A Propagation Model for Provenance Views of Public/Private Workflows

Susan B. Davidson
University of Pennsylvania
susan@cis.upenn.edu

Tova Milo
Tel Aviv University
milo@cs.tau.ac.il

Sudeepa Roy*
University of Washington
sudeepa@cs.washington.edu

ABSTRACT

We study the problem of concealing functionality of a proprietary or private module when provenance information is shown over repeated executions of a workflow which contains both *public* and *private* modules. Our approach is to use *provenance views* to hide carefully chosen subsets of data over all executions of the workflow to ensure Γ-privacy: for each private module and each input x, the module's output $f(x)$ is indistinguishable from $\Gamma-1$ other possible values given the visible data in the workflow executions. We show that Γ-privacy cannot be achieved simply by combining solutions for individual private modules; data hiding must also be *propagated* through public modules. We then examine how much additional data must be hidden and when it is safe to stop propagating data hiding. The answer depends strongly on the workflow topology as well as the behavior of public modules on the visible data. In particular, for a class of workflows (which include the common tree and chain workflows), taking private solutions for each private module, augmented with a *public closure* that is *upstream-downstream safe*, ensures Γ-privacy. We define these notions formally and show that the restrictions are necessary. We also study the related optimization problems of minimizing the amount of hidden data.

Categories and Subject Descriptors

F.2.0 [**Analysis of Algorithms And Problem Complexity**]: General; H.2.0 [**Database Management**]: General—*Security, integrity, and protection*; H.2.8 [**Database Management**]: Database applications—*Scientific databases*

General Terms

Algorithms, Theory

Keywords

workflows, provenance, privacy, optimization

*This work was done while the author was at the University of Pennsylvania.

EDBT/ICDT '13, March 18 - 22 2013, Genoa, Italy

1. INTRODUCTION

Workflow provenance has been extensively studied, and is increasingly captured in workflow systems to ensure reproducibility, enable debugging, and verify the validity and reliability of results. However, as pointed out in [18], there is a tension between provenance and privacy: Confidential intermediate data may be shown (*data privacy*); the functionality of proprietary modules may become exposed by showing the input and output values to that module over all executions of the workflow (*module privacy*); and the exact execution path taken in a specification, hence details of the connections between data, may be revealed (*structural privacy*). An increasing amount of attention is therefore being paid to specifying privacy concerns, and developing techniques to guarantee that these concerns are addressed [33, 35, 7, 8].

This paper focuses on privacy of module functionality, in particular in the general – and common – setting in which proprietary (*private*) modules are used in workflows which also contain non-proprietary (*public*) modules, whose functionality is assumed to be known by users. There are proprietary modules for tasks like gene sequencing, protein folding, medical diagnoses, that are commercially available and are combined with other modules in a *workflow* for different biological or medical experiments [2, 1]. The functionality of these proprietary modules (*i.e.* what result will be output for a given input) is not known, and owners of these proprietary modules would like to ensure that their functionality is not revealed when the provenance information is published. In contrast for a public module (*e.g.* a reformatting or sorting module), given an input to the module a user can construct the output even if the exact algorithm used by the module is not known by users (*e.g.* Merge sort vs Quick sort).

Following [17], the approach we use is to extend the notion of ℓ-diversity [30] to the workflow setting by carefully choosing a subset of intermediate input/output data to hide over *all* executions of the workflow so that each private module is "Γ-private": for every input x, the actual value of the output of the module, $f(x)$, is indistinguishable from $\Gamma - 1$ other possible values w.r.t. the visible data values in the provenance information (in Section 6 we discuss ideas related to differential privacy). The complexity of the problem arises from the fact that modules interact with each other through data flow defined by the workflow structure, and therefore merely hiding subsets of inputs/outputs for private modules may not guarantee their privacy when embedded in a workflow. We consider workflows with directed acyclic graph (DAG) structure, that are commonly used in practice [3], contain common chain and tree workflows, and comprise a fundamental yet non-trivial class of workflows for analyzing module privacy.

As an example, consider a private module m_2, which we assume is non-constant. Clearly, when executed in isolation as a *standalone*

module, then either hiding all its inputs or hiding all its outputs over all executions guarantees privacy for any privacy parameter Γ. However, suppose m_2 is embedded in a simple chain workflow $m_1 \longrightarrow m_2 \longrightarrow m_3$, where both m_1 and m_3 are public, equality modules. Then even if we hide *both* the input and output of m_2, their values can be retrieved from the input to m_1 and the output from m_3. Note that the same problem would arise if m_1 and m_3 were invertible functions, e.g. reformatting modules, a common case in practice.

In [17], we showed that in a workflow with only private modules (an *all-private workflow*) the problem has a simple, elegant solution: If a set of hidden input/output data guarantees Γ-standalone-privacy for a private module, then if the module is placed in an all-private workflow where a superset of that data is hidden, then Γ-workflow-privacy is guaranteed for that module in the workflow. In other words, in an all-private workflow, hiding the union of the corresponding hidden data of the individual modules guarantees Γ-workflow-privacy for all of them. Clearly, as illustrated above, this does not hold when the private module is placed in a workflow which contains public and private modules (a *public/private workflow*). In [17] we therefore explored *privatizing* public modules, i.e. hiding the names of carefully selected public modules so that their function is no longer known, and then hiding subsets of input/output data to ensure their Γ-privacy. Returning to the example above, if it were no longer known that m_1 was an equality module then hiding the input to m_2 (output of m_1) would be sufficient. Similarly, if m_3 was privatized then hiding the output of m_2 (input to m_3) would be sufficient. It may appear that merging some public modules with preceding or succeeding private modules may give a workflow with all private modules and then the methods from [17] can be applied. However, merging may be difficulty for workflows with complex network structure, large amounts of data may be needed to be hidden, and more importantly, it may not be possible to merge at all when the structure of the workflow is known.

Although privatization is a reasonable approach in some cases, there are many practical scenarios where it cannot be employed, *e.g.* when the workflow specification (the module names and connections) is already known to the users, or when the identity of the privatized public module can be discovered through the structure of the workflow and the names or types of its inputs/outputs.

To overcome this problem, we propose an alternative novel solution, based on the propagation of data hiding through public modules. Returning to our example, if the input to m_2 were hidden then the input to m_1 would also be hidden, although the user would still know that m_1 was the equality function. Similarly, if the output of m_2 were hidden then the output of m_3 would also be hidden; again, the user would still know that m_3 was the equality function. While in this example things appear to be simple, several technically challenging issues must be addressed when employing such a propagation model: 1) whether to propagate hiding upward (e.g. to m_1) or downward (e.g. to m_3); 2) how far to propagate data hiding; and 3) which data of public modules must be hidden. Overall the goal is to guarantee that the functionality of private modules is not revealed while minimizing the amount of hidden data.

In this paper we focus on *downward* propagation, for reasons that will be discussed in Section 3. We define a class of workflows, called single-private-predecessor workflows or simply *single-predecessor workflows*, which include the common tree and chain workflows. For these workflows, we show the following strong result: taking a solution for Γ-standalone-privacy of each private module, augmenting the solution with specially chosen input/output data of certain public modules, and hiding the union of these augmented solutions will ensure Γ-workflow privacy for all private modules. In particular, the augmented solution should ensure *upstream-downstream safety (`UD-safety`)* for modules in the *public closure* (up to a successor private module) of private modules. We define these notions formally in Section 3. We also show that single-predecessor workflows is the largest class of workflows for which propagation of data hiding only within the public closure suffices.

Since data may have different *costs* in terms of hiding, and there may be different safe subsets for private modules as well as different `UD-safe` subsets for public modules, the next problem we address is finding a minimum cost composition of the individual solutions: first identify safe and `UD-safe` subsets for the private and public modules, respectively, then assemble them together optimally. The complexity of identifying safe subsets for a private module was studied in [17] and the problem was shown to be NP-hard (in EXP-time) in the number of module attributes. Here we show that identifying `UD-safe` subsets for public modules is of similar complexity: even deciding whether a given subset is `UD-safe` for a module is coNP-hard in the number of inputs and outputs. We note however that this is not as negative as it might appear, since the number of inputs/outputs of individual modules is not high; furthermore, the computation may be performed as a preprocessing step or expert knowledge (from the module designer) can be used. We show that, for chain and tree-shaped workflows, the optimization problem has a poly-time solution in the size of the workflow and the maximum number of safe/`UD-safe` subsets for private/public modules. The algorithm can also be applied to general single-predecessor workflows where the public closures have chain or tree shapes. In contrast, when the public closure has an arbitrary DAG shape, the problem becomes NP-hard (in EXP-time) in the size of the public closure.

We then consider general acyclic workflows, and give a sufficient condition to ensure Γ-privacy that is not the trivial solution of hiding all data in the workflow. In contrast to single-predecessor workflows, hiding data within a public closure no longer suffices; data hiding must continue through other private modules to the entire downstream workflow. In return, the requirement from data hiding for public modules is somewhat weaker here: hiding must only ensure that the module is *downstream-safe* (`D-safe`), which typically involves fewer input/output data than `UD-safe`ty.

The remainder of the paper is organized as follows: Our workflow model and notions of standalone- and workflow-module privacy are given in Section 2. Section 3 describes our propagation model, defines upstream-downstream-safety and single-predecessor workflows, and states the privacy theorem for single-predecessor workflows. We give the proof of privacy theorem in Section 4 and discuss the related optimization problem. General public/private workflows are considered in Section 5. We review related work in Section 6 and conclude in Section 7.

2. PRELIMINARIES

We start by reviewing the formal definitions and notions of module privacy from [17], and then extend them to the context studied in this paper.[1] Readers familiar with the definitions and results in [17] can move directly to Section 3.

2.1 Modules, Workflows and Relations

Modules. A module m with a set I of input data and a set O of (computed) output data is modeled as a relation R. R has the set of attributes $A = I \cup O$, and satisfies the functional dependency $I \rightarrow O$. We assume that $I \cap O = \emptyset$ and will refer to I and O as the *input attributes* and *output attributes* of R respectively.

[1]The example in this section is also taken from [17].

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	1	0
1	1	1	0	1

(a) R_1: Functionality of m_1

a_1	a_2	a_3	a_4	a_5	a_6	a_7
0	0	0	1	1	1	0
0	1	1	1	0	0	1
1	0	1	1	0	0	1
1	1	1	0	1	1	1

(b) R: Workflow executions

$I \setminus H$	$O \setminus H$	
a_1	a_3	a_5
0	0	1
0	1	0
1	1	0
1	1	1

(c) $R' = \Pi_{A \setminus H}(R_1)$

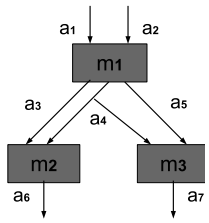


Figure 1: Module and workflow executions as relations, and view

We assume that the values of each attribute $a \in A$ come from a finite but arbitrarily large domain Δ_a, and let $\texttt{Dom} = \prod_{a \in I} \Delta_a$ and $\texttt{CoDom} = \prod_{a \in O} \Delta_a$ denote the *domain* and *co-domain* of the module m respectively.[2] The relation R thus represents the (possibly partial) function $m : \texttt{Dom} \rightarrow \texttt{CoDom}$ and tuples in R describe executions of m, namely for every $t \in R$, $\Pi_O(\mathbf{t}) = m(\Pi_I(\mathbf{t}))$. We overload the standard notation for projection, $\Pi_A(R)$, and use it for a tuple $\mathbf{t} \in R$. Thus $\Pi_A(\mathbf{t})$, for a set A of attributes, denotes the projection of $\mathbf{t}$ to the attributes in A.

Workflows. A workflow W consists of a set of modules $m_1, \cdots, m_n$, connected as a DAG (*e.g.* the workflow in Figure 1). We assume that (1) the output attributes of distinct modules are disjoint, namely $O_i \cap O_j = \emptyset$, for $i \neq j$ (i.e. each data item is produced by a unique module); and (2) whenever an output of a module m_i is fed as input to a module m_j the corresponding output and input attributes of m_i and m_j are the same. The DAG shape of the workflow guarantees that these requirements are not contradictory.

We model executions of W as a relation R over the set of attributes $A = \cup_{i=1}^{n} A_i$, satisfying the set of functional dependencies $F = \{I_i \rightarrow O_i : i \in [1, n]\}$. Each tuple in R describes an execution of the workflow W. In particular, for every $t \in R$, and every $i \in [1, n]$, $\Pi_{O_i}(\mathbf{t}) = m_i(\Pi_{I_i}(\mathbf{t}))$. One can think of R as containing (possibly a subset of) the join of the individual module relations.

EXAMPLE 1. *Figure 1 shows a workflow involving three modules m_1, m_2, m_3 with boolean input and output attributes implementing the following functions: (i) m_1 computes $a_3 = a_1 \vee a_2$, $a_4 = \neg(a_1 \wedge a_2)$ and $a_5 = \neg(a_1 \oplus a_2)$, where $\oplus$ denotes XOR; (ii) m_2 computes $a_6 = \neg(a_3 + a_4)$; and (iii) m_3 computes $a_7 = a_4 \wedge a_6$. The relational representation (functionality) R_1 of module m_1 with the functional dependency $a_1 a_2 \longrightarrow a_3 a_4 a_5$ is shown in Figure 1a. For clarity, we have added I (input) and O (output) above the attribute names to indicate their role. The relation R describing the workflow executions is shown in Figure 1b which has the functional dependencies $a_1 a_2 \longrightarrow a_3 a_4 a_5$, $a_3 a_4 \longrightarrow a_6$, $a_4 a_5 \longrightarrow a_7$ from modules m_1, m_2, m_3 respectively.*

Data sharing refers to an output attribute of a module acting as input to more than one module (hence $I_i \cap I_j \neq \emptyset$ for $i \neq j$). In the example above, attribute a_4 is shared by both m_2 and m_3.

[2]We distinguish between the possible range O of the function m that we call *co-domain* and the *actual range* $\{\mathbf{y} : \exists \mathbf{x} \in I \; s.t. \; \mathbf{y} = m(\mathbf{x})\}$

2.2 Module Privacy

We consider the privacy of a single module, which is called *standalone module privacy*, then privacy of modules when they are connected in a workflow, which is called *workflow module privacy*. We study this given two types of modules, *private* modules (the focus of [17]) and *public* modules (the focus here).

Standalone module privacy. Our approach to ensuring standalone module privacy, for a module represented by the relation R, is to hide a carefully chosen subset H of R's attributes (called *hidden attributes*). In other words, we project R on a restricted subset $A \setminus H$, where A is the set of all attributes of m. The set $A \setminus H$ is called *visible attributes*. The users are allowed access only to the view $R' = \Pi_{A \setminus H}(R)$.

One may distinguish two types of modules. (1) *Public modules* whose behavior is fully known to users. Here users have a prior knowledge about the full content of R and, even if given only the view R', they are able to fully (and exactly) reconstruct R. Examples include reformatting or sorting modules. (2) *Private modules* where such a priori knowledge does not exist. Here, the only information available to users, on the module's behavior, is the one given by R'. Examples include proprietary software, e.g. a genetic disorder susceptibility module.

Given a view (projected relation) R' of a private module m, the *possible worlds* of m are all the possible full relations (over the same schema as R) that are consistent with the view R'. Formally,

DEFINITION 1. *Let m be a private module with a corresponding relation R, having input and output attributes I and O respectively. Let $A = I \cup O$ be the set of all attributes. Given a set of hidden attributes H, the set of* **possible worlds** *for R with respect to H, denoted $\texttt{Worlds}(R, H)$, consists of all relations R' over the same schema as R that satisfy the functional dependency $I \rightarrow O$, and where $\Pi_{A \setminus H}(R') = \Pi_{A \setminus H}(R)$.*

To guarantee privacy of a module m, the view R' should ensure some level of uncertainly with respect to the value of the output $m(\Pi_I(\mathbf{t}))$, for tuples $t \in R$. To define this, we introduce the notion of Γ-standalone-privacy, for a given parameter $\Gamma \geq 1$. Informally, a view R' is Γ-standalone-private if for every $t \in R$, $\texttt{Worlds}(R, H)$ contains at least Γ distinct output values that could be the result of $m(\Pi_I(\mathbf{t}))$.

DEFINITION 2. *Let m be a private module with a corresponding relation R having input and output attributes I and O resp. Then m is* **Γ-standalone-private** *with respect to a set of hidden attributes H, if for every tuple $\mathbf{x} \in \Pi_I(R)$, $|\text{OUT}_{\mathbf{x},m,H}| \geq \Gamma$, where $\text{OUT}_{\mathbf{x},m,H} = \{\mathbf{y} \mid \exists R' \in \texttt{Worlds}(R, H), \; \exists \mathbf{t}' \in R' \; s.t \; \mathbf{x} = \Pi_I(\mathbf{t}') \wedge \mathbf{y} = \Pi_O(\mathbf{t}')\}$.*[3]

If m is Γ-standalone-private *with respect to hidden attributes H, then we call H a* safe subset *for m and Γ.*

A module cannot be differentiated from its possible worlds with respect to the visible attributes, and therefore, whether the original module, or one from its possible worlds is being used cannot be recognized. Hence, Γ-standalone-privacy implies that for *any* input the adversary cannot guess m's output with probability $> \frac{1}{\Gamma}$, even if the module is executed an arbitrary number of times.

EXAMPLE 2. *Returning to module m_1, suppose the hidden attributes are $H = \{a_2, a_4\}$ resulting in the view R' in Figure 1c.*

[3]In [17], we (equivalently) defined privacy with respect to visible attributes V instead of hidden attributes H, and we used the notation "$\text{OUT}_{\mathbf{x},m}$ with respect to V" instead of $\text{OUT}_{\mathbf{x},m,H}$.

For clarity, we have added $I \setminus H$ (visible input) and $O \setminus H$ (visible output) above the attribute names to indicate their role. Naturally, $R_1 \in \texttt{Worlds}(R_1, H)$, and we can check that overall there are 64 relations in $\texttt{Worlds}(R_1, H)$.

Furthermore, it can be verified that, if $H = \{a_2, a_4\}$, then for all $\mathbf{x} \in \Pi_I(R_1)$, $|\text{OUT}_{\mathbf{x},m_1,H}| \geq 4$, so $\{a_1, a_3, a_5\}$ is safe for m_1 and $\Gamma = 4$. As an example, when $\mathbf{x} = (0,0)$, $\text{OUT}_{\mathbf{x},m,H} \supseteq \{(0,\underline{0},1), (0,\underline{1},1), (1,\underline{0},0), (1,\underline{1},0)\}$ (hidden attributes are underlined) – we can define four possible worlds that map $(0,0)$ to these outputs (see [17] for details). Also, hiding any two output attributes from $O = \{a_3, a_4, a_5\}$ ensures standalone privacy for $\Gamma = 4$, e.g. *if $H = \{a_2, a_4\}$, then the input $(0,0)$ can be mapped to one of $(0,\underline{0},\underline{0})$, $(0,\underline{0},\underline{1})$, $(0,\underline{1},\underline{0})$ and $(0,\underline{1},\underline{1})$; this holds for other assignments of input attributes as well. However, $H = \{a_1, a_2\}$ (input attributes) is not safe for $\Gamma = 4$: for any input $\mathbf{x}$, $\text{OUT}_{\mathbf{x},m,H} = \{(0,1,1), (1,1,0), (1,0,1)\}$, containing only three possible output tuples.*

Workflow Module Privacy. To define privacy in the context of a workflow, we first extend the notion of *possible worlds* to a workflow view. Consider the view $R' = \Pi_{A\setminus H}(R)$ of the relation R of a workflow W, where A is the set of all attributes across all modules in W. Since W may contain private as well as public modules, a possible world for R' is a full relation that not only agrees with R' on the content of the visible attributes and satisfies the functional dependency, but is also consistent with respect to the expected behavior of the public modules. In the following definitions, $m_1, \cdots, m_n$ are the modules in W and $F = \{I_i \rightarrow O_i : 1 \leq i \leq n\}$ is the set of functional dependencies in R.

DEFINITION 3. *The set of* **possible worlds for the workflow relation** *R with respect to hidden attributes H (denoted by $\texttt{Worlds}(R, H)$) consists of all relations R' over the same attributes as R that satisfy (1) the functional dependencies in F, (2) $\Pi_{A\setminus H}(R') = \Pi_{A\setminus H}(R)$, and (3) $\Pi_{O_i}(\mathbf{t}') = m_i(\Pi_{I_i}(\mathbf{t}'))$ for every public module m_i in W and every tuple $\mathbf{t}' \in R'$.*

We can now define the notion of Γ-workflow-privacy, for a given parameter $\Gamma \geq 1$. Informally, a view R' is Γ-workflow-private if for every tuple $t \in R$, and every private module m_i in the workflow, the possible worlds $\texttt{Worlds}(R, H)$ contain at least Γ distinct output values that could be the result of $m_i(\Pi_{I_i}(\mathbf{t}))$.

DEFINITION 4. *A private module m_i in a workflow W is* **Γ-workflow-private** *with respect to a set of hidden attributes H, if for every tuple $\mathbf{x} \in \Pi_{I_i}(R)$, $|\text{OUT}_{\mathbf{x},W,H}| \geq \Gamma$, where $\text{OUT}_{\mathbf{x},W,H} = \{\mathbf{y} \mid \exists R' \in \texttt{Worlds}(R, H), s.t., \forall\, \mathbf{t}' \in R',\ \mathbf{x} = \Pi_{I_i}(\mathbf{t}') \Rightarrow \mathbf{y} = \Pi_{O_i}(\mathbf{t}')\}$.*

W is called Γ-private *if every private module m_i in W is Γ-workflow-private. If W (resp. m_i) is* Γ-private *(Γ-workflow-private) with respect to H, then we call H a* safe subset *for Γ-privacy of W (Γ-workflow-privacy of m_i).*

Similar to standalone module privacy, Γ-workflow-privacy ensures that for any input to a module m_i, the output cannot be guessed with probability $\geq \frac{1}{\Gamma}$ even if m_i belongs to a workflow with arbitrary DAG structure and interacts with other modules with known or unknown functionality, and even the workflow is executed an arbitrary number of times. For simplicity, the above definition assume that the privacy requirement of every module m_i is the same Γ. The results and proofs in this paper remain unchanged when different modules m_i have different privacy requirements Γ_i. Note that there is a subtle difference in workflow privacy of a module defined as above and standalone-privacy (Definition 2); the former uses the logical implication operator ($\Rightarrow$) for defining $\text{OUT}_{\mathbf{x},W,H}$ while the latter uses conjunction ($\wedge$) for defining $\text{OUT}_{\mathbf{x},m,H}$. This is due to the fact that some modules are not *onto*[4]; and as a result the input x itself may not appear in any execution of the possible world R'. Nevertheless, there is an alternative definition of module m_i that maps x to y and can be used in the workflow for R' consistently with the visible data.

2.3 Composability Theorem and Optimization

Given a workflow W and parameter Γ, there may be several incomparable (in terms of set inclusion) safe subsets H for the (standalone) modules in W and for the workflow as a whole. Some of the corresponding R' views may be preferable to others, e.g. they provide users with more useful information, allow more common/critical user queries to be answered, etc. If $\texttt{cost}(H)$ denotes the penalty of hiding the attributes in H, a natural goal is to choose a safe subset H that minimizes $\texttt{cost}(H)$. A particular instance of the problem is when the cost function is additive: each attribute a has some penalty value $\texttt{cost}(a)$ and the penalty of hiding H is $\texttt{cost}(H) = \Sigma_{a\in H}\texttt{cost}(a)$.

On the negative side, it was shown in [17] that the corresponding decision problem is hard in the number of attributes, even for a single module and even in the presence of an oracle that tests whether a given attribute subset is safe. On the positive side, however, it was shown that *when the workflow consists only of private modules* (we call these "*all-private*" workflows), once privacy has been analyzed for the individual modules, the results can be lifted to the whole workflow. In particular, the following theorem says that, hiding the union of hidden attributes of standalone-private solutions of the individual modules in an all-private workflow guarantees Γ-workflow-privacy for all of them.

THEOREM 1. **(Composability Theorem for All-private Workflows [17])** *Let W be a workflow consisting only of private modules $m_1, \cdots, m_n$. For each $i \in [1, n]$, let $H_i \subseteq A_i$ be a set of safe hidden attributes for Γ-standalone-privacy of m_i. Then the workflow W is Γ-private with respect to hidden attributes $H = \bigcup_{i=1}^{n} H_i$.*

It was also observed in [17] that the number of attributes of individual modules can be much smaller than the total number of attributes in a workflow, and that a proprietary module may be used in many different workflows. Therefore, the obvious brute-force algorithm, which is essentially the best possible, can be used (possibly as a pre-processing step) to find all standalone-private solutions of individual modules. Then any set of "local solutions" for each module can be composed to give a global feasible solution. Moreover, the composability theorem ensures that the private solutions are valid even with respect to future workflow executions which have not yet been recorded in the workflow relation.

Given Theorem 1, [17] focused on a modified optimization problem: find a workflow-private solution by optimally combining the standalone-private solutions. This optimization problem, which we refer to as **optimal composition problem**, remains NP-hard even in the simplest scenario, and therefore, [17] proposed efficient approximation algorithms.

3. PRIVACY VIA PROPAGATION

Workflows with both public and private modules are harder to handle than workflows with all private modules. In particular, the

[4]For a function $f : D \rightarrow C$, D is the *domain*, C is the *co-domain*, and $R = \{y \in C : \exists x \in D, f(x) = y\}$ is the *range*. The function f is *onto* if $C = R$.

composability theorem (Theorem 1) does not hold any more. To see why, we revisit the example mentioned in the introduction.

EXAMPLE 3. *Consider a workflow with three modules m_1, m_2 and m_3 as shown in Figure 2a. For simplicity, assume that all modules have a boolean input and a boolean output, and implement the equality function (*i.e.*, $a_1 = a_2 = a_3 = a_4$). Module m_2 is private, and the modules m_1, m_3 are public. When the private module m_2 is standalone, it can be verified that either hiding its input a_2 or hiding its output a_3 guarantees Γ-standalone-privacy for $\Gamma = 2$. However, in the workflow, if a_1 and a_4 are visible then the actual values of a_2 and a_3 can be found exactly since it is known that the public modules m_1, m_3 are equality modules.*

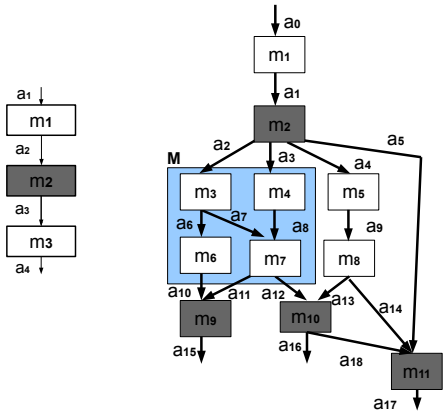
One intuitive way to overcome this problem is to propagate the hiding of data through the problematic public modules, i.e., to hide the attributes of public models that may disclose information about hidden attributes of private modules. To continue with the above example, if we choose to hide input a_2 (respectively, output a_3) to protect the privacy of module m_2, then we propagate the hiding *upstream* (resp. *downstream*) to the public modules and hide the input attribute a_1 of m_1 (respectively, the output attribute a_4 of m_3).

The workflow in the above example has a simple structure, and the functionality of its component modules is also simple. In general, three main issues arise when employing such a propagation model: (1) upward vs. downward propagation; (2) repeated propagation; and (3) choosing which attributes to hide. We discuss these issues next.

3.1 Upstream vs. Downstream propagation

Which form of propagation can be used depends on the safe subsets chosen for the private modules as well as properties of the public modules. To see this, consider again Example 3, and assume now that public module m_1 computes some constant function (*e.g.*, $m_1(0) = m_1(1) = 0$). If input attribute a_2 for module m_2 is hidden, then using upward propagation to hide the input attribute a_1 of m_1 does not preserve the Γ-workflow-privacy of m_2 for $\Gamma > 1$. This is because it suffices to look at the (visible) output attribute $a_3 = 0$ of m_2 to know that $m_2(0) = 0$. In general, upward propagation from a subset of input attributes which gives Γ_1-standalone-privacy for a private module m will only yield Γ_2-workflow-privacy for m, where $\Gamma_1 \geq \Gamma_2$. It is possible that $\Gamma_1 >> \Gamma_1$ unless upstream public modules are onto functions; in the worst case, if upstream modules are constant functions, then $\Gamma_2 = 1$ whereas Γ_1 can be arbitrarily large. Unfortunately, it is not common for modules to be onto functions (e.g. some output values may be well-known to be non-existent).

In contrast, when the privacy of a private module is achieved by *hiding output attributes only*, using downstream propagation it is possible to achieve the same privacy guarantee in the workflow as with the standalone case without imposing any restrictions on the public modules. Observe that safe subsets of output attributes always exist for all private modules – one can always hide *all* the output attributes. They may incur higher cost than that of an optimal subset of both input and output attributes, but, in terms of privacy, by hiding only output attributes one does not harm its maximum achievable privacy. In particular, it is not hard to see that hiding all input attributes can give a maximum of Γ_1-workflow-privacy, where Γ_1 is the size of the range of the module. On the other hand hiding all output attributes can give a maximum of Γ_2-workflow-privacy, where Γ_2 is the size of the co-domain of the module, which can be much larger than the actual range. *We therefore focus in the rest of this paper on safe subsets that contain only output attributes.*



(a) (b)

Figure 2: (a) Propagation model, (b) A single-predecessor workflow. White modules are public, grey are private; the box denotes the composite module M for $H_2 = \{a_3\}$.

3.2 Repeated Propagation

Consider again Example 3, and assume now that public module m_3 sends its output to another public module m_4 that implements an equality function (or a one-one invertible function). Even if the output of m_3 is hidden as described above, if the output of m_4 remains visible, the privacy of m_2 is again jeopardized since the output of m_3 can be inferred using the inverse function of m_4. We thus need to propagate the attribute hiding to m_4 as well. More generally, we need to propagate the attribute hiding repeatedly, through all adjacent public modules, until we reach another private module.

To formally define the *closure* of public modules to which attributes hiding must be propagated, we use the notion of a *public path*. Intuitively, there is a public path from a public module m_i to a public module m_j if we can reach m_j from m_i by a path comprising only public modules. In what follows, we define both directed and undirected public paths; recall that $A_i = I_i \cup O_i$ denotes the set of input and output attributes of module m_i.

DEFINITION 5. *A public module m_1 has* **a directed (resp. an undirected) public path** *to a public module m_2 if there is a sequence of public modules $m_{i_1}, m_{i_2}, \cdots, m_{i_j}$ such that $m_{i_1} = m_1$, $m_{i_j} = m_2$, and for all $1 \leq k < j$, $O_{i_k} \cap I_{i_{k+1}} \neq \emptyset$ (resp. $A_{i_k} \cap A_{i_{k+1}} \neq \emptyset$).*

This notion naturally extends to module attributes. We say that an input attribute $a \in I_1$ of a public module m_1 has an (un)directed public path to a public module m_2 (and also to any output attribute $b \in O_2$), if there is an (un)directed public path from m_1 to m_2. The set of public modules to which attribute hiding will be propagated can now be defined as follows.

DEFINITION 6. *Given a private module m_i and a set of hidden output attributes $h_i \subseteq O_i$ of m_i, the* **public-closure** *$C(h_i)$ of m_i with respect to h_i is the set of public modules reachable from some attribute in h_i by an undirected public path.*

EXAMPLE 4. *We illustrate these notions using Figure 2b. The public module m_4 has an undirected public path to the public module m_6 through the modules m_7 and m_3. For private module m_2, if hidden output attributes $h_2 = \{a_2\}$, $\{a_3\}$, or $\{a_2, a_3\}$, the public closure $C(h_2) = \{m_3, m_4, m_6, m_7\}$. For $h_2 = \{a_4\}$, $C(h_2) = \{m_5, m_8\}$. In our subsequent analysis, it will be convenient to view the public-closure as a virtual* ***composite module*** *that encapsulates the sub-workflow and behaves like it. For instance,*

a_1	a_2	a_3	a_4
0	0	0	0
0	1	0	1
1	0	1	0
1	1	1	1

(a) R_1

a_1	a_2	a_3	a_4
0	0	1	0
0	1	1	0
1	0	0	1
1	1	0	1

(b) R_2

Figure 3: UD−safe solutions for modules

the box in Figure 2b denotes the composite module M representing $C(\{a_2\})$, that has input attributes a_2, a_3, and output attributes a_{10}, a_{11} and a_{12}.

3.3 Selection of hidden attributes

In Example 3, it is fairly easy to see which attributes of m_1 or m_3 need to be hidden to preserve the privacy of m_2. For the general case, where the public modules are not as simple as equality functions, to determine which attributes of a given public module need to be hidden we use the notions of *upstream* and *downstream* safety. To define them we use the following notion of tuple equivalence with respect to a given set of hidden attributes. Recall that A denotes the set of all attributes in the workflow; we also use bold-faced letters $\mathbf{x}, \mathbf{y}, \mathbf{z}$, etc. to denote tuples in the workflow or module relations with one or more attributes.

DEFINITION 7. *Given two tuples $\mathbf{x}$ and $\mathbf{y}$ on a subset of attributes $B \subseteq A$, and a subset of hidden attributes $H \subseteq A$, we say that $\mathbf{x} \equiv_H \mathbf{y}$ iff $\Pi_{B \setminus H}(\mathbf{x}) = \Pi_{B \setminus H}(\mathbf{y})$.*

DEFINITION 8. *Given a subset of hidden attributes $H \subseteq A_i$ of a public module m_i, m_i is called*

- ***downstream-safe** (or, **D-safe** in short) with respect to H if for any two equivalent input tuples $\mathbf{x}, \mathbf{x}'$ to m_i with respect to H, their outputs are also equivalent:*

$$\left[\mathbf{x} \equiv_H \mathbf{x}'\right] \Rightarrow \left[m_i(\mathbf{x}) \equiv_H m_i(\mathbf{x}')\right],$$

- ***upstream-safe** (or, **U-safe** in short) with respect to H if for any two equivalent outputs $\mathbf{y}, \mathbf{y}'$ of m_i with respect to H, all of their preimages are also equivalent:*

$$\left[(\mathbf{y} \equiv_H \mathbf{y}') \wedge (m_i(\mathbf{x}) = \mathbf{y}, m_i(\mathbf{x}') = \mathbf{y}')\right] \Rightarrow \left[\mathbf{x} \equiv_H \mathbf{x}'\right],$$

- ***upstream-downstream-safe** (or, **UD-safe** in short) with respect to H if it is both U−safe and D−safe.*

Note that UD−safe ty is not monotone with respect to set inclusion. Also, if $H = A$ (i.e. all attributes are hidden) then m_i is clearly UD−safe with respect to to H. We call this the *trivial* UD−safe subset for m_i.

EXAMPLE 5. *Figure 3 shows some example module relations. For an (identity) module having relation R_1 in Figure 3a, the hidden subsets $\{a_1, a_3\}$ and $\{a_2, a_4\}$ are UD−safe. Note that $H = \{a_1, a_4\}$ is not a UD−safe subset: for tuples having the same values of visible attribute a_2, say 0, the values of a_3 are not the same. For a module having relation R_2 in Figure 3b, a UD−safe hidden subset is $\{a_2\}$, but there is no UD−safe subset that does not include a_2. It can also be checked that the module m_1 in Figure 1a does not have any non-trivial UD−safe subset.*

The first question we attempt to answer is whether there is a composability theorem analogous to Theorem 1 that works in the presence of public modules. In particular, we will show that for a class of workflows called *single-predecessor workflows* one can construct a private solution for the whole workflow by taking safe standalone solutions for the private modules, and then ensuring the UD−safe properties of the public modules in the corresponding public-closure. Next we define this class of workflows:

DEFINITION 9. *A workflow W is called a* **single-predecessor workflow***, if*

1. *W has no data-sharing, i.e. for $m_i \neq m_j$, $I_i \cap I_j = \emptyset$, and,*
2. *for every public module m_j that belongs to a public-closure with respect to some output attribute(s) of a private module m_i, m_i is the only private module that has a directed public path to m_j (i.e. m_i is the single private predecessor of m_j).*

EXAMPLE 6. *Again consider Figure 2b which shows a single-predecessor workflow. Modules m_3, m_4, m_6, m_7 have undirected public paths from $a_2 \in O_2$ (output attribute of m_2), whereas m_5 and m_8 have undirected (also directed) public paths from $a_4 \in O_2$; also m_1 is the single private-predecessor of $m_3, ..., m_8$ that has a directed path to each of module. The public module m_1 does not have any private predecessor, but m_1 does not belong to the public-closure with respect to the output attributes of any private module.*

Although single-predecessor workflows are more restrictive than general workflows, the above example illustrates that they can still capture fairly intricate workflow structures, and more importantly, they can capture commonly found chain and tree workflows [3]. Next in Section 4, we focus on single-predecessor workflows; then we explain in Section 5 how general workflows can be handled.

4. SINGLE-PREDECESSOR WORKFLOWS

The main motivation behind the study of single-predecessor workflows is to obtain a composability theorem similar to Theorem 1 combining solutions of standalone private and public modules. In Section 4.1, we show that such a composability theorem indeed exists for this class of workflows. Then we study how to optimally compose the standalone solutions in Section 4.2.

4.1 Composability Theorem for Privacy

The following composability theorem says that, for each private module m_i, it suffices to (i) find a safe hidden subset of output attributes (downstream propagation), (ii) find a superset of these hidden attributes such that each public module in their public closure is UD−safe, and (iii) no attributes outside the public closure and m_i are hidden (*i.e.* no unnecessary hiding). The union of these subsets of hidden attributes is workflow-private for each private module in the workflow. Theorem 2 stated below formalizes these three conditions.

THEOREM 2. *(***Composability Theorem for Single-predecessor Workflows***) Let W be a single-predecessor workflow. For each private module m_i in W, let H_i be a subset of hidden attributes such that (i) $h_i = H_i \cap O_i$ is safe for Γ-standalone-privacy of the module m_i, (ii) each public module m_j in the public-closure $C(h_i)$ is UD−safe with respect to $A_j \cap H_i$, and (iii) $H_i \subseteq O_i \cup \bigcup_{j:m_j \in C(h_i)} A_j$. Then the workflow W is Γ-private with respect to $H = \bigcup_{i:m_i \text{ is private}} H_i$.*

First, in Section 4.1.1, we argue why the conditions and assumptions in the above theorem are necessary; then we prove the theorem in Section 4.1.2.

4.1.1 Necessity of the Assumptions in Theorem 2

Theorem 2 has two non-trivial conditions: (1) the workflows are single-predecessor workflows, and (2) the public modules in the public closure must be UD−safe with respect to the hidden subset; the third condition that there is no unnecessary data hiding is required since the property UD−safety of public modules is not

valid with respect to set inclusion. The necessity of the first two conditions are discussed in Propositions 1 and 2 respectively.

In the proof of these propositions we will consider the different possible worlds of the workflow view and focus on the behavior (input-to-output mapping) $\widehat{m}_i$ of the module m_i as seen in these worlds. This may be different than its true behavior recorded in the actual workflow relation R, and we will say that m_i is *redefined* as $\widehat{m}_i$ in the given world. Note that m_i and $\widehat{m}_i$, viewed as relations, agree on the visible attributes of the the view but may differ in the non visible ones.

Necessity of Single-Predecessor Workflows. The next proposition shows that single-predecessor workflows constitute the largest class of workflows for which a composability theorem involving both public and private modules can succeed.

PROPOSITION 1. *There is a workflow W, which is not a single-predecessor workflow, (either because it has data sharing or because more (or fewer) than one such private-predecessor exists for some public module), and a private module m_i in W, where even hiding all output attributes of m_i and all attributes of all the public modules in W does not give Γ-privacy for* any $\Gamma > 1$.

PROOF. By Definition 9, a workflow W is *not* a single-predecessor workflow if one of the following holds: (i) there is a public module m_j in W that belongs to a public-closure of a private module m_i but has no directed path from m_i, (ii) such a public module m_j has a directed path from more than one private module, or (iii) W has data sharing. We now show an example for (i). Examples for the remaining conditions can be found in the full version [19].

Consider the workflow W_a in Figure 4a. Here the public module m_2 belongs to the public-closure $C(\{a_3\})$ of m_1, but there is no directed public path from m_1 to m_2, thereby violating the condition of single-predecessor workflows (though there is no data sharing). Module functionality is as follows: (i) m_1 takes a_1 as input and produces $a_3 = m_1(a_1) = a_1$. (ii) m_2 takes a_2 as input and produces $a_4 = m_2(a_2) = a_2$. (iii) m_3 takes a_3, a_4 as input and produces $a_5 = m_3(a_3, a_4) = a_3 \vee a_4$ (OR). (iv) m_4 takes a_5 as input and produces $a_6 = m_4(a_5) = a_5$. All attributes take values in $\{0, 1\}$.

Clearly, hiding output $\{a_3\}$ of m_1 gives 2-standalone privacy. We claim that hiding all output attributes of m_1 and all attributes of all public modules (*i.e.* $\{a_2, a_3, a_4, a_5\}$) gives only trivial 1-workflow-privacy for m_1, although it satisfies the `UD-safe` condition of m_2, m_3. To see this, consider the relation R_a of all executions of W_a given in Table 1, where the hidden values are in Grey. The rows (tuples) here are numbered $r_1, \ldots, r_4$ for later reference.

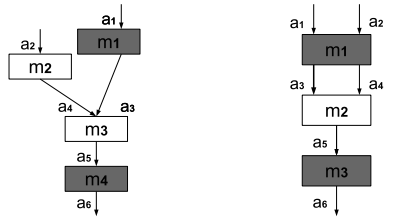
	a_1	a_2	a_3	a_4	a_5	a_6
r_1	0	0	0	0	0	0
r_2	0	1	0	1	1	1
r_3	1	0	1	0	1	1
r_4	1	1	1	1	1	1

Table 1: Relation R_a for workflow W_a given in Figure 4a

When a_3 is hidden, a possible candidate output of input $a_1 = 0$ to m_1 is 1. So we need to have a possible world where m_1 is redefined as $\widehat{m}_1(0) = 1$. This would restrict a_3 to 1 whenever $a_1 = 0$. But note that whenever $a_3 = 1$, $a_5 = 1$, irrespective of the value of a_4 (m_3 is an OR function).

This affects the rows r_1 and r_2 in R. Both these rows must have $a_5 = 1$, however r_1 has $a_6 = 0$, and r_2 has $a_6 = 1$. But this is impossible since, whatever the new definition $\widehat{m}_4$ of private module m_4 is, it cannot map a_5 to both 0 and 1; $\widehat{m}_4$ must be a function and maintain the functional dependency $a_5 \rightarrow a_6$. Hence all possible worlds of R_a must map $\widehat{m}_1(0)$ to 0, and therefore $\Gamma = 1$. □

Necessity of `UD-safe`ty for public modules. Example 3 in the previous section motivated why the downward-safety condition is necessary and natural. The following proposition illustrates the need for the additional upward-safety condition in Theorem 2, even when we consider downstream-propagation.



(a) Workflow W_a (b) Workflow W_b

Figure 4: Necessity of the conditions in Theorem 2: (a) Single-predecessor workflows, (b) `UD-safe`ty for public modules; White modules are public, grey are private.

PROPOSITION 2. *There is a workflow W with a private module m_i, and a safe subset of hidden attributes h_i guaranteeing Γ-standalone-privacy for m_i ($\Gamma > 1$), such that satisfying only the downstream-safety condition for the public modules in $C(h_i)$ does not give Γ-workflow-privacy for m_i for* any $\Gamma > 1$.

PROOF. Consider the chain workflow W_b given in Figure 4b with three modules m_1, m_2, m_3 defined as follows. (i) $(a_3, a_4) = m_1(a_1, a_2)$ where $a_3 = a_1$ and $a_4 = a_2$, (ii) $a_5 = m_2(a_3, a_4) = a_3 \vee a_4$ (OR), (iii) $a_6 = m_3(a_5) = a_5$. m_1, m_3 are private whereas m_2 is public. All attributes take values in $\{0, 1\}$. Clearly hiding output a_3 of m_1 gives Γ-standalone privacy for $\Gamma = 2$. Now suppose a_3 is hidden in the workflow. Since m_2 is public (known to be OR function), a_5 must be hidden (downstream-safety condition). Otherwise from visible output a_5 and input a_4, some values of hidden input a_3 can be uniquely determined (eg. if $a_5 = 0, a_4 = 0$, then $a_3 = 0$ and if $a_5 = 1, a_4 = 0$, then $a_3 = 1$). On attributes $(a_1, a_2, \underline{a_3}, a_4, \underline{a_5}, a_6)$, the original relation R is shown in Table 2 (the hidden attributes and their values are underlined in the text and in grey in the table).

a_1	a_2	a_3	a_4	a_5	a_6
0	0	0	0	0	0
0	1	0	1	1	1
1	0	1	0	1	1
1	1	1	1	1	1

Table 2: Relation R for workflow given in Figure 4b

Let us first consider an input $(0, 0)$ to m_1. When a_3 is hidden, a possible candidate output y of input tuple $x = (0, 0)$ to m_1 is $(\underline{1}, 0)$. So we need to have a possible world where m_1 is redefined as $\widehat{m}_1(0, 0) = (1, 0)$. To be consistent on the visible attributes, this forces us to redefine m_3 to $\widehat{m}_3$ where $\widehat{m}_3(1) = 0$; otherwise the row $(0, 0, \underline{0}, 0, \underline{0}, 0)$ in R changes to $(0, 0, \underline{1}, 0, \underline{1}, 1)$. This in turn forces us to define $\widehat{m}_1(1, 0) = (0, 0)$ and $\widehat{m}_3(0) = 1$. (This is because if we map $\widehat{m}_1(1, 0)$ to any of $\{(1, 0), (0, 1), (1, 1)\}$, either we have inconsistency on the visible attribute a_4, or $a_5 = 1$, and $\widehat{m}_3(1) = 0$, which gives a contradiction on the visible attribute $a_6 = 1$.)

Now consider the input $(1, 1)$ to m_1. For the sake of consistency on the visible attribute a_3, $\widehat{m}_1(1, 1)$ can take value $(1, 1)$ or $(0, 1)$.

But if $\widehat{m}_1(1,1) = (1,1)$ or $(0,1)$, we have an inconsistency on the visible attribute a_6. For this input in the original relation R, $a_5 = a_6 = 1$. Due to the redefinition of $\widehat{m}_3(1) = 0$, we have inconsistency on a_6. But note that the downstream-safety condition has been satisfied so far by hiding a_3 and a_5. To have consistency on the visible attribute a_6 in the row $(1,1,\underline{1},1,\underline{1},1)$, we must have $a_5 = 0$ (since $\widehat{m}_3(0) = 1$). The pre-image of $a_5 = 0$ is $a_3 = 0, a_4 = 0$, hence we have to redefine $\widehat{m}_1(1,1) = (\underline{0},0)$. But $(\underline{0},0)$ is not equivalent to original $m_1(1,1) = (\underline{1},1)$ with respect to the visible attribute a_4. So the only solution in this case for $\Gamma > 1$, assuming that we do not hide output a_6 of private module m_3, is to hide a_4, which makes the public module m_2 both upstream and downstream-safe. □

This example also suggests that upstream-safety is needed only when a private module gets input from a module in the public-closure. We will see later in the proof of Lemma 1 (Section 4.1.2) that this is indeed the case.

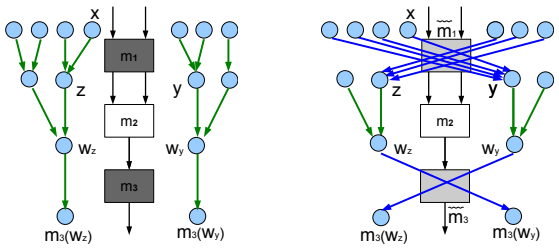
4.1.2 *Proof of Composability Theorem*

To prove Γ-privacy, we need to show the existence of at least Γ possible outputs for each input to each private module, originating from the possible worlds of the workflow relation with respect to the visible attributes. First we present a crucial lemma, which shows the existence of many possible outputs for any fixed input to any fixed private module in the workflow, when the conditions in Theorem 2 are satisfied. In particular, this lemma shows that any candidate output for a given input for standalone privacy remains a candidate output for workflow-privacy, even when the private module interacts with other private and public module in a (single-predecessor) workflow. Therefore, if there are $\geq \Gamma$ candidate outputs for standalone-privacy, there will be $\geq \Gamma$ candidate outputs for workflow-privacy. Later in this section we will formally prove Theorem 2 using this lemma.

LEMMA 1. *Consider a standalone private module m_i, a set of hidden attributes h_i, any input $\mathbf{x}$ to m_i, and any candidate output $\mathbf{y} \in \text{OUT}_{\mathbf{x},m_i,h_i}$ of $\mathbf{x}$. Then $\mathbf{y} \in \text{OUT}_{\mathbf{x},W,H_i}$ when m_i belongs to a single-predecessor workflow W, and a set attributes $H_i \subseteq A$ is hidden such that (i) $h_i \subseteq H_i$, (ii) only output attributes from O_i are included in h_i (i.e. $h_i \subseteq O_i$), and (iii) every module m_j in the public-closure $C(h_i)$ is* `UD-safe` *with respect to $A_j \cap H_i$.*

To prove the lemma, we will (arbitrarily) fix a private module m_i, an input $\mathbf{x}$ to m_i, a hidden subset h_i, and a candidate output $\mathbf{y} \in \text{OUT}_{\mathbf{x},m_i,h_i}$ for $\mathbf{x}$. The proof comprises two steps:

- **(Step-1)** Consider the connected subgraph $C(h_i)$ as a single *composite* public module M, or equivalently assume that $C(h_i)$ contains a single public module. By the properties of single-predecessor workflows, M gets all its inputs from m_i, but can send its outputs to one, multiple, or zero (for final output) private modules. Let I (respectively O) be the input (respectively output) attribute sets of M. In Figure 2b, the box is M, $I = \{a_2, a_3\}$ and $O = \{a_{10}, a_{11}, a_{12}, a_{13}\}$. We argue that when M is `UD-safe` with respect to visible attributes $(I \cup O) \cap H_i$, and the other conditions of Lemma 1 are satisfied, then $\mathbf{y} \in \text{OUT}_{\mathbf{x},W,H_i}$.
- **(Step-2)** We show that if every public module in the composite module $M = C(h_i)$ is `UD-safe`, then M is `UD-safe`. To continue with our example, in Figure 2b, assuming that m_3, m_4, m_6, m_7 are `UD-safe` with respect to the hidden attributes, we have to show that M is `UD-safe`.



(a) Original modules m_1, m_3 (b) Redefined $\widehat{m_1}, \widehat{m_3}$

Figure 5: Illustration of Example 7: Input-output relationship in (a) original workflow, (b) possible world mapping $\mathbf{x}$ to $\mathbf{y}$.

Proof of Step-1. The proof of Lemma 1 is involved even for the restricted scenario in Step-1, in which $C(h_i)$ contains a single public module; the proof can be found in the full version of the paper [19]. Here we illustrate here the key ideas using a simple example of a chain workflow.

EXAMPLE 7. *Consider a chain workflow, for instance, the one given in Figure 4b with the relation in Table 2. Fix module $m_i = m_1$. Hiding its output $h_1 = \{a_3\}$ gives Γ-standalone-privacy for $\Gamma = 2$. Fix input $\mathbf{x} = (0,0)$, with original output $\mathbf{z} = m_1(\mathbf{x}) = (\underline{0},0)$ (hidden attribute a_3 is underlined). Also fix a candidate output $\mathbf{y} = (\underline{1},0) \in \text{OUT}_{\mathbf{x},m_1,h_1}$. Note that $\mathbf{y}$ and $\mathbf{z}$ are equivalent on the visible attribute $\{a_4\}$.*

First, consider the simpler case when m_3 does not exist, i.e. *W contains only two modules m_1, m_2, and the column for a_6 does not exist in Table 2. As we mentioned before, when the composite public module does not have any private successor, we only need the downstream-safety property for modules in $C(h_i)$; in this case, $C(h_i)$ comprises a single public module, m_2. We construct a possible world R' of R by redefining module m_1 to $\widehat{m}_1$ as follows: $\widehat{m}_1$ simply maps all pre-images of $\mathbf{y}$ to $\mathbf{z}$, and all pre-images of $\mathbf{z}$ to $\mathbf{y}$. In this case, both $\mathbf{y}, \mathbf{z}$ have single pre-image. So $\mathbf{x} = (0,0)$ gets mapped to $(\underline{1},0)$ and input $(1,0)$ gets mapped to $(\underline{0},0)$. To make m_2 downstream-private, we hide output a_5 of m_2. Therefore, the set of hidden attributes $H_1 = \{a_3, a_5\}$. Finally R' is formed by the join of relations for $\widehat{m}_1$ and m_2. Note that the projection of R, R', will be the same on visible attributes a_1, a_2, a_4 (in R', the first row will be $(0,0,\underline{1},0,\underline{0})$ and the third row will be $(1,0,\underline{0},0,\underline{0})$).*

Next consider the more complicated case, when the modules in $C(h_i)$ have private successors (in this example, when the private module m_3 is present). We already argued in the proof of Proposition 2 that we also need to hide the input a_4 to ensure workflow privacy for $\Gamma > 1$ (`UD-safe`*ty). Let us now describe the proof strategy when a_4 is hidden,* i.e. *$H_1 = \{a_3, a_4, a_5\}$.*

Let $\mathbf{w_y} = m_2(\mathbf{y})$ and $\mathbf{w_z} = m_2(\mathbf{z})$ (see Figure 5a). We redefine m_1 to $\widehat{m}_1$ as follows (see Figure 5b). For all input $\mathbf{u}$ to m_1 such that $\mathbf{u} \in m_1^{-1}m_2^{-1}(\mathbf{w_z})$ (respectively $\mathbf{u} \in m_1^{-1}m_2^{-1}(\mathbf{w_y})$), we define $\widehat{m}_1(\mathbf{u}) = \mathbf{y}$ (respectively $\widehat{m}_1(\mathbf{u}) = \mathbf{z}$). Note that the mapping of tuples $\mathbf{u}$ that are not necessarily $m_1^{-1}(\mathbf{y})$ or $m_1^{-1}(\mathbf{z})$ are being redefined under m_1 (see Figure 5b). For $\widehat{m}_3$, we define, $\widehat{m}_3(\mathbf{w_y}) = m_3(\mathbf{w_z})$ and $\widehat{m}_3(\mathbf{w_z}) = m_3(\mathbf{w_y})$. Recall that $\mathbf{y} \equiv_{H_1} \mathbf{z}$ ($\mathbf{y}, \mathbf{z}$ have the same values of visible attributes). Since m_2 is downstream-safe $\mathbf{w_y} \equiv_{H_1} \mathbf{w_z}$. Since m_2 is also upstream-safe, for all input $\mathbf{u}$ to m_1 that are being redefined by $\widehat{m}_1$, their images under m_1 are equivalent with respect to H_1 (and therefore with $\mathbf{y}$ and $\mathbf{z}$). In our example, $\mathbf{w_y} = m_2(\underline{1},\underline{0}) = (\underline{1})$, and $\mathbf{w_z} = m_3(\underline{0},\underline{0}) = (\underline{0})$. $m_1^{-1}m_2^{-1}(\mathbf{w_z}) = \{(0,0)\}$ and $m_1^{-1}m_2^{-1}(\mathbf{w_y}) = \{(0,1), (1,0), (1,1)\}$. So $\widehat{m}_1$ maps $(0,0)$ to $(\underline{1},\underline{0})$ and all of $\{(0,1), (1,0), (1,1)\}$ to $(\underline{0},\underline{0})$; $\widehat{m}_3$ maps $(\underline{0})$ to (1) and $\underline{1}$ to (0).

Consider the relation R' formed by joining the relations of $\widehat{m}_1$,

a_1	a_2	a_3	a_4	a_5	a_6
0	0	1	0	1	0
0	1	0	0	0	1
1	0	0	0	0	1
1	1	0	0	0	1

Table 3: Relation R', a possible world of the relation R for the workflow in Figure 4b with respect to $H_1 = \{a_3, a_4, a_5\}$.

m_2, $\widehat{m}_3$ (see Table 3). The relation R' has the same projection on visible attributes $\{a_1, a_2, a_6\}$ as R in Table 2, and the public module m_2 is unchanged. So R' is a possible world of R that maps $\mathbf{x} = (0, 0)$ to $\mathbf{y} = (1, 0)$ as desired, i.e. $\mathbf{y} \in \text{OUT}_{\mathbf{x},W,H_1}$. □

The argument for more general single-predecessor workflows, like the one given in Figure 2b, is more complex. Here a private module (like m_{11}) can get inputs from m_i (in Figure 2b, m_2), from its public-closure $C(h_i)$ (in the figure, m_8), and also from the private successors of the modules in $C(h_i)$ (in the figure, m_{10}). In this case, the tuples $\mathbf{w_y}, \mathbf{w_z}$ are not well-defined, and redefining the private modules is more complex. In the proof of the lemma we give the formal argument using an *extended flipping function*, that selectively changes part of inputs and outputs of the private module based on their connection with the private module m_i.

Proof of Step-2. We formalize the claim in Step-2 below:

LEMMA 2. *Let M be a composite module consisting only of public modules. Let H be a subset of hidden attributes such that every public module m_j in M is* `UD-safe` *with respect to $A_j \cap H$. Then M is* `UD-safe` *with respect to $(I \cup O) \cap H$.*

PROOF SKETCH. The formal proof of this lemma can be found in the full version of the paper [19]. We sketch here the main ideas. To prove the lemma, we show that if every module in the public-closure is downstream-safe (respectively upstream-safe), then M is downstream-safe (respectively upstream-safe). For downstream-safety, we consider the modules in M in topological order, say $m_{i_1}, \cdots, m_{i_k}$ (in Figure 2b, $k = 4$ and the modules in order may be m_3, m_6, m_4, m_7). Let M^j be the (partial) composite public module formed by the union of modules $m_{i_1}, \cdots, m_{i_j}$, and let I^j, O^j be its input and output (the attributes that are either from a module not in M^j to a module in M^j, or to a module not in M^j from a module in M^j. Clearly, $M^1 = \{m_{i_1}\}$ and $M^k = M$. Then by induction from $j = 1$ to k, we show that M^j is downstream-safe with respect to $(I^j \cup O^j) \cap H$ if all of m_{i_ℓ}, $1 \leq \ell \leq j$ are downstream-safe with respect to $(I_{i_\ell} \cup O_{i_\ell}) \cap H = A_{i_\ell} \cap H$. For upstream-safety, we consider the modules in *reverse topological order*, $m_{i_k}, \cdots, m_{i_1}$, and give a similar argument. □

Proof of Theorem 2. Now we complete the proof of Theorem 2 using Lemma 1.

PROOF OF THEOREM 2. We first argue that if H_i satisfies the conditions in Theorem 2 then $H'_i = \bigcup_{\ell:m_\ell \text{ is private}} H_\ell$ satisfies the conditions in Lemma 1. Since $h_i = H_i \cap O_i$, (i) $h_i \subseteq H_i \subseteq \bigcup_{\ell:m_\ell \text{ is private}} H_\ell = H'_i$; and (ii) $h_i \subseteq O_i$. Next we argue that the third condition in the lemma also holds: (iii) every module m_j in the public-closure $C(h_i)$ is `UD-safe` with respect to $H'_i \cap A_j$.

To see (iii), observe that the Theorem 2 has an additional condition on H_i: $H_i \subseteq O_i \cup \bigcup_{j:m_j \in C(h_i)} A_j$. Since W is a single-predecessor workflow, for two private modules m_i, m_ℓ, the public closures $C(h_i) \cap C(h_\ell) = \emptyset$ (this follows directly from the definition of single-predecessor workflows). Further, since W is single-predecessor, W has no data-sharing by definition. So for any two modules m_j, m_ℓ in W (public or private), the set of attributes $A_j \cap A_\ell = \emptyset$. Clearly, when m_i is a private module, $m_i \notin C(h_\ell)$ for any private module m_ℓ in W, by the definition of public-closure. Hence for any two private modules m_i, m_ℓ, $\left(O_i \cup \bigcup_{j:m_j \in C(h_i)} A_j\right) \cap \left(O_\ell \cup \bigcup_{j:m_j \in C(h_\ell)} A_j\right) = \emptyset$. In particular, for two private modules $m_i \neq m_\ell$, $H_i \cap H_\ell = \emptyset$. Hence, for a public module $m_j \in C(h_i)$, and for any other private module m_ℓ, $A_j \cap H_\ell = \emptyset$. Therefore, $A_j \cap H'_i = A_j \cap (\bigcup_{\ell:m_\ell \text{ is private}} H_\ell)$ $= A_j \setminus H_i$. Since m_j is `UD-safe` with respect to $A_j \cap H_i$ from the condition in the theorem, m_j is also `UD-safe` with respect to $A_j \cap H'_i$. Hence H'_i satisfies the conditions in the lemma.

Theorem 2 assumes that each private module m_i is Γ-standalone-private with respect to h_i, i.e., $|\text{OUT}_{\mathbf{x},m_i,h_i}| \geq \Gamma$ for all input $\mathbf{x}$ to m_i (see Definition 2). From Lemma 1, using H'_i in place of H_i, this implies that for all input $\mathbf{x}$ to private modules m_i, $|\text{OUT}_{\mathbf{x},W,H'_i}| \geq \Gamma$ where $H'_i = \bigcup_{\ell:m_\ell \text{ is private}} H_\ell$. From Definition 4, this implies that each private module m_i is Γ-workflow-private in H'_i which is the same as H in Theorem 2. Since this holds for all private modules m_i, W is Γ-private with respect to H. □

4.2 Optimal Composition for Single Predecessor Workflows

Recall the *optimal composition problem* mentioned in Section 2.3. This problem focused on optimally combining the safe solutions for private modules in an all-private workflow in order to minimize the cost of hidden attributes. In this section, we consider optimal composition for a single-predecessor workflow W with private and public modules. Our goal is to find subsets H_i for each private module m_i in W satisfying the conditions given in Theorem 2 such that $\texttt{cost}(H)$ is minimized for $H = \bigcup_{i:m_i \text{ is private}} H_i$. This we solve in four steps: (I) find the safe solutions for standalone-privacy for individual private modules; (II) find the `UD-safe` solutions for individual public modules; (III) find the optimal hidden subset H_i for the public-closure of every private module m_i using the outputs of the first two steps; and (IV) combine H_i-s to find the final optimal solution H. We next consider each of these steps.

I. Private Solutions for Individual Private Modules. For each private module m_i we compute the set of safe subsets $\mathbf{S}_i = \{S_{i1}, \cdots, S_{ip_i}\}$, where each $S_{i\ell} \subseteq O_i$ is standalone-private for m_i. Here p_i is the number of safe subsets for m_i. Recall from Theorem 2 that the choice of safe subset for m_i determines its public-closure (and consequently the possible H_i sets and the cost of the overall solution). It is thus not sufficient to consider only the safe subsets that have the minimum cost; we need to keep *all* safe subsets for m_i, to be examined by subsequent steps.

The complexity of finding safe subsets for individual private modules has been thoroughly studied in [17] under the name *standalone* `Secure-View` *problem*. It was shown that deciding whether a given hidden subset of attributes is safe for a private module is NP-hard in the number of attributes of the module. It was further shown that the set of *all* safe subsets for the module can be computed in time exponential in the number of attributes assuming constant domain size, which almost matches the lower bounds.

Although the lower and upper bounds are somewhat disappointing, as argued in [17], the number of attributes of an individual module is fairly small. Further, this computation is done only once as a pre-processing step and the cost can be amortized over possibly many uses of the module in different workflows. The integers and reals are represented using a fixed number of bits, and the domain size for these inputs/outputs can still be assumed to be a constant. However, in these cases the individual relations can be big. For practical purposes, the module designers should be able to provide some insight, from their semantic knowledge of what the module

does, without actually enumerating all possibilities.

II. Safe Solutions for Individual Public Modules. This step focuses on finding the set of all UD-safe solutions for the individual public modules. We denote the UD-safe solutions for a public module m_j by $\mathbf{U}_j = \{U_{j1}, \cdots, U_{jp_j}\}$, where each UD-safe subset $U_{j\ell} \subseteq A_j$; p_j denotes the number of UD-safe solutions for the public module m_j. We will see below in Theorem 3 that even deciding whether a given subset is UD-safe for a module is coNP-hard in the number of attributes (and that the set of all such subsets can be computed in exponential time). However, as argued in the first step, this computation can be done once as a pre-processing step with its cost amortized over possibly many workflows where the module is used. In addition, it suffices to compute the UD-safe subsets for only those public modules that belong to some public-closure for some private module.

THEOREM 3. *Given public module m_j with k attributes, and a subset of hidden attributes H, deciding whether m_j is UD-safe with respect to H is coNP-hard in k. Further, all UD-safe subsets can be found in EXP-time in k.*

PROOF SKETCH OF CONP-HARDNESS. The reduction is from the UNSAT problem, where given n variables $x_1, \cdots, x_n$, and a 3CNF formula $f(x_1, \cdots, x_n)$, the goal is to check whether f is *not* satisfiable. In our construction, m_i has $n+1$ inputs $x_1, \cdots, x_n$ and y, and the output is $z = m_i(x_1, \cdots, x_n, y) = f(x_1, \cdots, x_n) \vee y$ (OR). The set of hidden attributes is $x_1, \cdots, x_n$ (*i.e.* y, z are visible). We claim that f is not satisfiable if and only if m_i is UD-safe with respect to H. □

The above construction, with attributes y and z assigned cost zero and all other attributes assigned some higher constant cost, can be used to show that testing whether a safe subset with cost smaller than a given threshold exists is also coNP-hard.

Regarding the upper bound, the trivial algorithm of going over all 2^k subsets h of A_j, and checking if h is UD-safe for m_j, can be done in EXP-time in k when the domain size is constant. Since the UD-safe property is *not monotone* with respect to further deletion of attributes, if h is UD-safe, its supersets may not be UD-safe. Recall however that the trivial solution $h = A_j$ (deleting all attributes) is always UD-safe for m_j. So for practical purposes, when the public-closure for a private module involves a small number of attributes of the public modules in the closure, or if the attributes of those public modules have small cost, this solution can be used. The complete proof of the theorem can be found in the full version of the paper [19].

III. Optimal H_i for Each Private Module. The third step aims to find a set H_i of hidden attributes, of minimum cost, for every private module m_i. As per the theorem statement, this set H_i should satisfy the conditions: (a) $H_i \cap O_i = S_{i\ell}$, for some safe subset $S_{i\ell} \in \mathbf{S}_i$; (b) for every public module m_j in the closure $C(S_{i\ell})$, there exists a UD-safe subset $U_{jq} \in \mathbf{U}_j$ such that $U_{jq} = A_j \cap H_i$; and (c) H_i does not include any attribute outside O_i and $C(S_{i\ell})$. We show that, for the important class of chain and tree workflows, this optimization problem is solvable in time polynomial in the number of modules n, the total number of attributes in the workflow $|A|$, and the maximum number of sets in $\mathbf{S}_i$ and $\mathbf{U}_j$ (denoted by L):

THEOREM 4. *For each private module m_i in a tree workflow (and therefore, in a chain workflow), the optimal subset H_i can be found in polynomial time in n, $|A|$ and L.*

On the other hand, the problem is NP-hard when the workflow has arbitrary DAG structure even when both the number of attributes and the number of safe and UD-safe subsets of the individual modules are bounded by a small constant.

In contrast, the problem becomes NP-hard in n when the public-closure forms an arbitrary directed acyclic subgraph, even when L is a constant and the number of attributes of the individual modules is bounded by a small constant.

Chain workflows are the simplest class of tree-shaped workflow, hence clearly any algorithm for trees will also work for chains. However, for the sake of simplicity, we give the optimal algorithm for chain workflows first; then we discuss how it can be proved for tree workflows.

Optimal algorithm for chain workflows. Consider any private module m_i. Given a safe subset $S_{i\ell} \in \mathbf{S}_i$, we show below how an optimal subset H_i in $C(S_{i\ell})$ satisfying the desired properties can be obtained. We then repeat this process for all safe subsets (bounded by L) $S_{i\ell} \in \mathbf{S}_i$, and output the subset H_i with minimum cost. We drop the subscripts to simplify the notation (*i.e.* use S for $S_{i\ell}$, C for $C(S_{i\ell})$, and H for H_i).

Our poly-time algorithm employs dynamic programming to find the optimal H. First note that since C is the public-closure of output attributes for a chain workflow, C should be a chain itself. Let the modules in C be renumbered as $m_1, \cdots, m_k$ in order. Now we solve the problem by dynamic programming as follows. Let Q be an $k \times L$ two-dimensional array, where $Q[j, \ell]$ denotes the cost of minimum cost hidden subset $H^{j\ell}$ that satisfies the UD-safe condition for all public modules m_1 to m_j and $A_j \cap H^{j\ell} = U_{j\ell} \in \mathbf{U}_j$. Here $j \leq k$, $\ell \leq p_j \leq L$, and A_j is the attribute set of m_j; the actual solution can be stored easily by standard argument.

The initialization step is, for $1 \leq \ell \leq p_1$,

$$Q[1, \ell] = c(U_{1,\ell}) \quad \text{if } U_{1,\ell} \supseteq S, \qquad = \infty \quad \text{otherwise}$$

Recall that for a chain, $O_{j-1} = I_j$, for $j = 2$ to k. Then for $j = 2$ to k, $\ell = 1$ to p_j, $Q[j, \ell]$

$$= \quad \infty \quad \text{if } \nexists 1 \leq q \leq p_{j-1} \text{ s.t. } U_{j-1,q} \cap O_{j-1} = U_{j,\ell} \cap I_j$$
$$= \quad c(O_j \cap U_{j\ell}) + \min_q Q[j-1, q] \qquad \text{otherwise}$$

It is interesting to note that such a q always exists for at least one $\ell \leq p_j$: while defining UD-safe subsets in Definition 8, we discussed that any public module m_j is UD-safe when its entire attribute set A_j is hidden. Hence $A_{j-1} \in \mathbf{U}_{j-1}$ and $A_j \in \mathbf{U}_j$, which will make the equality check true (for a chain $O_{j-1} = I_j$). It can be shown that (see the full version [19]) $Q[j, \ell]$ correctly stores the desired value. Then the optimal solution H has cost $\min_{1 \leq \ell \leq p_k} Q[k, \ell]$; the corresponding solution H can be found by standard procedure, which proves Theorem 4 for chain workflows.

Observe that, more generally, the algorithm may also be used for non-chain workflows, if the public-closures of the safe subsets for private modules have chain shape. This observation also applies to the following discussion on tree workflows.

Optimal algorithm for tree workflows. Now consider tree-shaped workflows, where every module in the workflow has at most one immediate predecessor (for all modules m_i, if $I_i \cap O_j \neq \emptyset$ and $I_i \cap O_k \neq \emptyset$, then $j = k$), but a module can have one or more immediate successors.

The treatment of tree-shaped workflows is similar to what we have seen for chains. Observe that, here again, since C is the public-closure of output attributes for a tree-shaped workflow, C will be a collection of trees all rooted at m_i. As for the case of

chains, the processing of the public closure is based on dynamic-programming. The key difference is that the modules in the tree are processed bottom up (rather than top down as in what we have seen above) to handle branching. The proof of Theorem 4 for tree workflows can be found in the full version [19].

NP-hardness for public-closure of arbitrary shape. Finding the minimal-cost solution for public-closure with arbitrary DAG shape is NP-hard. The NP-hardness of this problem follows by a reduction from 3SAT (see the full version [19]). The NP algorithm simply guesses a set of attributes and checks whether it forms a legal solution and has cost lower than the given bound; the optimal solution can be found in EXP-time by iterating over all subsets.

The NP-hardness here is in the number of modules in the public closure. Hence whenever the number of public modules in the public closure is small, our solution is better than the the naive one, which is exponential in the size of the *full* workflow.

IV. Optimal Hidden Subset H for the Workflow. According to Theorem 2, $H = \bigcup_{i:m_i \text{ is private}} H_i$ is a Γ-private solution for the workflow. Observe that finding the optimal (minimum cost) such solution H for single-predecessor workflows is straightforward, once the minimum cost H_i-s are found: Due to the condition in Theorem 2 that no unnecessary data are hidden, it can be easily checked that for any two private modules m_i, m_k in a single predecessor workflow, $H_i \cap H_k = \emptyset$. Hence the optimal solution H can be obtained by taking the union of the optimal hidden subsets H_i for individual private modules obtained in the previous step.

5. GENERAL WORKFLOWS

The previous sections focused on single-predecessor workflows. In particular, we presented a privacy theorem for such workflows and studied optimization with respect to this theorem. The following two observations highlight how this privacy theorem can be extended to general workflows. For lack of space the discussion is informal; the proof techniques are similar to single-predecessor workflows and are given in the full version of the paper [19].

Observation 1: Need for propagation through private modules. All examples in the previous sections that showed the necessity of the single-predecessor assumption for private module m_i had another private module m_k as which is a successor of one public module in the public closure of m_i. For instance, in the proof of Proposition 1 (see Figure 4a) $m_i = m_1$ and $m_k = m_4$. If we had continued hiding output attributes of m_4, we could obtain the required possible worlds leading to a non-trivial privacy guarantee $\Gamma > 1$. This implies that for general workflows, the propagation of attribute hiding should continue outside the public closure and through the descendant private modules.

Observation 2: `D-safety` suffices (instead of `UD-safety`). The proof of Lemma 1 shows that the `UD-safety` property of modules in the public-closure is needed only when some public module in the public-closure has a private successor whose output attributes are visible. If all modules in the public closure have no such private successor, then a downstream-safety property (called the **`D-safety` property**) is sufficient. More generally, if attribute hiding is propagated through private modules (as discussed above), then it suffices to require the hidden attributes to satisfy `D-safety` rather than the stronger `UD-safety` property.

The intuition from the above two observations is formalized in a *privacy theorem for general workflows*, analogous to Theorem 2. First, instead of public-closure, it uses *downward-closure*: for a private module m_i, and a set of hidden attributes h_i, the downward-closure $D(h_i)$ consists of all modules (public or private) m_j, that are reachable from m_i by a directed path. Second, instead of requiring the sets H_i of hidden attributes to ensure `UD-safety` , it requires them to only ensure `D-safety`.

The proof of the revised theorem is similar to that of Theorem 2, with the added complication that the H_i subsets are no longer disjoint. This is resolved by proving that `D-safe` subsets are closed under union, allowing for the (possibly overlapping) H_i subsets computed for the individual private modules to be unioned.

The hardness results from the previous section transfer to the case of general workflows. Since the H_i-s in this case may be overlapping, the union of optimal H_i solutions for individual modules m_i may not be optimal for the workflow. Whether or not there exists a non-trivial approximation is an interesting open problem.

To conclude the discussion, note that for single-predecessor workflows, we now have two options to ensure workflow-privacy: (i) to consider public-closures and ensure `UD-safety` properties for their modules (following the privacy theorem for single-predecessor workflows); or (ii) to consider downward-closures and ensure the `D-safety` property for their modules (following the privacy theorem for general workflows). Observe that these two options are incomparable: Satisfying `UD-safety` properties may require hiding more attributes than what is needed for satisfying `D-safety` properties. On the other hand, the downward-closure includes more modules than the public-closure (for instance the reachable private modules), and additional attributes must be hidden to satisfy their `D-safety` properties. One could therefore run both algorithms, and choose the lower cost solution.

6. RELATED WORK

Privacy concerns with respect to provenance were articulated in [18], in the context of scientific workflows, and in [20], in the context of business processes. Preserving module privacy in all-private workflows was studied in [17] and the idea of privatizing (hiding the "name" of) public modules to achieve privacy in public/private workflows was proposed. Unfortunately this is not realistic for many common scenarios. This paper thus presents a novel *propagation model* for attribute hiding which does not place any assumptions on the user's prior knowledge about public modules.

Recent work by other authors includes the development of fine-grained access control languages for provenance [33, 35, 7, 8], and a graph grammar approach for rewriting redaction policies over provenance [9]. The approach in [6] provides users with informative graph query results using *surrogates*, which give less sensitive versions of nodes/edges, and proposes a utility measure for the result. A framework to output a *partial* view of a workflow that conforms to a given set of access permissions on the connections and input/output ports was proposed in [10]. Although related to module privacy, the approach may disconnect connections between modules rather than just hiding the data which flows between them. More importantly, the notion of privacy is informal and no guarantees on the quality of the solutions are provided. Also related to our work are the recent papers on provenance security [13, 12]; a general and formal model for provenance and its security properties like *obfuscation* and *disclosure* are proposed in [12].

A related area is that of *privacy-preserving data mining* (see surveys [4, 36], and the references therein). Here, the goal is to hide individual data attributes while retaining the suitability of the data for mining patterns. Privacy preserving approaches have been studied for *social networks* (e.g. [5]), *auditing queries* (e.g. [32]), *network routing* [27], and several other contexts.

Our notion of module privacy is closest to the notion of ℓ-diversity [30] which addresses some shortcomings of κ-anonymity [34]. The notion of ℓ-diversity tries to generalize the values of the *non-sensitive attributes* so that for every such generalization, there are at least ℓ

different values of *sensitive attributes*. The view-based approach for k-anonymity along with its complexity has been studied in [40]. Leakage of information due to knowledge on the techniques for minimizing data loss has been studied in [37, 25, 16, 38]; however, our privacy guarantees are information theoretic under our assumptions.

Nevertheless, the privacy notion of ℓ-diversity is susceptible to attack when the user has background knowledge [26, 28]. *Differential privacy* [23, 21, 22], which requires that the output distribution is *almost* invariant to the inclusion of any particular record, gives a stronger privacy guarantee. Although it was first proposed for *statistical databases* and *aggregate queries*, it has since been studied in domains such as mechanism design [31], data streaming [24], and several database-related applications (e.g. [29, 39, 15, 11]). However, it is well-known that no *deterministic* algorithm can guarantee differential privacy, and the standard approach of including random noise is not suitable for our purposes — provenance queries are typically not aggregate queries, and we need the output views to be consistent (e.g. the same module must map the same input to the same output in all executions of the workflow). Defining an appropriate notion of differential privacy for module functionality with respect to provenance queries is an interesting open problem. It would also be interesting to study natural attacks for our application, and (theoretically or empirically) study the effectiveness of various notions of privacy under these attacks [14].

7. CONCLUSIONS

In this paper, we addressed the problem of preserving module privacy in public/private workflows (called workflow-privacy), by providing a view of provenance information in which the input to output mapping of private modules remains hidden. As several examples in this paper show, the workflow-privacy of a module critically depends on the structure (connection patterns) of the workflow, the behavior/functionality of other modules in the workflow, and the selection of hidden attributes. We showed how workflow-privacy can be achieved by propagating data hiding through public modules in both single-predecessor and general workflows.

Several interesting future research directions related to the application of differential privacy were discussed in Section 6. We assumed certain assumptions in the paper (constant domain size, acyclic nature of workflows, analysis using relations of executions, etc.). Even with these assumptions, the problem is highly non-trivial and large and important classes of workflows can be captured even under these assumption. However, it would be immensely important to have models and solutions that can be used in scientific experiments in practice. We have also mentioned the shortcomings of the Γ-privacy and the difficulty in using stronger privacy notions like differential privacy. It will be interesting to see if the possible world model thoroughly studied in this paper can be used to facilitate the use of other privacy models under provenance queries.

Acknowledgements. We thank the anonymous reviewers for their insightful comments. This work was supported in part by the NSF Awards IIS-0803524 and CCF-1116961, the European Research Council under the European Union's Seventh Framework Programme (FP7/2007-2013) / ERC grant agreement 291071-MoDaS, the Israel Ministry of Science, the Binational (US-Israel) Science Foundation, and a Google Ph.D. Fellowship.

8. REFERENCES

[1] *https://www.dna20.com*,.
[2] *http://www.smartgene.com*,.
[3] *www.myexperiment.org*.
[4] C. C. Aggarwal and P. S. Yu. Privacy-preserving data mining: Models and algorithms. 2008.
[5] L. Backstrom, C. Dwork, and J. M. Kleinberg. Wherefore art thou r3579x?: Anonymized social networks, hidden patterns, and structural steganography. In *WWW*, 2007.
[6] B. T. Blaustein, A. Chapman, L. Seligman, M. D. Allen, and A. Rosenthal. Surrogate parenthood: Protected and informative graphs. *PVLDB*, 4(8):518–527, 2011.
[7] T. Cadenhead, M. Kantarcioglu, and B. M. Thuraisingham. A framework for policies over provenance. In *TaPP*, 2011.
[8] T. Cadenhead, V. Khadilkar, M. Kantarcioglu, and B. M. Thuraisingham. A language for provenance access control. In *CODASPY*, pages 133–144, 2011.
[9] T. Cadenhead, V. Khadilkar, M. Kantarcioglu, and B. M. Thuraisingham. Transforming provenance using redaction. In *SACMAT*, pages 93–102, 2011.
[10] A. Chebotko, S. Chang, S. Lu, F. Fotouhi, and P. Yang. Scientific workflow provenance querying with security views. *WAIM*, pages 349–356, July 2008.
[11] R. Chen, N. Mohammed, B. C. M. Fung, B. C. Desai, and L. Xiong. Publishing set-valued data via differential privacy. *PVLDB*, 4(11):1087–1098, 2011.
[12] J. Cheney. A formal framework for provenance security. *2012 IEEE 25th Computer Security Foundations Symposium*, 0:281–293, 2011.
[13] S. Chong. Towards semantics for provenance security. In *First workshop on on Theory and practice of provenance*, TAPP'09, pages 2:1–2:5, 2009.
[14] G. Cormode. Personal privacy vs population privacy: learning to attack anonymization. In *KDD*, pages 1253–1261, 2011.
[15] G. Cormode, C. M. Procopiuc, D. Srivastava, and T. T. L. Tran. Differentially private publication of sparse data. *CoRR*, abs/1103.0825, 2011.
[16] G. Cormode, D. Srivastava, N. Li, and T. Li. Minimizing minimality and maximizing utility: analyzing method-based attacks on anonymized data. *Proc. VLDB Endow.*, 3(1-2):1045–1056, Sept. 2010.
[17] S. B. Davidson, S. Khanna, T. Milo, D. Panigrahi, and S. Roy. Provenance views for module privacy. In *PODS*, pages 175–186, 2011.
[18] S. B. Davidson, S. Khanna, S. Roy, J. Stoyanovich, V. Tannen, and Y. Chen. On provenance and privacy. In *ICDT*, pages 3–10, 2011.
[19] S. B. Davidson, T. Milo, and S. Roy. A propagation model for provenance views of public/private workflows. *CoRR*, abs/1212.2251, 2012.
[20] D. Deutch and T. Milo. A quest for beauty and wealth (or, business processes for database researchers). In *PODS*, pages 1–12, 2011.
[21] C. Dwork. Differential privacy: A survey of results. In *TAMC'08*, pages 1–19.
[22] C. Dwork. The differential privacy frontier (extended abstract). In *TCC*, pages 496–502, 2009.
[23] C. Dwork, F. Mcsherry, K. Nissim, and A. Smith. Calibrating noise to sensitivity in private data analysis. In *TCC*, pages 265–284. Springer, 2006.
[24] C. Dwork, M. Naor, T. Pitassi, G. N. Rothblum, and S. Yekhanin. Pan-private streaming algorithms. In *ICS*, pages 66–80, 2010.
[25] C. Fang and E.-C. Chang. Information hiding. chapter Information Leakage in Optimal Anonymized and Diversified Data, pages 30–44. 2008.
[26] S. R. Ganta, S. P. Kasiviswanathan, and A. Smith. Composition attacks and auxiliary information in data privacy. In *KDD*, pages 265–273, 2008.
[27] A. J. T. Gurney, A. Haeberlen, W. Zhou, M. Sherr, and B. T. Loo. Having your cake and eating it too: Routing security with privacy protections. In *HotNets'11*.
[28] D. Kifer. Attacks on privacy and definetti's theorem. In *SIGMOD*, pages 127–138, 2009.
[29] D. Kifer and A. Machanavajjhala. No free lunch in data privacy. In *SIGMOD*, pages 193–204, 2011.
[30] A. Machanavajjhala, J. Gehrke, D. Kifer, and M. Venkitasubramaniam. l-Diversity: Privacy Beyond k-Anonymity. In *ICDE*, 2006.
[31] F. McSherry and K. Talwar. Mechanism design via differential privacy. In *FOCS*, pages 94–103, Washington, DC, USA, 2007. IEEE Computer Society.
[32] S. U. Nabar, B. Marthi, K. Kenthapadi, N. Mishra, and R. Motwani. Towards robustness in query auditing. In *VLDB*, pages 151–162, 2006.
[33] Q. Ni, S. Xu, E. Bertino, R. S. Sandhu, and W. Han. An access control language for a general provenance model. In *Secure Data Management'09*, pages 68–88.
[34] L. Sweeney. k-anonymity: a model for protecting privacy. *Int. J. Uncertain. Fuzziness Knowl.-Based Syst.*, 10(5):557–570, 2002.
[35] V. Tan, P. T. Groth, S. Miles, S. Jiang, S. Munroe, S. Tsasakou, and L. Moreau. Security issues in a SOA-based provenance system. In *IPAW'06*, pages 203–211.
[36] 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.
[37] R. C.-W. Wong, A. W.-C. Fu, K. Wang, and J. Pei. Minimality attack in privacy preserving data publishing. In *VLDB*, pages 543–554, 2007.
[38] X. Xiao, Y. Tao, and N. Koudas. Transparent anonymization: Thwarting adversaries who know the algorithm. *ACM Trans. Database Syst.*, 35(2):8:1–8:48, May 2010.
[39] X. Xiao, G. Wang, and J. Gehrke. Differential privacy via wavelet transforms. In *ICDE*, pages 225–236, 2010.
[40] C. Yao, X. S. Wang, and S. Jajodia. Checking for k-anonymity violation by views. In *VLDB*, pages 910–921, 2005.