial work on searching and querying workflow specifications, discuss how search and query over provenance interact with privacy concerns, and finally contrast this with related work.

## 4.1 Search and Query

We have developed a technique for keyword search on workflow *specifications* [26]. A *keyword search* consists of a set of comma-separated search *terms* which can match the name, keywords, or description associated with modules; the search *result* is a list of simple workflows. Each simple workflow in the result represents a *minimal view* of some specification in the repository that contains distinct matches to *all* search terms. By the definition of a view (see Section 2), such a result captures the dataflow among search term matches.

For example, suppose a user issues the search "*database, disorder risk*" on a repository of workflows that contains the sample workflow in Figure 1. Note that "*database*" matches both $M4$ and $M5$, and that "*disorder risk*" matches $M2$. The sample workflow contains a match to each term in the keyword search, and the question is what should be returned. One option would be to return the entire hierarchy shown in Figure 1, but this could be overwhelming for the user. A simpler option would be to return a portion of the workflow in which the $\tau$ expansion for M2 is hidden, showing only $W1$, $W2$ and $W4$. The more concise result would be the simple workflow shown in Figure 5, which is the view obtained from the prefix $\{W1, W2, W4\}$. In this view, the dataflow between keywords is easily seen.

This technique can be naturally extended to workflow *executions*. In addition to matching terms with module names, keywords and descriptions, terms could match the names or values of data flowing over the edges in an execution. If a term matches an edge, the connecting modules would be treated as matches.[6] The result would be a view of the execution using the minimal view for the match.

[6]Recall that data can only be written once.

Users may also wish to ask *structural queries* over the repository of specifications and executions (see [5, 24] for provenance query languages which could be extended to our setting). Examples of queries that would match our sample workflow specification include: " Workflow specifications in which *Expand SNP* is performed before *Query PubMed*" and "Workflow specifications in which there is a parallel execution of modules that match *query*." Examples of queries over executions include "Executions in which data items $d1$ or $d2$ are used at some point before executing module $M$, and that produce data item $d19$ at some point after executing $M$", or the more typical provenance-style queries: "Data items that were used to produce data item $d10$" and "Data items that depend on data item $d2$".

While structural queries share with keyword search the need to match terms (e.g., *Expand SNP*, *Query PubMed*, $d10$, *query*), the format of the result is typically specified in a **return** clause. Both matching and the result format therefore have interesting interactions with access views.

### 4.2 Preserving Privacy through Access Views

To preserve privacy while searching and querying, the user's *access view* (see Section 3) must be used for both matching and formatting the result. In particular, search results can be defined as the minimal view that "subsumes" the access view and maximally contains the search keywords. For example, if the user's access view is $\{W1, W2, W4\}$ for our sample search "*database, disorder risk*", then the search matches our sample specification and the result is the view corresponding to the prefix $\{W1, W2, W4\}$ (shown in Figure 5). If the user's access view is $\{W1, W2, W3\}$, then "database" would now match only $M4$, while "disorder risk" would still match $M2$. The search result would then be the view corresponding to the prefix $\{W1, W2\}$. Finally, if the user's access view is $\{W1\}$ or $\{W1, W3\}$, then the specification would not be returned since there is no match for "database".

Similarly, the structural query "Workflows in which *Expand SNP* is performed before *Query PubMed*" would match our sample workflow for the access view $\{W1, W2, W4\}$ but not for $\{W1\}$ or $\{W1, W2\}$. Note that both the match and the result to be returned would be evaluated within the context of the access view.

The result returned by the provenance query "What data was used to produce data item d10?" would also depend on the user's access view. For example, for the access view $\{W1\}$ in Figure 2 the result would be $\{d0, d1\}$, whereas for the access view $\{W1, W2, W4\}$ in Figure 4 the result would be $\{d0, d1, d5, d6, d7, d8, d9\}$.

### 4.3 Related Work

Keyword search has been extensively studied, however, existing approaches are not appropriate for searching workflow repositories. For example, keyword search for graph-structured data (e.g., [3]) and tree-structured data (e.g., [25]) typically return results as smallest trees that contain matches to query keywords. A natural question to ask is whether we can use these approaches by treating a workflow specification as a graph. However, there are two problems with this. First, users expect to see workflows rather than portions of workflows that contain the keywords. Second, workflow specifications involve both dataflow and expansion ($\tau$) edges, and ignoring the difference between these may yield search results that do not capture the dataflow among keyword matches. In contrast, our approach yields search results that are minimal views of matching workflows, and that indicate the connections between the keyword matches.

Keyword search is supported in Kepler [9], Triana [40], and Taverna/myGrid [33]. These approaches allow users to search for *modules* using keywords, but are not extended to workflows. myExperiment [32] allows users to issue a keyword search, and the result is a set of hierarchical workflow specifications, each containing all the keywords. However, the search result is not minimized. Extending this simple approach to executions of workflows, which are even larger than specifications due to repeated executions of modules through parallel execution (map operations) and looping, would be overwhelming to users. Furthermore, the result does not explicitly show connections between the keyword matches.

None of existing work addresses keyword search on workflows in the presence of privacy requirements.

## 5. CONCLUSIONS

In this paper, we discuss how to integrate privacy guarantees in the design of provenance management systems for scientific workflows. We start by identifying privacy concerns – data, module, and structural privacy – and frame several natural questions: (i) Can we formally analyze data, module, and structural privacy, giving provable privacy guarantees for an unlimited/bounded number of provenance queries? (ii) How can we answer search and structural queries over repositories of workflow specifications and their executions, providing as much information as possible to the user while still guaranteeing privacy?

There are many directions for future work. Our initial results for *module privacy* in general workflows show that the the interaction between private and public modules makes the problem much harder. For example, a one-to-one public function (e.g., a data reformatting module) can reveal data that was kept hidden as the input or output of a private module. In addition to privatization (discussed in Section 3), we will explore a variety of ideas, such as using the topology of the network or identifying natural restrictions on the behavior of public modules. We would also like to consider giving *imprecise* values for data, rather than simply hiding data, and explore connections with differential privacy [17, 18]. Although we have sketched initial ideas on *structural privacy*, a formal characterization and study remains to be done. Finally, the issue of *collusion* between users should be studied in our context.

There are also many challenges to efficiently implementing search and querying in light of privacy guarantees. For example, to achieve data and module privacy we need to be able to efficiently identify the data that a user is able to access, which is typically achieved using inverted indices. Since users may have different privileges, we must now manage an inverted index with different user views. We also need to generate search results with respect to user access views. One approach would be to first find the result, oblivious to the privacy requirement. If the result would reveal sensitive information, we gradually "zoom-out" the view by hiding details of composite modules until privacy is achieved. However, this can be expensive as each zoom-out may involve a disk access. Techniques must be developed to efficiently find user-specific results.

## 6. ACKNOWLEDGMENTS

This work was supported in part by the US National Science Foundation grants IIS-0803524, IIS-0629846, IIS-0915438, CCF-0635084 and IIS-0904314, by NSF CAREER award IIS-0845647, and by grant 0937060 to the Computing Research Association for the CIFellows Project. This work was also supported in part by an IBM faculty award.

## 7. REFERENCES

[1] C. C. Aggarwal and P. S. Yu, editors. *Privacy-Preserving Data Mining: Models and Algorithms*. Springer, 2008.

[2] G. Aggarwal, T. Feder, K. Kenthapadi, S. Khuller, R. Panigrahy, D. Thomas, and A. Zhu. Achieving anonymity via clustering. In *PODS*, pages 153–162, New York, NY, USA, 2006. ACM.

[3] S. Agrawal, S. Chaudhuri, and G. Das. DBXplorer: A System for Keyword-Based Search over Relational Databases. In *ICDE*, pages 5–16, 2002.

[4] L. Backstrom, C. Dwork, and J. M. Kleinberg. Wherefore art thou r3579x?: anonymized social networks, hidden patterns, and structural steganography. In *WWW*, pages 181–190, 2007.

[5] C. Beeri, A. Eyal, T. Milo, and A. Pilberg. Monitoring business processes with queries. In *VLDB '07: Proceedings of the 33rd international conference on Very large data bases*, pages 603–614. VLDB Endowment, 2007.

[6] E. Bertino and E. Ferrari. Secure and selective dissemination of XML documents. *ACM Trans. Inf. Syst. Secur.*, 5(3):290–331, 2002.

[7] O. Biton, S. C. Boulakia, S. B. Davidson, and C. S. Hara. Querying and managing provenance through user views in scientific workflows. In *ICDE*, pages 1072–1081, 2008.

[8] O. Biton, S. B. Davidson, S. Khanna, and S. Roy. Optimizing user views for workflows. In *ICDT '09: Proceedings of the 12th International Conference on Database Theory*, pages 310–323, 2009.

[9] S. Bowers and B. Ludäscher. Actor-oriented design of scientific workflows. In *Int. Conf. on Concept. Modeling*, pages 369–384, 2005.

[10] U. Braun, A. Shinnar, and M. Seltzer. Securing provenance. In *USENIX HotSec, The 3rd USENIX Workshop on Hot Topics in Security*, USENIX HotSec, pages 1–5, Berkeley, CA, USA, July 2008. USENIX Association.

[11] A. Chebotko, S. Chang, S. Lu, F. Fotouhi, and P. Yang. Scientific workflow provenance querying with security views. *WAIM*, pages 349–356, July 2008.

[12] E. Damiani, S. D. C. di Vimercati, S. Paraboschi, and P. Samarati. A fine-grained access control system for XML documents. *ACM Trans. Inf. Syst. Secur.*, 5(2):169–202, 2002.

[13] S. Davidson, S. Khanna, T. Milo, D. Panigrahi, and S. Roy. Provenance views for module privacy. Unpublished manuscript, 2011.

[14] S. B. Davidson, S. Khanna, D. Panigrahi, and S. Roy. Preserving module privacy in workflow provenance. *Manuscript available at http://arxiv.org/abs/1005.5543.*

[15] S. B. Davidson, S. Khanna, S. Roy, and S. Cohen-Boulakia. Privacy issues in scientific workflow provenance. In *Proceedings of the 1st International Workshop on Workflow Approaches for New Data-Centric Science*, June 2010.

[16] I. Dinur and K. Nissim. Revealing information while preserving privacy. In *PODS '03: Proceedings of the twenty-second ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 202–210, New York, NY, USA, 2003. ACM.

[17] C. Dwork. Differential privacy: A survey of results. In *TAMC*, pages 1–19, 2008.

[18] C. Dwork. The differential privacy frontier (extended abstract). In *TCC*, pages 496–502, 2009.

[19] W. Fan, C. Y. Chan, and M. N. Garofalakis. Secure XML querying with security views. In *SIGMOD Conference*, pages 587–598, 2004.

[20] J. Freire, C. T. Silva, S. P. Callahan, E. Santos, C. E. Scheidegger, and H. T. Vo. Managing rapidly-evolving scientific workflows. In *IPAW*, volume 4145 of *LNCS*, pages 10–18. Springer, 2006.

[21] A. Gil, W. K. Cheung, V. Ratnakar, and K. kin Chan. Privacy enforcement in data analysis workflows. In *PEAS*, 2007.

[22] Y. Gil and C. Fritz. Reasoning about the appropriate use of private data through computational workflows. In *Intelligent Information Privacy Management, Papers from the AAAI Spring Symposium*, pages 69–74, March 2010.

[23] R. Hasan, R. Sion, and M. Winslett. Introducing secure provenance: problems and challenges. In *StorageSS '07*, pages 13–18, New York, NY, USA, 2007. ACM.

[24] G. Karvounarakis, Z. G. Ives, and V. Tannen. Querying data provenance. In *SIGMOD Conference*, pages 951–962, 2010.

[25] Z. Liu and Y. Chen. Identifying Meaningful Return Information for XML Keyword Search. In *SIGMOD*, 2007.

[26] Z. Liu, Q. Shao, and Y. Chen. Searching workflows with hierarchical views. *PVLDB*, 3(1), 2010.

[27] J. Lyle and A. Martin. Trusted computing and provenance: Better together. In *TaPP '10: 2nd Workshop on the Theory and Practice of Provenance*, 2010.

[28] A. Machanavajjhala, D. Kifer, J. Gehrke, and M. Venkitasubramaniam. L-diversity: Privacy beyond k-anonymity. *ACM Trans. Knowl. Discov. Data*, 1(1):3, 2007.

[29] G. Miklau and D. Suciu. A formal analysis of information disclosure in data exchange. In *SIGMOD*, 2004.

[30] L. Moreau, J. Freire, J. Futrelle, R. E. McGrath, J. Myers, and P. Paulson. The open provenance model: An overview. In *IPAW*, pages 323–326, 2008.

[31] R. Motwani, S. U. Nabar, and D. Thomas. Auditing SQL queries. In *ICDE*, pages 287–296, 2008.

[32] myExperiment. http://www.myexperiment.org/workflows.

[33] T. Oinn, M. Addis, J. Ferris, D. Marvin, M. Senger, R. Greenwood, K. Carver, M. G. Pocock, A. Wipat, and P. Li. Taverna: a tool for the composition and enactment of bioinformatics workflows. *Bioinformatics*, 20(1):3045–3054, 2003.

[34] R. Ramakrishnan and J. Gehrke. *Database Management Systems*. McGraw-Hill, 3rd edition, 2002.

[35] V. Rastogi, M. Hay, G. Miklau, and D. Suciu. Relationship privacy: output perturbation for queries with joins. In *PODS*, pages 107–116, 2009.

[36] S. S. Shapiro. Privacy by design: moving from art to practice. *Commun. ACM*, 53(6):27–29, 2010.

[37] J. Stoyanovich and I. Pe'er. MutaGeneSys: estimating individual disease susceptibility based on genome-wide SNP array data. *Bioinformatics*, 24(3):440–442, 2008.

[38] P. Sun, Z. Liu, S. B. Davidson, and Y. Chen. Detecting and resolving unsound workflow views for correct provenance analysis. In *SIGMOD '09: Proceedings of the 35th SIGMOD international conference on Management of data*, pages 549–562, New York, NY, USA, 2009. ACM.

[39] L. Sweeney. k-anonymity: a model for protecting privacy. *Int. J. Uncertain. Fuzziness Knowl.-Based Syst.*, 10(5):557–570, 2002.

[40] I. Taylor, M. Shields, I. Wang, and A. Harrison. The Triana Workflow Environment: Architecture and Applications. In I. Taylor, E. Deelman, D. Gannon, and M. Shields, editors, *Workflows for e-Science*, pages 320–339. Springer, New York, 2007.

[41] V. S. Verykios, E. Bertino, I. N. Fovino, L. P. Provenza, Y. Saygin, and Y. Theodoridis. State-of-the-art in privacy preserving data mining. *SIGMOD Rec.*, 33(1):50–57, 2004.