A Propagation Model for Provenance Views of Public/Private Workflows

Susan B. Davidson∗ Tova Milo† Sudeepa Roy‡

May 5, 2014

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 $\Gamma$-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 $\Gamma$-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 $\Gamma$-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.

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 [16], 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 [30, 32, 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 [15], the approach we use is to extend the notion of ℓ-diversity [27] 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 Γ − 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 m2, 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 m2 is embedded in a simple chain workflow m1 → m2 → m3, where both m1 and m3 are public, equality modules. Then even if we hide both the input and output of m2, their values can be retrieved from the input to m1 and the output from m3. Note that the same problem would arise if m1 and m3 were invertible functions, *e.g.* reformatting modules, a common case in practice.

In [15], 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 [15] 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 m1 was an equality module then hiding the input to m2 (output of m1) would be sufficient. Similarly, if m3 was privatized then hiding the output of m2 (input to m3) 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 [15] can be applied. However, merging may be difficulty for workflows with complex network structure, large amount 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. For instance, 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 m2 were hidden then the input to m1 would also be hidden, although the user would still know that m1 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 in the general case: 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. Using a downward propagation model, we show the following strong results: For a special class of common workflows, single (private)-predecessor workflows, or simply single-predecessor workflows (which include the common tree and chain workflows), taking solutions for $\Gamma$-standalone-privacy of each private module (safe subsets) augmented with specially chosen input/output data of public modules in their public closure (up to a successor private module) that is rendered upstream-downstream safe ($UD$-safe) by the data hiding, and hiding the union of data in the augmented solutions for each private module will ensure $\Gamma$-workflow privacy for all private modules. We define these notions formally in Section 3 and go on to 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 many different safe subsets for private modules and $UD$-safe subsets for public modules, the next problem we address is finding a minimum cost solution — the optimum view problem. Using the result from above, we show that for single-predecessor workflows the optimum view problem may be solved by first identifying safe and $UD$-safe subsets for the private and public modules, respectively, then assembling them together optimally. The complexity of identifying safe subsets for a private module was studied in [15] and the problem was shown to be NP-hard (EXP-time) in the number of module attributes. We show here 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 input/output data. 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 pre-processing step with the cost being amortized over possibly many uses of the module in different workflows. In particular we show that, given the computed subsets, for chain and tree-shaped workflows, the optimum view problem has a polynomial time solution in the size of the workflow and the maximum number of safe/$UD$-safe subsets for a private/public modules. Furthermore, the algorithm can 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 (EXP-time) in the size of the public closure.

We then consider general acyclic workflows, and give a sufficient condition to ensure $\Gamma$-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 upward-downstream-safety ($UD$-safe).

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. Section 4.1 discusses the proof of the privacy theorem, and the necessity of the upstream-downstream-safety condition as well as the single-predecessor restriction. The optimization problem is studied in Section 4.2. We then discuss general public/private workflows in Section 4.1 before giving related work in Section 6 and
concluding in Section 7.

2 Preliminaries

We start by reviewing the formal definitions and notions of module privacy from [15], and then extend them to the context studied in this paper. Readers familiar with the definitions and results in [15] 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 function 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.

We assume that the values of each attribute $a \in A$ come from a finite but arbitrarily large domain $\Delta_a$, and let $\text{Dom} = \prod_{a \in I} \Delta_a$ and $\text{CoDom} = \prod_{a \in O} \Delta_a$ denote the domain and co-domain of the module $m$ respectively.

The relation $R$ thus represents the (possibly partial) function $m : \text{Dom} \rightarrow \text{CoDom}$ and tuples in $R$ describe executions of $m$, namely for every $t \in R$, $\Pi_O(t) = m(\Pi_I(t))$. We overload the standard notation for projection, $\Pi_A(R)$, and use it for a tuple $t \in R$. Thus $\Pi_A(t)$, for a set $A$ of attributes, denotes the projection of $t$ to the attributes in $A$.

Workflows A workflow $W$ consists of a set of modules $m_1, \ldots, m_n$, connected as a DAG (see, for instance, 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 $A = \bigcup^n_{i=1} A_i$, 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 $i \in [1, n]$, $\Pi_{O_i}(t) = m_i(\Pi_{I_i}(t))$. 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 $a_3 = a_1 \lor a_2$, $a_4 = \neg(a_1 \land a_2)$ 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 $a_7 = a_4 \land a_6$. The relational representation (functionality) $R_1$ of module $m_1$ with the functional dependency $a_1a_2 \rightarrow a_3a_4a_5$ 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 $a_1a_2 \rightarrow a_3a_4a_5$, $a_3a_4 \rightarrow a_6$, $a_4a_5 \rightarrow a_7$ 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 $m_3$.

---

1The example in this section is also taken from [15].

2We distinguish between the possible range $O$ of the function $m$ that we call co-domain and the actual range $\{y : \exists x \in I \text{ s.t. } y = m(x)\}$.
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 [15]) 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. 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 $\text{worlds}(R,H)$, 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 uncertainty with respect to the value of the output $m(\Pi_I(t))$, for tuples $t \in R$. To define this, we introduce the notion of $\Gamma$-standalone-privacy, for a given parameter $\Gamma \geq 1$. Informally, a view $R'$ is $\Gamma$-standalone-private if for every $t \in R$, $\text{worlds}(R,H)$ contains at least $\Gamma$ distinct output values that could be the result of $m(\Pi_I(t))$.

**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 $\Gamma$-standalone-private with respect to a set of hidden attributes $H$, if for every tuple $x \in \Pi_I(R)$, $|\text{OUT}_{x,m,H}| \geq \Gamma$, where $\text{OUT}_{x,m,H} = \{y \mid \exists R' \in \text{worlds}(R,H), \exists t' \in R' \text{ s.t. } x = \Pi_I(t') \land y = \Pi_O(t')\}$\(^3\)

---

\(^3\)In [15], we (equivalently) defined privacy with respect to visible attributes $V$ instead of hidden attributes $H$, and we used the notation “$\text{OUT}_{x,m}$ with respect to $V$” instead of $\text{OUT}_{x,m,H}$.
If $m$ is $\Gamma$-standalone-private with respect to hidden attributes $H$, then we call $H$ a safe subset for $m$ and $\Gamma$.

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, $\Gamma$-standalone-privacy implies that for any input the adversary cannot guess $m$’s output with probability $> \frac{1}{2}$, 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[4]. 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, $R \in \text{Worlds}(R_1, H)$, and we can check that overall there are 64 relations in $\text{Worlds}(R_1, H)$.

Furthermore, it can be verified that, if $H = \{a_2, a_4\}$, then for all $x \in \Pi_i(R_i)$, $|\text{OUT}_{x,m,H}| \geq 4$, so $\{a_1, a_3, a_5\}$ is safe for $m_1$ and $\Gamma = 4$. As an example, when $x = (0,0)$, $\text{OUT}_{x,m,H} \supseteq \{(0,0,1), (0,1,1), (1,0,0), (1,1,0)\}$ (hidden attributes are underlined) – we can define four possible worlds that map $(0,0)$ to these outputs (see [15] 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 $(0,0,0), (0,0,1), (0,1,0)$ and $(0,1,1)$; 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 $x$, $\text{OUT}_{x,m,H} = \{(0,1,1), (1,1,0), (1,0,1)\}$, 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 $R' = \Pi_A(R)$ 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, $m_1, \ldots, m_n$ 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 $\text{Worlds}(R,H)$) 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) $\Pi_{O_i}(t') = m_i(\Pi_i(t'))$ for every public module $m_i$ in $W$ and every tuple $t' \in R'$.

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

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

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

Similar to standalone module privacy, $\Gamma$-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 \( \Gamma \). The results and proofs in this paper remain unchanged when different modules \( m_i \) have different privacy requirements \( \Gamma_i \). 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 \( \text{OUT}_{x,W,H} \) while the latter uses conjunction \( (\wedge) \) for defining \( \text{OUT}_{x,m,H} \). 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 \( \Gamma \), 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 \( \text{cost}(H) \) denotes the penalty of hiding the attributes in \( H \), a natural goal is to choose a safe subset \( H \) that minimizes \( \text{cost}(H) \). A particular instance of the problem is when the cost function is additive: each attribute \( a \) has some penalty value \( \text{cost}(a) \) and the penalty of hiding \( H \) is \( \text{cost}(H) = \sum_{a \in H} \text{cost}(a) \).

On the negative side, it was shown in \([15]\) 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 \( \Gamma \)-workflow-privacy for all of them.

**Theorem 1. (Composability Theorem for All-private Workflows \([15]\))** 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 \( \Gamma \)-standalone-privacy of \( m_i \). Then the workflow \( W \) is \( \Gamma \)-private with respect to hidden attributes \( H = \bigcup_{i=1}^n H_i \).

It was also observed in \([15]\) 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 ensure 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 \([1][15]\) focused on a modified optimization problem: combine standalone-private solutions optimally to get a workflow-private solution. This optimization problem, which we refer to as optimal composition problem, remains NP-hard even in the simplest scenario, and therefore, \([15]\) proposed efficient approximation algorithms.

---

\(^4\)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 \( C = R \).
3 Privacy via propagation

Workflows with both public and private modules are harder to handle than workflows with all private modules. In particular, the 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 $\Gamma$-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 $\Gamma$-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 $\Gamma_1$-standalone-privacy for a private module $m$ will only yield $\Gamma_2$-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 $\Gamma_1$ 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 $\Gamma_1$-workflow-privacy, where $\Gamma_1$ is the size of the range of the module. On the other hand hiding all output attributes can give a maximum of $\Gamma_2$-workflow-privacy, where $\Gamma_2$ 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.
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 $m_{i_1}, m_{i_2}, \ldots, m_{i_j}$ 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 $m_5$. For private module $m_2$, if hidden output attributes...
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, 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 x, y, z, etc. to denote tuples in the workflow or module relations with one or more attributes.

Definition 7. Given two tuples x and y on a subset of attributes B \subseteq A, and a subset of hidden attributes H \subseteq A, we say that x \equiv_H y iff \Pi_{B \setminus H}(x) = \Pi_{B \setminus H}(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 x, x' to m_i with respect to H, their outputs are also equivalent:
  \[ x \equiv_H x' \Rightarrow [m_i(x) \equiv_H m_i(x')] \]

- **upstream-safe** (or, **U-safe** in short) with respect to H if for any two equivalent outputs y, y' of m_i with respect to H, all of their preimages are also equivalent:
  \[ [(y \equiv_H y') \land (m_i(x) = y, m_i(x') = y')] \Rightarrow [x \equiv_H x'] \]

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

Note that if H = A (i.e. all attributes are hidden) then m_i is clearly UD-safe with respect 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_3. 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 $m_5,m_8$ have an undirected (also directed) public path 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). Then 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 $\Gamma$-standalone-privacy of $m_i$, (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) $H_i \subseteq O_j \cup \bigcup_{j \in C(H_i)} A_j$. Then the workflow $W$ is $\Gamma$-private with respect to $H = \bigcup_{j \in m}$ 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) $\hat{m}_i$ 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 $\hat{m}_i$ in the given world. Note that $m_i$ and $\hat{m}_i$, viewed as relations, agree on the visible attributes of 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, 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 $\Gamma$-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 $m_i$, or, (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 condition (i). Examples for the remaining conditions can be found in Appendix A.1.

Consider the workflow $W_6$ 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 $a_5 = m_3(a_3, a_4) = a_3 \lor a_4$ (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 $W_6$ 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.

When $a_3$ is hidden, a possible candidate output of input $a_4 = 0$ to $m_1$ is 1. So we need to have a possible world where $m_1$ is redefined as $\hat{m}_1(0) = 1$. 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).
Necessity of UD-safety 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.

**Proposition 2.** There is a workflow $W$ with a private module $m_i$, and a safe subset of hidden attributes $h_i$ guaranteeing $\Gamma$-standalone-privacy for $m_i (\Gamma > 1)$, such that satisfying only the downward-safety condition for the public modules in $C(h_i)$ does not give $\Gamma$-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) $a_5 = m_2(a_3, a_4) = a_3 \lor a_4$ (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 $\Gamma$-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 $a_5 = 0, a_4 = 0$).

---

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

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

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

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 $\hat{m}_4$ of private module $m_4$ is, it cannot map $a_5$ to both 0 and 1; $\hat{m}_4$ must be a function and maintain the functional dependency $a_5 \rightarrow a_6$. Hence all possible worlds of $R_a$ must map $\hat{m}_1(0)$ to 0, and therefore $\Gamma = 1$. □
then \( a_3 = 0 \) and if \( a_5 = 1, a_6 = 0, \) then \( a_3 = 1 \). On attributes \((a_1, a_2, a_3, a_4, a_5, a_6)\), 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 \((1, 0)\). So we need to have a possible world where \( m_1 \) is redefined as \( \hat{m}_1(0, 0) = (1, 0) \).

To be consistent on the visible attributes, this forces us to redefine \( m_3 \) where \( \hat{m}_3(1) = 0 \); otherwise the row \((0, 0, 0, 0)\) in \( R \) changes to \((0, 0, 1, 0, 1)\). This in turn forces us to define \( \hat{m}_1(1, 0) = (0, 0) \) and \( \hat{m}_3(0) = 1 \). This is because if we map \( \hat{m}_3(1) \) to any of \( \{(1, 0), (0, 1), (1, 1)\} \), either we have inconsistency on the visible attribute \( a_4 \), or \( a_5 = 1 \), and \( \hat{m}_3(1) = 0 \), 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 \( a_3 \), \( \hat{m}_1(1, 1) \) can take value \((1, 1)\) or \((0, 1)\). But if \( \hat{m}_1(1, 1) = (1, 1) \) 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 \( \hat{m}_3(1) = 0 \), 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 \((1, 1, 1, 1, 1, 1)\), we must have \( a_5 = 0 \) (since \( \hat{m}_3(0) = 1 \)). The pre-image of \( a_5 = 0 \) is \( a_3 = 0, a_4 = 0 \), hence we have to redefine \( \hat{m}_1(1, 1) = (0, 0) \). But \((0, 0)\) is not equivalent to original \( m_1(1, 1) = (1, 1) \), 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 the proof of Lemma 1 (Section 4.1.2) that this is indeed the case.

### 4.1.2 Proof of Composability Theorem

To prove \( \Gamma \)-privacy, we need to show the existence of at least \( \Gamma \) 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 \( x \) to \( m_i \), and any candidate output \( y \in \text{OUT}_{x, m_i, h_i} \) of \( x \). Then \( y \in \text{OUT}_{x, w, h} \) when \( m_i \) belongs to a single-predecessor workflow.
We show that if every public module in the composite module \( C \) contains a single public module, then \( M \) is UD-safe with respect to the hidden attributes.

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

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 \( y \in \text{OUT}_{x,W,H_i} \).

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.

**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. Due to space constraints, the proof is given in Appendix A.2 and 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 \( m_1 = m_1 \). Hiding its output \( h_1 = \{a_3\} \) gives \( \Gamma \)-standalone-privacy for \( \Gamma = 2 \). Fix input \( x = (0,0) \), with original output \( z = m_1(x) = (0,0) \) (hidden attribute \( a_3 \) is underlined). Also fix a candidate output \( y = (1,0) \in \text{OUT}_{x,m_1,h_1} \). Note that \( y \) and \( z \) 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 \( \hat{m}_1 \) as follows: \( \hat{m}_1 \) simply maps all pre-images of \( y \) to \( z \), and all pre-images of \( z \) to \( y \). In this...
In this case, both \( y, z \) have single pre-image. So \( x = (0, 0) \) gets mapped to \((1, 0)\) and input \((1, 0)\) gets mapped to \((0, 0)\). 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 \((0, 0, 1, 0, 0)\) and the third row will be \((1, 0, 0, 0, 1)\)).

Next consider the more complicated case, when the modules in \( C(h_1) \) 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-safety). Let us now describe the proof strategy when \( a_4 \) is hidden, i.e. \( H_1 = \{a_3, a_4, a_5\} \).

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

Consider the relation \( R' \) formed by joining the relations of \( \widehat{m}_1, m_2, \widehat{m}_3 \) (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 \( x = (0, 0) \) to \( y = (1, 0) \) as desired, i.e. \( y \in \text{OUT}_{x,W,H_1} \).

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), 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 \( w_y, w_z \) 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 \( m_i \) considered in the lemma.

Proof of Step-2. The following lemma formalizes the claim in Step-2:

**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 \( (I \cup O) \cap H \).

**Sketch.** The formal proof of this lemma is given in Appendix A.3. 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 $m_i, \ldots, m_k$ (in Figure 2(b) $k = 4$ 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 $m_i, \ldots, m_j$, 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, $M^1 = \{m_i\}$ 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 $m_i$, $1 \leq \ell \leq j$ are downstream-safe with respect to $(I_i \cup O_i) \cap H = A_i \cap H$. For upstream-safety, we consider the modules in reverse topological order, $m_i, \ldots, m_1$, and give a similar argument by an induction on $j = k$ down to 1.

Proof of Theorem 2 We first argue that if $H_i$ satisfies the conditions in Theorem 2 then $H_i = \bigcup_{j: m_j \in H_i} A_j$ satisfies the conditions in Lemma 1. Since $h_i = H_i \cap O_i$, (i) $h_i \subseteq H_i \subseteq \bigcup_{j: m_j \in H_i} A_j$; and (ii) $h_i \subseteq O_i$. Next we argue that the third requirement in the lemma, (iii) every module $m_j$ in the public-closure $C(h_i)$ is UD-safe with respect to $H_i \cap A_j$, also holds.

To see (iii), observe that the Theorem 2 has an additional condition on $H_i$: $H_i \subseteq O_i \cup \bigcup_{j: m_j \in H_i} A_j$. Since $W$ is a single-predecessor workflow, for two private modules $m_i, m_\ell$, 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 $m_j, m_\ell$ in $W$ (public or private), the set of attributes $A_j \cap A_\ell = \emptyset$. Clearly, when $m_j$ is a private module, $m_j \notin C(h_i)$ for any private module $m_j$ in $W$, by the definition of public-closure. Hence for any two private modules $m_i, m_\ell$,

$$\left( O_i \cup \bigcup_{j: m_j \in C(h_i)} A_j \right) \cap \left( O_\ell \cup \bigcup_{j: m_j \in C(h_\ell)} A_j \right) = \emptyset.$$

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 $m_\ell$, $A_j \cap H_\ell = \emptyset$. Therefore, $A_j \cap H_\ell = A_j \cap (\bigcup_{j: m_j \in H_i} C(h_i)) = A_j \setminus H_i$. 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 $A_j \cap H_i$. This shows that $H_i$ satisfies the conditions stated in the lemma.

Theorem 2 also states that each private module $m_i$ is $\Gamma$-standalone-private with respect to $h_i$, i.e., $|\text{OUT}_{x, m_i, h_i}| \geq \Gamma$ for all input $x$ to $m_i$ (see Definition 2). From Lemma 1 using $H_i'$ in place of $H_i$, this implies that for all input $x$ to private modules $m_i$, $|\text{OUT}_{x, W, H_i}'| \geq \Gamma$ where $H_i' = \bigcup_{j: m_j \in H_i} C(h_j)$. From Definition 4 this implies that each private module $m_i$ is $\Gamma$-workflow-private $H_i'$ which is the same as $H$ in Theorem 2. Since this is true for all private module $m_i$ in $W$, $W$ is $\Gamma$-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 cost($H$) is minimized for $H = \bigcup_{j: m_j \in H_i} C(h_i)$. 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 \( S_i = \{ S_{i1}, \cdots, S_{ip_i} \} \), where each \( S_{it} \subseteq O_t \) 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 \( H_i \)-s 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 [15] 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 [15], the number of attributes of an individual module is fairly small. The assumption of constant domain is reasonable for practical purposes, assuming that the integers and reals are represented in a fixed number of bits. In these cases the individual relations can be big, however this computation can be done only once as a pre-processing step and the cost can be amortized over possibly many uses of the module in different workflows. Expert knowledge (from the module designer) can also be used to help find the safe subsets.

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 \( U_j = \{ U_{j1}, \cdots, U_{jp_j} \} \), where each UD-safe subset \( U_{jt} \subseteq A_j \), and \( p_j \) 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 \( k \).

**Sketch of NP-hardness.** The reduction is from the UNSAT problem, where given \( n \) variables \( x_1, \cdots, x_n \), and a 3NF 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 \( z = m_i(x_1, \cdots, x_n, y) = f(x_1, \cdots, x_n) \lor y \) (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 same construction in the NP-hardness proof, 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 is given in Appendix \[B.1\].

### 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) \( H_i \cap O_i = S_i \), for some safe subset \( S_i \subseteq S \); (b) for every public module \( m_j \) in the closure \( C(S_i) \), there exists a UD-safe subset \( U_{jq} \subseteq U_j \) such that \( U_{jq} = A_j \cap H_i \); and (c) \( H_i \) does not include any attribute outside \( O_i \) and \( C(S_i) \).

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 \( S_i \) and \( U_j \) (denoted by \( L = \max_{i\in[1,n]} p_i \)):

**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 \( S_i \subseteq S \), we show below how an optimal subset \( H_i \) in \( C(S_i) \) satisfying the desired properties can be obtained. We then repeat this process for all safe subsets (bounded by \( L \) \( S_i \subseteq S \)), and output the subset \( H_i \) with minimum cost. We drop the subscripts to simplify the notation (*i.e.* use \( S \) for \( S_i \), \( C \) for \( C(S_i) \), 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 \( m_1, \ldots, m_k \) 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 \( A_j \cap H^{j, \ell} = U_j \subseteq U_j \). 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 \( 1 \leq \ell \leq p_1 \),

\[
Q[1, \ell] = \begin{cases} 
  c(U_1, \ell) & \text{if } U_1, \ell \supseteq S \\
  \infty & \text{otherwise}
\end{cases}
\]
Recall that for a chain, \( O_{j-1} = I_j \), for \( j = 2 \) to \( k \). Then for \( j = 2 \) to \( k \), \( \ell = 1 \) to \( p_j \),

\[
Q[j,\ell] = \begin{cases} 
\infty & \text{if there is no } 1 \leq q \leq p_{j-1} \\
\text{such that } U_{j-1,q} \cap O_{j-1} = U_{j,\ell} \cap I_j & \\
c\left(O_j \cap U_{j,\ell}\right) + \min_{q} Q[j-1,q] & \\
\end{cases}
\]

where the minimum is taken over all such \( q \).

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 \( A_{j-1} \in U_{j-1} \) and \( A_j \in U_j \), which will make the equality check true (for a chain \( O_{j-1} = I_j \)). In Appendix B.2 we show that shows that \( Q[j,\ell] \) 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 is given in Appendix B.3.

**NP-hardness for public-closure of arbitrary shape.** Finding the minimal-cost solution for public-closure with arbitrary DAG shape is NP-hard. We give a reduction from 3SAT (see Appendix B.4). 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; a corresponding EXP-time algorithm that iterates over all subsets can be used to find the optimal solution.

The NP-completeness here is in \( n \), the number of modules in the public closure. We note, however, that in practice the number of public modules that process the output on an individual private module is small. So the obtained solution to the optimum-view problem is still better than 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, \( H = \bigcup_{i; \text{is private}} H_i \) is a \( \Gamma \)-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, \( H_i \cap H_k = \emptyset \). This implies that the optimal solution \( H \) can be obtained 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 Appendix C.

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 the D-safety property 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 guaranteed to be disjoint. This is resolved by proving that D-safety 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 give the optimal solution 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 D-safety properties 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
algorithm, and choose the lower cost solution.

6 Related Work

Privacy concerns with respect to provenance were articulated in [16], in the context of scientific workflows, and in [17], in the context of business processes. Preserving module privacy in all-private workflows was studied in [15] 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 [30, 32, 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; furthermore, it may hide more provenance information than our mechanism. More importantly, the notion of privacy is informal and no guarantees on the quality of the solutions are provided.

A related area is that of privacy-preserving data mining (see surveys [4, 33], 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. [29]), network routing [24], and several other contexts.

Our notion of module privacy is closest to the notion of ℓ-diversity considered in [27] which addresses some shortcomings of κ-anonymity [31]. 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 [37]. Leakage of information due to knowledge on the techniques for minimizing data loss has been studied in [34, 22, 14, 35]; 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 [23, 25]. Differential privacy [20, 18, 19], 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 [28], data streaming [21], and several database-related applications (e.g. [26, 36, 13, 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 (see e.g. [12]).
7 Conclusion

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 $\Gamma$-privacy and the difficulty in using stronger privacy notions like differential privacy in the previous section. 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.

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. Wherfore 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] G. Cormode. Personal privacy vs population privacy: learning to attack anonymization. In KDD, pages 1253–1261, 2011.

[13] G. Cormode, C. M. Procopiuc, D. Srivastava, and T. T. L. Tran. Differentially private publication of sparse data. CoRR, abs/1103.0825, 2011.
[14] 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.

[15] S. B. Davidson, S. Khanna, T. Milo, D. Panigrahi, and S. Roy. Provenance views for module privacy. In *PODS*, pages 175–186, 2011.

[16] S. B. Davidson, S. Khanna, S. Roy, J. Stoyanovich, V. Tannen, and Y. Chen. On provenance and privacy. In *ICDT*, pages 3–10, 2011.

[17] D. Deutch and T. Milo. A quest for beauty and wealth (or, business processes for database researchers). In *PODS*, pages 1–12, 2011.

[18] C. Dwork. Differential privacy: A survey of results. In *TAMC’08*, pages 1–19.

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

[20] C. Dwork, F. Mcsherry, K. Nissim, and A. Smith. Calibrating noise to sensitivity in private data analysis. In *TCC*, pages 265–284. Springer, 2006.

[21] C. Dwork, M. Naor, T. Pitassi, G. N. Rothblum, and S. Yekhanin. Pan-private streaming algorithms. In *ICS*, pages 66–80, 2010.

[22] C. Fang and E.-C. Chang. Information hiding. chapter Information Leakage in Optimal Anonymized and Diversified Data, pages 30–44. 2008.

[23] S. R. Ganta, S. P. Kasiviswanathan, and A. Smith. Composition attacks and auxiliary information in data privacy. In *KDD*, pages 265–273, 2008.

[24] 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*.

[25] D. Kifer. Attacks on privacy and definiti’s theorem. In *SIGMOD*, pages 127–138, 2009.

[26] D. Kifer and A. Machanavajjhala. No free lunch in data privacy. In *SIGMOD*, pages 193–204, 2011.

[27] A. Machanavajjhala, J. Gehrke, D. Kifer, and M. Venkitasubramaniam. 1-Diversity: Privacy Beyond k-Anonymity. In *ICDE*, 2006.

[28] F. McSherry and K. Talwar. Mechanism design via differential privacy. In *FOCS*, pages 94–103, Washington, DC, USA, 2007. IEEE Computer Society.

[29] S. U. Nabar, B. Marthi, K. Kenthapadi, N. Mishra, and R. Motwani. Towards robustness in query auditing. In *VLDB*, pages 151–162, 2006.

[30] 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.

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

[32] V. Tan, P. T. Groth, S. Miles, S. Jiang, S. Munroe, S. Tsaskou, and L. Moreau. Security issues in a SOA-based provenance system. In *IPAW’06*, pages 203–211.

[33] 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.

[34] 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.

[35] 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.

[36] X. Xiao, G. Wang, and J. Gehrke. Differential privacy via wavelet transforms. In *ICDE*, pages 225–236, 2010.

[37] C. Yao, X. S. Wang, and S. Jajodia. Checking for k-anonymity violation by views. In *VLDB*, pages 910–921, 2005.
Table 4: Relation $R_a$ for workflow $W_a$ given in Figure 6a

|   | $a_0$ | $a_1$ | $a_2$ | $a_3$ | $a_4$ | $a_5$ | $a_6$ |
|---|-------|-------|-------|-------|-------|-------|-------|
| $r_1$ | 0     | 0     | 0     | 0     | 0     | 0     | 0     |
| $r_2$ | 1     | 0     | 1     | 0     | 1     | 1     | 0     |
| $r_3$ | 0     | 0     | 1     | 0     | 1     | 1     | 1     |
| $r_4$ | 1     | 1     | 1     | 1     | 1     | 1     | 1     |

A Proofs from Section 4

A.1 Proof of Proposition 1

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 the public-closure of a private module $m_i$ but has no directed path from $m_i$, or, (ii) such a public module $m_j$ has directed path from more than one private modules, or, (iii) $W$ has data sharing.

To prove the proposition we provide three example workflows where exactly one of the violating conditions (i), (ii), (iii) holds, and Theorem 2 does not hold in those workflows. Case (i) was shown in Section 4.1.1. To complete the proof we demonstrate here cases (ii) and (iii).

Multiple private predecessor We give an example where Theorem 2 does not hold when a public module belonging to a public-closure has more than one private predecessors.

Example 8. Consider the workflow $W_a$ in Figure 6a which is a modification of $W_a$ by the addition of private module $m_0$, that takes $a_0$ as input and produces $a_2 = m_0(a_0) = a_0$ as output. The public module $m_3$ is in public-closure of $m_1$, but has directed public paths from both $m_0$ and $m_1$. The relation $R_a$ for $W_a$ in given in Table 4 where the hidden attributes $\{a_2, a_3, a_4, a_5\}$ are colored in grey.

Now we have exactly the same problem as before: When $\hat{m}_1$ maps 0 to 1, $a_5 = 1$ irrespective of the value of $a_4$. In the first row $a_6 = 0$, whereas in the second row $a_6 = 1$. However, whatever the new definitions of $\hat{m}_0$ are for $m_0$ and $\hat{m}_4$ for $m_4$, $\hat{m}_4$ cannot map 1 to both 0 and 1. Hence $\Gamma = 1$. □

Data sharing Now we give an example where Theorem 2 does not hold when the workflow has data sharing.

Example 9. Consider the workflow, say $W_b$, given in Figure 6b. All attributes take values in $\{0, 1\}$. The initial inputs are $a_1, a_2$, and final outputs are $a_6, a_7$; only $m_4$ is public. The functionality of modules is as follows: (i) $m_1$ takes $a_1, a_2$ as input and produces $m_1(a_1, a_2) = (a_3 = a_1, a_4 = a_2)$. (ii) $m_2$ takes $a_3, a_4$ as input and produces $a_5 = m_2(a_3, a_4) = a_3 \lor a_4$ (OR). (iii) $m_3$ takes $a_5$ as input and produces $a_6 = m_3(a_5) = a_5$. (iv) $m_4$ takes $a_3$ as input and produces $a_7 = m_4(a_3) = a_3$. Note that data $a_3$ is input to both $m_2, m_4$, hence the workflow has data sharing.

Now focus on private module $m_1 = m_k$. Clearly hiding output $a_3$ of $m_1$ gives 2-standalone privacy, and for hidden attribute $h_1 = \{a_3\}$, the public-closure $C(h_1) = \{m_2\}$. As given in the theorem, $H_1 \subseteq O_1 \cup \bigcup_{j, m_j \in C(h_1)} A_j = \{a_3, a_4, a_5\}$ in this case.

We claim that hiding even all of $\{a_3, a_4, a_5\}$ gives only trivial 1-workflow-privacy of $m_1$, although the UD-safe condition is satisfied for $m_2$ (actually hiding $a_3, a_4$ gives 4-standalone-privacy for $m_1$). Table 5 gives the relation $R_b$, where the hidden attribute values are in Grey.
When \( a_3 \) (and also \( a_4 \)) is hidden, a possible candidate output of input tuple \( x = (0,0) \) to \( m_1 \) is \((1,0)\). So we need to have a possible world where \( m_1 \) is redefined as \( \hat{m}_1(0,0) = (1,0) \). Then \( a_5 \) takes value 1 in the first row, and this is the only row with visible attributes \( a_1 = 0, a_2 = 0 \). So this requires that \( \hat{m}_3(a_5 = 1) = (a_6 = 0) \) and \( \hat{m}_4(a_3 = 1) = (a_7 = 0) \), to have the same projection on visible \( a_6, a_7 \).

The second, third and fourth rows, \( r_2, r_3, r_4 \), have \( a_6 = 1 \), so to have the same projection, we need \( a_5 = 0 \) for these three rows, so we need \( \hat{m}_3(a_5 = 0) = (a_6 = 1) \) (since we had to already define \( \hat{m}_3(1) = (0) \)). When \( a_5 \) is 0, since the public module \( m_2 \) is an OR function, the only possibility of the values of \( a_3, a_4 \) in rows \( r_2, r_3, r_4 \) are \((0,0)\). Now we have a conflict on the value of the visible attribute \( a_7 \), which is 0 for \( r_2 \) but 1 for \( r_3, r_4 \), whereas for all these rows the value of \( a_3 \) is 0. \( \hat{m}_3 \) being a function with dependency \( a_3 \rightarrow a_7 \), cannot map \( a_3 \) to both \( 0 \) and \( 1 \). Similarly we can check that if \( \hat{m}_1(0,0) = (0,1) \) or \( \hat{m}_1(0,0) = (1,1) \) (both \( a_3, a_4 \) are hidden), we will have exactly the same problem. Hence all possible worlds of \( R_d \) with these hidden attributes must map \( \hat{m}_1(0,0) \) to \((0,0)\), and therefore \( \Gamma = 1 \).

\[ □ \]

### A.2 Proof of Lemma 1

The proof of Lemma 1 uses the following lemma. It states that the if \( y \) is a candidate output of an input \( x \) to module \( m_i \) with respect to hidden attributes \( h_i \) (i.e. \( y \in \text{OUT}_{x,m_i,h_i} \)), then \( y \) and the actual output of \( x \), \( z = m_i(x) \), must be equivalent.

**Lemma 3.** Let \( m_i \) be a standalone private module with relation \( R_i \), let \( x \) be an input to \( m_i \), and let \( h_i \subseteq O_i \) be a subset of hidden attributes. If \( y \in \text{OUT}_{x,m_i,h_i} \) then \( y \equiv_{h_i} z \) where \( z = m_i(x) \).
Note that, in Example 7, \( y = (1, 0) \) and \( z = (0, 0) \) are equivalent on the visible attributes as Lemma 8 says (hidden attributes are underlined).

**Proof.** A subset of output attributes of \( m_i, h_i \subseteq O_i \), is hidden. Recall that \( A_i = I_i \cup O_i \) denotes the set of attributes of \( m_i \) and let \( R_i \) be the standalone relation for \( m_i \). If \( y \in \text{OUT}_{x,m_i,h_i} \), then from Definition 2,

\[
\exists R' \in \text{Worlds}(R_i, h_i), \; \exists \ t' \in R' \; \text{s.t.} \; x = \Pi_{h_i}(t') \wedge y = \Pi_{O_i}(t')
\]

(1) Further, from Definition 1, \( R' \in \text{Worlds}(R_i, h_i) \) only if \( \Pi_{A_i \setminus h_i}(R_i) = \Pi_{A_i \setminus h_i}(R') \). Hence there must exist a tuple \( t \in R_i \) such that

\[
\Pi_{A_i \setminus h_i}(t) = \Pi_{A_i \setminus h_i}(t')
\]

(2) Since \( h_i \subseteq O_i, I_i \subseteq A_i \setminus h_i \). From (2), \( \Pi_{I_i}(t) = \Pi_{I_i}(t') = x \). Let \( z = \Pi_{O_i}(t) \), i.e. \( z = m_i(x) \). From (2), \( \Pi_{O_i \setminus h_i}(t) = \Pi_{O_i \setminus h_i}(t') \), then \( \Pi_{O_i \setminus h_i}(z) = \Pi_{O_i \setminus h_i}(y) \). Tuples \( y \) and \( z \) are defined on \( O_i \). Hence from Definition 7, \( y \equiv_{h_i} z \).

**Corollary 1.** For a module \( m_i \) and hidden attributes \( h_i \subseteq O_i \), if two tuples \( y, z \) defined on \( O_i \) are such that \( y \equiv_{h_i} z \), then also \( y \equiv_{H_i} z \) where \( H_i \supseteq h_i \) is a set of hidden attributes in the workflow.

**Proof.** Since \( y \equiv_{h_i} z \), \( \Pi_{O_i \setminus h_i}(y) = \Pi_{O_i \setminus h_i}(z) \). Since \( h_i \subseteq H_i \), \( O_i \setminus h_i \supseteq O_i \setminus H_i \). Therefore, \( \Pi_{O_i \setminus H_i}(y) = \Pi_{O_i \setminus H_i}(z) \), i.e. \( y \equiv_{H_i} z \).

**Note.** Lemma 3 does not use any property of single-predecessor workflows and also works for general workflows. This lemma will be used again for the privacy theorem of general workflows (Theorem 5).

**Definition of \( \text{FLIP} \) and \( \text{EFLIP} \) (extended \( \text{FLIP} \)) functions.** To prove Lemma 1, we need to show existence of a possible world satisfying the criteria. This possible world will be obtained by joining alternative definitions of private modules, and the original definition of public modules. We will need the following flipping functions to formally present how we derive the alternative module definitions from original modules. These function examines parts of inputs, and possibly changes parts of original outputs.

**Definition 10.** Given subsets of attributes \( P, Q \subseteq A \), two tuples \( p, q \) defined on \( P \), and a tuple \( u \) defined on \( Q \), \( \text{FLIP}_{p,q}(u) = w \) is a tuple defined on \( Q \) constructed as follows:

- if \( \Pi_{Q \setminus P}(u) = \Pi_{Q \setminus P}(p) \), then \( w \) is such that \( \Pi_{Q \setminus P}(w) = \Pi_{Q \setminus P}(q) \) and \( \Pi_{Q \setminus P}(w) = \Pi_{Q \setminus P}(u) \).
- else if \( \Pi_{Q \setminus P}(u) = \Pi_{Q \setminus P}(q) \), then \( w \) is such that \( \Pi_{Q \setminus P}(w) = \Pi_{Q \setminus P}(p) \) and \( \Pi_{Q \setminus P}(w) = \Pi_{Q \setminus P}(u) \).
- otherwise, \( w = u \).

The following observations capture the properties of \( \text{FLIP} \) function.

**Observation 1.**

1. If \( \text{FLIP}_{p,q}(u) = w \), then \( \text{FLIP}_{p,q}(w) = u \).
2. \( \text{FLIP}_{p,q}(\text{FLIP}_{p,q}(u)) = u \).
3. If \( P \cap Q = \emptyset \), \( \text{FLIP}_{p,q}(u) = u \).
4. F\text{LIP}_{p,q}(p) = q, F\text{LIP}_{p,q}(q) = p.

5. If \( \Pi_{Q\cap P}(p) = \Pi_{Q\cap P}(q) \), then F\text{LIP}_{p,q}(u) = u.

6. If \( Q = Q_1 \cup Q_2 \), where \( Q_1 \cap Q_2 = \emptyset \), and if F\text{LIP}_{p,q}(\Pi_{Q_1}(u)) = w_1 \) and F\text{LIP}_{p,q}(\Pi_{Q_2}(u)) = w_2, then F\text{LIP}_{p,q}(u) = w \text{ such that } \Pi_{Q_1}(w) = w_1 \text{ and } \Pi_{Q_2}(w) = w_2.

The above definition of flipping will be useful when we consider the scenario where \( M \) does not have any successor. When \( M \) has successors, we need an extended definition of tuple flipping based on other tuples, denoted by EFLIP, as defined below.

**Definition 11.** Given subsets of attributes \( P, Q, R \subseteq A \), where two tuples \( p, q \) defined on \( P \cup R \), a tuple \( u \) defined on \( Q \) and a tuple \( v \) defined on \( R \), EFLIP_{p,q,v}(u) = w \text{ is a tuple defined on } Q \text{ constructed as follows:}

- if \( v = \Pi_R(p) \), then \( w \text{ is such that } \Pi_{Q\cap P}(w) = \Pi_{Q\cap P}(q) \text{ and } \Pi_{Q\cap R}(w) = \Pi_{Q\cap R}(p) \).
- else if \( v = \Pi_R(q) \), then \( w \text{ is such that } \Pi_{Q\cap P}(w) = \Pi_{Q\cap P}(p) \text{ and } \Pi_{Q\cap R}(w) = \Pi_{Q\cap R}(q) \).
- otherwise, \( w = u \).

**Observation 2.**

1. If EFLIP_{p,q,v}(u) = w, and \( u' \text{ is a tuple defined on } Q' \subseteq Q \), then EFLIP_{p,q,v}(u') = \Pi_{Q'}(w).

**Proof of Lemma**

Now we are ready to prove the lemma. As mentioned in Section 4.1.2, we will assume that there is a single (composite) public module \( M \) in the public closure \( C(h_i) \) of \( m_i \). Recall that \( I, O, A \) denote the set of input, output and all attributes of \( m_i \) respectively.

**Lemma**

Consider a standalone private module \( m_i \), a set of hidden attributes \( h_i \), any input \( x \) to \( m_i \), and any candidate output \( y \in \text{OUT}_{x,m_i,h_i} \) of \( x \). Then \( y \in \text{OUT}_{x,m_i,h_i} \) 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 \( H_i \).

**Proof:** We fix a module \( m_i \), an input \( x \) to \( m_i \), hidden attributes \( h_i \subseteq O_i \), and a candidate output \( y \in \text{OUT}_{x,m_i,

From the conditions in the lemma, a set \( H_i \) is hidden in the workflow where (i) \( h_i \subseteq H_i \), (ii) \( h_i \subseteq O_i \), and (iii) \( M \) is UD-safe with respect to \( H_i \). We will show that \( y \in \text{OUT}_{x,W,H_i} \). We prove this by showing the existence of a possible world \( R' \in \text{Worlds}(R,H_i) \), such that if \( \Pi_{I_i}(t) = x \) for some \( t \in R' \), then \( \Pi_{O_i}(t) = y \). Since \( y \in \text{OUT}_{x,m_i,h_i} \), by Lemma 3 \( y = m_i(x) \). We consider two cases separately based on whether \( M \) has no successor or at least one private successors.
**Case I.** First consider the easier case that \( M \) does not have any successor, so all outputs of \( M \) belong to the set of final outputs. We redefine the module \( m_i \) to \( \hat{m}_i \) as follows. For an input \( u \) to \( m_i \), \( \hat{m}_i(u) = \text{FLIP}_{yz}(m_i(u)) \). All public modules are unchanged, \( \hat{m}_j = m_j \). All private modules \( m_j \neq m_i \) are redefined as follows: On an input \( u \) to \( m_j \), \( \hat{m}_j(u) = \text{FLIP}_{yz}(m_j(u)) \). The required possible world \( R' \) is obtained by taking the join of the standalone relations of these \( \hat{m}_j \)'s, \( j \in [n] \).

First note that by the definition of \( \hat{m}_i \), \( \hat{m}_i(x) = y \) (since \( \hat{m}_i(x) = \text{FLIP}_{yz}(m_i(x)) = \text{FLIP}_{yz}(z) = y \), from Observation \( \text{[14]} \)). Hence if \( \Pi_{I_i}(t) = x \) for some \( t \in R' \), then \( \Pi_{O_j}(t) = y \).

Next we argue that \( R' \in \text{Worlds}(R,H_i) \). Since \( R' \) is the join of the standalone relations for modules \( \hat{m}_j \)'s, \( R' \) maintains all functional dependencies \( I_j \rightarrow O_j \). Also none of the public modules are unchanged, hence for any public module \( m_j \) and any tuple \( t \) in \( R' \), \( \Pi_{O_j}(t) = m_j(\Pi_{I_i}(t)) \). So we only need to show that the projection of \( R \) and \( R' \) on the visible attributes are the same.

Let us assume, wlog. that the modules are numbered in topologically sorted order. Let \( I_0 \) be the initial input attributes to the workflow, and let \( p \) be a tuple defined on \( I_0 \). There are two unique tuples \( t \in R \) and \( t' \in R' \) such that \( \Pi_{I_i}(t) = \Pi_{I_j}(t') = p \). Since \( M \) does not have any successor, let us assume that \( M = m_{n+1} \), also wlog. assume that the public modules in \( C \) are not counted in \( j = 1 \) to \( n + 1 \) by renumbering the modules. Note that any intermediate or final attribute \( a \in A \setminus I_0 \) belongs to \( O_j \), for a unique \( j \in [1,n] \) (since for \( j \neq \ell \), \( O_j \cap O_{\ell} = \emptyset \)). So it suffices to show that \( t,t' \) projected on \( O_j \) are equivalent with respect to visible attributes for all module \( j, j = 1 \) to \( n + 1 \).

Let \( c_{j,m}, c_{j,\hat{m}} \) be the values of input attributes \( I_j \) and \( d_{j,m}, d_{j,\hat{m}} \) be the values of output attributes \( O_j \) of module \( m_j \). In \( t \in R \) and \( t' \in R' \) respectively on initial input attributes \( p \) (i.e. \( c_{j,m} = \Pi_{I_i}(t), c_{j,\hat{m}} = \Pi_{I_j}(t'), d_{j,m} = \Pi_{O_j}(t) \) and \( d_{j,\hat{m}} = \Pi_{O_j}(t') \)). We prove by induction on \( j = 1 \) to \( n \) that

\[
\forall j, 1 \leq j \leq n, d_{j,\hat{m}} = \text{FLIP}_{yz}(d_{j,m}) \tag{3}
\]

First we argue that proving \( \text{(3)} \) shows that the join of \( \langle \hat{m}_i \rangle_{1 \leq i \leq n} \) is a possible world of \( R \) with respect to hidden attributes \( H_i \). (A) When \( m_j \) is a private module, note that \( d_{j,m} \) and \( d_{j,\hat{m}} = \text{FLIP}_{yz}(d_{j,m}) \) may differ only on attributes \( O_j \cap O_i \). But \( y \equiv_{h_0} z \), i.e. these tuples are equivalent on the visible attributes. Hence for all private modules, the \( t,t' \) are equivalent with respect to \( O_j \). (actually for all \( j \neq i \), \( O_j \cap O_i = \emptyset \), so the outputs are equal and therefore equivalent). (B) When \( m_j \) is a public module, \( j \neq n + 1 \), \( O_j \cap O_{\ell} = \emptyset \), hence the values of \( t,t' \) on \( O_j \) are the same and therefore equivalent. (C) Finally, consider \( M = m_{n+1} \) that is not covered by \( \text{(3)} \). \( M \) gets all its inputs from \( m_i \). From \( \text{(3)} \),

\[
d_{i,\hat{m}} = \text{FLIP}_{yz}(d_{i,m})
\]

Since \( y,z,d_{i,m}, d_{i,\hat{m}} \) are all defined on attributes \( O_i \), and input to \( m_{n+1} \), \( I_{n+1} \subseteq O_i \),

\[
c_{n+1,\hat{m}} = \text{FLIP}_{yz}(c_{n+1,m})
\]

Hence \( c_{n+1,\hat{m}} \equiv_{h_0} c_{n+1,m} \). Since these two inputs of \( m_{n+1} \) are equivalent with respect to \( H_i \), by the \text{UD-safe} \ property of \( M = m_{n+1} \), its outputs are also equivalent, i.e. \( d_{n+1,\hat{m}} \equiv_{h_0} d_{n+1,m} \). Hence the projections of \( t,t' \) on \( O_{n+1} \) are also equivalent. Combining (A), (B), (C), \( t,t' \) are equivalent with respect to \( H_i \).

**Proof of \( \text{(3)} \).** The base case follows for \( j = 1 \). If \( m_1 \neq m_i \) (\( m_j \) can be public or private), then \( I_1 \cap O_i = \emptyset \), so for all input \( u \),

\[
\hat{m}_j(u) = m_j(\text{FLIP}_{yz}(u)) = m_j(u)
\]
Since the inputs $c_{1,\tilde{m}} = c_{1,m}$ (both projections of initial input $p$ on $I_1$), the outputs $d_{1,\tilde{m}} = d_{1,m}$. This shows (3). If $m_1 = m_i$, the inputs are the same, and by definition of $\tilde{m}_1$,

$$
\begin{align*}
d_{1,\tilde{m}} &= \tilde{m}_1(c_{1,\tilde{m}}) \\
&= \text{FLIP}_{y,z}(m_i(c_{1,\tilde{m}})) \\
&= \text{FLIP}_{y,z}(m_i(c_{1,m})) \\
&= \text{FLIP}_{y,z}(d_{1,m})
\end{align*}
$$

This shows (3).

Suppose the hypothesis holds until $j - 1$, consider $m_j$. From the induction hypothesis, $c_{j,\tilde{m}} = \text{FLIP}_{y,z}(c_{j,m})$, hence $c_{j,m} = \text{FLIP}_{y,z}(c_{j,\tilde{m}})$ (see Observation 1(1)).

(i) If $j = i$, again,

$$
\begin{align*}
d_{i,\tilde{m}} &= \tilde{m}_i(c_{i,\tilde{m}}) \\
&= \text{FLIP}_{y,z}(m_i(c_{i,\tilde{m}})) \\
&= \text{FLIP}_{y,z}(m_i(\text{FLIP}_{y,z}(c_{i,m}))) \\
&= \text{FLIP}_{y,z}(m_i(c_{i,m})) \\
&= \text{FLIP}_{y,z}(d_{i,m})
\end{align*}
$$

$\text{FLIP}_{y,z}(c_{i,m}) = c_{i,m}$ follows due to the fact that $I_i \cap O_i = \emptyset$, $y, z$ are defined on $O_i$, whereas $c_{i,m}$ is defined on $I_i$ (see Observation 1(3)).

(ii) If $j \neq i$ and $m_j$ is a private module,

$$
\begin{align*}
d_{j,\tilde{m}} &= \tilde{m}_j(c_{j,\tilde{m}}) \\
&= m_j(\text{FLIP}_{y,z}(c_{j,\tilde{m}})) \\
&= m_j(c_{j,m}) \\
&= d_{j,m} \\
&= \text{FLIP}_{y,z}(d_{j,m})
\end{align*}
$$

$\text{FLIP}_{y,z}(d_{j,m}) = d_{j,m}$ follows due to the fact that $O_j \cap O_i = \emptyset$, $y, z$ are defined on $O_i$, whereas $d_{i,m}$ is defined on $O_j$ (again see Observation 1(3)).

(iii) If $m_j$ is a public module, $j \leq n$, $\tilde{m}_j = m_j$.

Here

$$
\begin{align*}
d_{j,\tilde{m}} &= \tilde{m}_j(c_{j,\tilde{m}}) \\
&= m_j(c_{j,\tilde{m}}) \\
&= m_j(\text{FLIP}_{y,z}(c_{j,m})) \\
&= m_j(c_{j,m}) \\
&= d_{j,m} \\
&= \text{FLIP}_{y,z}(d_{j,m})
\end{align*}
$$

30
Let the visible attributes are the same. Hence \( \Pi_{M \cap O_i} = \Pi_{M \cap O_j} \), so it suffices to show that the projection of \( R \) and \( R' \) on the visible attributes are the same.

Let \( I_0 \) be the initial input attributes to the workflow, and let \( p \) be a tuple defined on \( I_0 \). There are two unique tuples \( t \in R \) and \( t' \in R' \) such that \( \Pi_{I_0}(t) = \Pi_{I_0}(t') = p \). Note that any intermediate or final attribute \( a \in A \setminus I_0 \) belongs to \( O_j \), for a unique \( j \in [1, n] \) (since for \( j \neq \ell \), \( O_j \cap O_\ell = \emptyset \)). So it suffices to show that \( t, t' \) projected on \( O_j \) are equivalent with respect to visible attributes for all module \( j, j = 1 \) to \( n+1 \).
Let \( c_{j,m}, d_{j,m} \) be the values of input attributes \( I_j \) and \( m_j \) be the values of output attributes \( O_j \) of module \( m_j \), in \( t \in R \) and \( t' \in R' \) respectively on initial input attributes \( p \) (i.e. \( c_{j,m} = \Pi_I(t), d_{j,m} = \Pi_O(t') \), \( d_{j,m} = \Pi_{O_j}(t) \) and \( d_{j,m} = \Pi_{O_j}(t') \)). We prove by induction on \( j = 1 \) to \( n \) that

\[
\forall j \neq i, 1 \leq j \leq n, d_{j,i} = \text{FLIP}_{Y,Z}(d_{j,m}) \tag{4}
\]

\[
\Pi_{\hat{h}}(d_{i,m}) = \text{EFLIP}_{Y,Z,M}(\Pi_{\hat{h}}(d_{i,m}))(\Pi_{\hat{h}}(d_{i,m})) \tag{5}
\]

\[
\Pi_{O_j \setminus \hat{h}}(d_{i,m}) = \text{FLIP}_{Y,Z}(\Pi_{O_j \setminus \hat{h}}(d_{i,m})) \tag{6}
\]

First we argue that proving (4), (5) and (6) shows that the join of \( \langle \hat{m}_i \rangle \) with respect to \( H_i \)

(A) When \( m_j \) is a private module, \( j \neq i \), note that \( d_{j,m}, d_{j,i} = \text{FLIP}_{Y,Z}(d_{j,m}) \) may differ only on attributes \( (O_k \cup O_j) \cap O_j \). But for \( j \neq i \) and \( j \neq k \) (\( m_j \) is private module whereas \( m_k \) is the composite public module), \( (O_k \cup O_j) \cap O_j = \emptyset \). Hence for all private modules other than \( m_i \), the \( t,t' \) are equal with respect to \( O_j \) and therefore equivalent.

(B) For \( m_i \), from (5),

\[
\Pi_{\hat{h}}(d_{i,m}) = \text{EFLIP}_{Y,Z,M}(\Pi_{\hat{h}}(d_{i,m}))(\Pi_{\hat{h}}(d_{i,m})). \tag{5}
\]

Here \( \Pi_{\hat{h}}(d_{i,m}) \) and \( \Pi_{\hat{h}}(d_{i,i}) \) may differ on \( I_k \) only if \( M(\Pi_{\hat{h}}(d_{i,m})) \in \{w_y,w_z\} \). By Corollary cor:out-equiv, \( y \equiv_{H_i} z \), i.e. \( \Pi_{\hat{h}}(y) \equiv_{H_i} \Pi_{\hat{h}}(z) \). But since \( M \) is UDS, by the downstream-safety property, \( w_y \equiv_{H_i} w_z \). Then by the upstream-safety property, all inputs \( \Pi_{\hat{h}}(d_{i,m}) \equiv_{H_i} y \equiv_{H_i} z \) such that \( M(\Pi_{\hat{h}}(d_{i,m})) \in \{w_y,w_z\} \). In particular, if \( M(\Pi_{\hat{h}}(d_{i,m})) = w_y \), then \( \Pi_{\hat{h}}(d_{i,i}) = \Pi_{\hat{h}}(z) \), and \( \Pi_{\hat{h}}(z), \Pi_{\hat{h}}(d_{i,i}) \) will be equivalent with respect to \( H_i \). Similarly, if \( M(\Pi_{\hat{h}}(d_{i,m})) = w_z \), then \( \Pi_{\hat{h}}(d_{i,i}) = \Pi_{\hat{h}}(y) \), and \( \Pi_{\hat{h}}(y), \Pi_{\hat{h}}(d_{i,m}) \) will be equivalent with respect to \( H_i \). So \( t,t' \) are equivalent with respect to \( I_k \setminus H_i \).

Next we argue that \( t,t' \) are equivalent with respect to \( (O_i \setminus I_k) \setminus H_i \). From (5),

\[
\Pi_{O_j \setminus \hat{h}}(d_{i,m}) = \text{FLIP}_{Y,Z}(\Pi_{O_j \setminus \hat{h}}(d_{i,m}))
\]

\( \Pi_{O_j \setminus \hat{h}}(d_{i,m}) \) and \( \Pi_{O_j \setminus \hat{h}}(d_{i,i}) \) can differ only if

\[
\Pi_{O_j \setminus \hat{h}}(d_{i,i}) = \Pi_{O_j \setminus \hat{h}}(y). \tag{7}
\]

Then \( \Pi_{O_j \setminus \hat{h}}(d_{i,i}) = \Pi_{O_j \setminus \hat{h}}(z) \), or \( \Pi_{O_j \setminus \hat{h}}(d_{i,i}) = \Pi_{O_j \setminus \hat{h}}(z) \). Therefore, \( \Pi_{O_j \setminus \hat{h}}(d_{i,i}) = \Pi_{O_j \setminus \hat{h}}(y) \). But \( \Pi_{O_j \setminus \hat{h}}(y) \)

(C) When \( m_j \) is a public module, \( d_{j,m}, d_{j,i} = \text{FLIP}_{Y,Z}(d_{j,m}) \). Here \( d_{j,m}, d_{j,i} \) can differ only on \( (O_k \cup O_j) \cap O_j \). If \( j \neq k \), the intersection is empty, and we are done. If \( j = k \), \( d_{j,m}, d_{j,i} \) may differ only if \( d_{j,m} \in \{w_y,w_z\} \). But note that \( y \equiv_{H_i} z \), so \( \Pi_{\hat{h}}(y) \equiv_{H_i} \Pi_{\hat{h}}(z) \), and \( \Pi_{\hat{h}}(y) \equiv_{H_i} \Pi_{\hat{h}}(z) \). Since \( m_k \) is UDS, for these two equivalent inputs the respective outputs \( w_y,w_z \) are also equivalent. Hence in all cases the values of \( t,t' \) on \( O_k \) are equivalent.

Combining (A), (B), (C), the projections of \( t,t' \) on \( O_j \) are equivalent for all \( 1 \leq j \leq n \); i.e. \( t,t' \) are equivalent with respect to \( H_i \)
Proof of (4), (5) and (6). The base case follows for $j = 1$. If $m_1 \neq m_i$ ($m_1$ can be public or private, but $k \neq 1$ since $m_i$ is its predecessor), then $I_j \cap (O_i \cup O_k) = \emptyset$, so for all input $u$, $m_j(u) = m_j(\text{FLIP}_{Y,Z}(u)) = m_j(u)$ (if $m_1$ is private) and $\tilde{m}_j(u) = m_j(u)$ (if $m_1$ is public). Since the inputs $c_{1,\tilde{m}} = c_{1,m}$ (both projections of initial input $p$ on $I_j$), the outputs $d_{1,\tilde{m}} = d_{1,m}$. This shows (4). If $m_1 = m_i$, the inputs are the same, and by definition of $\tilde{m}_1$,

$$\Pi_{I_k}(d_{1,\tilde{m}}) = \Pi_{I_k}(\tilde{m}_1(c_{1,\tilde{m}})) = \text{FLIP}_{Y,Z}(\Pi_{O_i \setminus I_k}(m_1(c_{1,\tilde{m}})))$$

This shows (5) for $i = 1$. Again, by definition of $\tilde{m}_1$,

$$\Pi_{O_1 \setminus I_k}(d_{1,\tilde{m}}) = \Pi_{O_1 \setminus I_k}(\tilde{m}_1(c_{1,\tilde{m}})) = \text{FLIP}_{Y,Z}(\Pi_{O_1 \setminus I_k}(m_1(c_{1,\tilde{m}})))$$

This shows (6).

Suppose the hypothesis holds until $j - 1$, consider $m_j$. From the induction hypothesis, if $I_j \cap O_i = \emptyset$ ($m_j$ does not get input from $m_i$) then $c_{j,\tilde{m}} = \text{FLIP}_{Y,Z}(c_{j,m})$, hence $c_{j,m} = \text{FLIP}_{Y,Z}(c_{j,\tilde{m}})$ (see Observation 1).

(i) If $j = i$, $I_i \cap O_i = \emptyset$, hence $c_{i,\tilde{m}} = \text{FLIP}_{Y,Z}(c_{i,m}) = c_{i,m}$ ($I_i \cap (O_i \cup O_k) = \emptyset$, $m_k$ is a successor of $m_i$, so $m_i$ cannot be successor of $m_k$). By definition of $\tilde{m}_i$,

$$\Pi_{I_k}(d_{i,\tilde{m}}) = \Pi_{I_k}(\tilde{m}_i(c_{i,\tilde{m}})) = \text{FLIP}_{Y,Z}(\Pi_{O_i \setminus I_k}(m_i(c_{i,\tilde{m}})))$$

This shows (5).

Again,

$$\Pi_{O_i \setminus I_k}(d_{i,\tilde{m}}) = \Pi_{O_i \setminus I_k}(\tilde{m}_i(c_{i,\tilde{m}})) = \text{FLIP}_{Y,Z}(\Pi_{O_i \setminus I_k}(m_i(c_{i,\tilde{m}})))$$

This shows (6).

(ii) If $j = k$, $m_k$ gets all its inputs from $m_i$, so $\Pi_{I_k}(d_{i,m}) = c_{k,m}$. Hence

$$c_{k,\tilde{m}} = \text{FLIP}_{Y,Z,M}(\Pi_{I_k}(d_{i,m}))(c_{k,m}) = \text{FLIP}_{Y,Z,M}(c_{k,m})(c_{k,m})$$

33
Therefore,
\[
\mathbf{d}_{k,\hat{m}} = \hat{m}_k(\mathbf{c}_{k,\hat{m}}) \\
= m_k(\mathbf{c}_{k,\hat{m}}) \\
= m_k(\text{EFLIP}_YZ.\mathbf{d}_{k,m}(\mathbf{c}_{k,m}))
\]

Let's evaluate the term \(m_k(\text{EFLIP}_YZ.\mathbf{d}_{k,m}(\mathbf{c}_{k,m}))\). This says that for an input to \(m_k\) is \(\mathbf{c}_{k,m}\), and its output \(\mathbf{d}_{k,m}\), (a) if \(\mathbf{d}_{k,m} = \mathbf{w}_y\), then
\[
\text{EFLIP}_YZ.\mathbf{d}_{k,m}(\mathbf{c}_{k,m}) = \Pi_k(\mathbf{z})
\]
and in turn
\[
\mathbf{d}_{k,\hat{m}} = m_k(\text{EFLIP}_YZ.\mathbf{d}_{k,m}(\mathbf{c}_{k,m})) = \mathbf{w}_z;
\]
(b) if \(\mathbf{d}_{k,m} = \mathbf{w}_z\), then
\[
\text{EFLIP}_YZ.\mathbf{d}_{k,m}(\mathbf{c}_{k,m}) = \Pi_k(\mathbf{y})
\]
and in turn
\[
\mathbf{d}_{k,\hat{m}} = m_k(\text{EFLIP}_YZ.\mathbf{d}_{k,m}(\mathbf{c}_{k,m})) = \mathbf{w}_y;
\]
(c) otherwise
\[
\mathbf{d}_{k,\hat{m}} = m_k(\text{EFLIP}_YZ.\mathbf{d}_{k,m}(\mathbf{c}_{k,m})) = \mathbf{d}_{k,m}
\]

According to Definition 10, the above implies that
\[
\mathbf{d}_{k,\hat{m}} = \text{FLIP}_{\mathbf{w}_y,\mathbf{w}_z}(\mathbf{d}_{k,m}) = \text{FLIP}_YZ(\mathbf{d}_{k,m})
\]

This shows (4).

(iii) If \(j \neq i\) and \(m_j\) is a private module, \(m_j\) can get inputs from \(m_i\), (but since there is no data sharing \(I_j \cap \overline{I}_k = \emptyset\)), and other private or public modules \(m_{\ell}, \ell \neq i\) (\(\ell\) can be equal to \(k\)). Let us partition the input to \(m_j\) (\(\mathbf{c}_{j,m}\) and \(\mathbf{c}_{j,\hat{m}}\) defined on \(I_j\)) on attributes \(I_j \cap O_i\) and \(I_j \setminus O_i\). From (4), using the induction hypothesis,
\[
\Pi_{I_j \setminus O_i}(\mathbf{c}_{j,\hat{m}}) = \text{FLIP}_YZ(\Pi_{I_j \setminus O_i}(\mathbf{c}_{j,m}))
\]

Now \(I_k \cap I_j = \emptyset\), since there is no data sharing. Hence \((I_j \cap O_i) \subseteq (O_i \setminus I_k)\). From (6) using Observation 2.
\[ \Pi_{I_j \cap O_i}(e_{j,\hat{m}}) = \text{FLIP}_{Y,Z}(\Pi_{I_j \cap O_i}(e_{j,m})) \]  

(8)

From (7) and (8), using Observation 4, and since \( e_{j,m}, e_{j,\hat{m}} \) are defined on \( I_j \), so

\[ e_{j,\hat{m}} = \text{FLIP}_{Y,Z}(e_{j,m}) \]  

(9)

From (9),

\[ d_{j,\hat{m}} = \hat{m}_j(e_{j,\hat{m}}) \]

\[ = m_j(\text{FLIP}_{Y,Z}(e_{j,\hat{m}})) \]

\[ = m_j(e_{j,m}) \]

\[ = d_{j,m} \]

\[ = \text{FLIP}_{Y,Z}(d_{j,m}) \]

\( \text{FLIP}_{Y,Z}(d_{j,m}) = d_{j,m} \) follows due to the fact that \( O_j \cap (O_i \cup O_k) = \emptyset \) (\( j \neq \{i,k\} \)), \( Y,Z \) are defined on \( O_i \cup O_k \), whereas \( d_{j,m} \) is defined on \( O_j \) (again see Observation 1(3)).

(iv) Finally consider \( m_j \) is a public module such that \( j \neq k \). \( m_j \) can still get input from \( m_i \), but none of the attributes in \( I_j \cap O_j \) can be hidden by the definition of \( m_k = M = C(h_i) \). Further, by the definition of \( M = m_k \), \( m_j \) cannot get any input from \( m_k \) (\( M \) is the closure of public module); so \( I_j \cap O_k = \emptyset \). Let us partition the inputs to \( m_j \) (\( e_{j,m} \) and \( e_{j,\hat{m}} \) defined on \( I_j \)) into three two disjoint subsets: (a) \( I_j \cap O_i \), and (b) \( I_j \setminus O_i \). Since there is no data sharing \( I_k \cap I_j = \emptyset \), and we again get (9) that

\[ e_{j,\hat{m}} = \text{FLIP}_{Y,Z}(e_{j,m}) \]

\[ = e_{j,m} \]

\( \text{FLIP}_{Y,Z}(e_{j,m}) = e_{j,m} \) follows due to following reason. If \( I_j \cap O_i = \emptyset \), i.e. if \( m_j \) does not get any input from \( m_i \), again this is true (then \( I_j \cap (O_i \cup O_k) = (I_j \cap O_i) \cup (I_j \cap O_k) = \emptyset \)). If \( m_j \) gets an input from \( m_i \), i.e. \( I_j \cap O_i \neq \emptyset \), since \( j \neq k \), \( I_j \cap O_i \) does not include any hidden attributes from \( h_i \) (\( m_k \) is the closure \( C(h_i) \)). But \( y \equiv_{h_i} z \), i.e. the visible attribute values of \( y,z \) are the same. In other words, \( \Pi_{I_j \cap O_i}(y) = \Pi_{I_j \cap O_i}(z) \), and again from Observation 1(5),

\[ \text{FLIP}_{Y,Z}(e_{j,m}) = \text{FLIP}_{Y,Z}(e_{j,m}) = e_{j,m} \]

(again, \( I_j \cap O_k = \emptyset \)).

Therefore,

\[ d_{j,\hat{m}} = \hat{m}_j(e_{j,\hat{m}}) \]

\[ = m_j(e_{j,\hat{m}}) \]

\[ = m_j(\text{FLIP}_{Y,Z}(e_{j,m})) \]

\[ = m_j(e_{j,m}) \]

\[ = d_{j,m} \]

\[ = \text{FLIP}_{Y,Z}(d_{j,m}) \]

\( \text{FLIP}_{Y,Z}(d_{j,m}) = d_{j,m} \) again follows due to the fact that \( O_j \cap (O_i \cup O_k) = \emptyset \), since \( j \not\in \{i,k\} \).

Hence all the cases for the induction hypothesis hold true, and this completes the proof of the lemma for Case-II. \( \square \)
A.3 Proof of Lemma 2

Recall that \( I_i, O_i, A_i \) denote the set of input, output and all attributes of a module \( m_i \).

**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.** Let us assume, wlog., that the modules in \( M \) are \( m_1, \ldots, m_p \) where modules are listed in topological order. For \( j = 1 \) to \( p \), let \( M^j \) be the composite module comprising \( m_1, \ldots, m_j \), and let \( I^j, O^j \) be its input and output. Hence \( M^p = M, I^p = I \) and \( O^p = O \). We prove by induction on \( 2 \leq j \leq p \) that \( M^j \) is UD-safe with respect to \( H \cap (I^j \cup O^j) \). We present the proof without going through the notations for the sake of simplicity.

The base case directly follows for \( j = 1 \), since \( A_1 = I_1 \cup O_1 = I^1 \cup O^1 \). Let the hypothesis hold until \( M^j \) and consider \( M^{j+1} \). By induction hypothesis, \( M^j \) is UD-safe with respect to \((I^j \cup O^j) \cap H\). The module \( m_{j+1} \) may consume some outputs of \( M^j \) (\( m_2 \) to \( m_j \)). Hence

\[
I^{j+1} = I^j \cup I_{j+1} \setminus O^j \quad \text{and} \quad O^{j+1} = O^j \cup O_{j+1} \setminus I_{j+1}
\]

Consider two equivalent inputs \( x_1, x_2 \) with respect to hidden attributes \( H \cap (I^{j+1} \cup O^{j+1}) \). Therefore, their projection on visible attributes \( I^{j+1} \setminus H = (I^{j+1} \cup O^{j+1}) \setminus H \) are the same ———–(A)

Partition \( I^{j+1} \) into \( I^j \) and \( I^{j+1} \setminus I^j = I_{j+1} \setminus I^j \). Projection of \( x_1 \) and \( x_2 \), let \( x_{11}, x_{12} \), on \( I^j \setminus H \) will be the same. Therefore, the inputs to \( M^j \) are equivalent. By hypothesis, their outputs, say \( z_1, z_2 \) will have same values on \( O^j \setminus H = (I^{j+1} \cup O^{j+1}) \setminus H \) ———— (B).

Again, on inputs \( x_1, x_2 \) to \( M^{j+1} \), inputs to \( m_{j+1} \) will be concatenation of (i) projection of output \( z_1, z_2 \) from \( M^j \) on \( O^j \cap I_{j+1} \) and (ii) projection of \( x_1, x_2 \) on \( I_{j+1} \setminus I^j \). From (A) and (B), they will be equivalent on visible attributes \( (I^{j+1} \cup O^{j+1}) \setminus H \). Therefore, the inputs to \( M^j \) are equivalent with respect to \( H \). Since \( m_{j+1} \) is UD-safe, the outputs, say \( w_1, w_2 \) are also equivalent ———–(C).

Now note that \( y_1 \) is defined on \( O^{j+1} = (O^j \setminus I_{j+1}) \cup O_{j+1} \). Its projection on \( O^j \setminus I_{j+1} \) is projection of \( z_1 \) on \( O^j \setminus I_{j+1} \), and its projection on \( O_{j+1} \) is \( z_1 \). Similarly \( y_2 \) can be partitioned. From (B) and (C), the projections are equivalent, therefore the outputs \( y_1 \) and \( y_2 \) are equivalent.

This shows that for two equivalent input the outputs are equivalent. The other direction, for two equivalent outputs all of their inputs are equivalent can be proved in similar way by considering modules in reverse topological order from \( m_k \) to \( m_2 \).

\[\Box\]

B Proofs from Section 5

B.1 Proof of Theorem 3

**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 of NP-hardness

Proof. We do a reduction from UNSAT, where given $n$ variables $x_1, \ldots, x_n$, and a boolean formula $f(x_1, \ldots, x_n)$, the goal is to check whether $f$ is not satisfiable. In our construction, $m_i$ has $n + 1$ inputs $x_1, \ldots, x_n$ and $y$, and the output is $z = m_i(x_1, \ldots, x_n, y) = f(x_1, \ldots, x_n) \lor y$ (OR). Hidden attributes $H = \{x_1, \ldots, x_n\}$, so $y, z$ are visible. We claim that $f$ is not satisfiable if and only if $m_i$ is UD-safe with respect to $H$.

Suppose $f$ is not satisfiable, so for all assignments of $x_1, \ldots, x_n$, $f(x_1, \ldots, x_n) = 0$. For output $z = 0$, then the visible attribute $y$ must have 0 value in all the rows of the relation of $m_i$. Also for $z = 1$, the visible attribute $y$ must have 1 value, since in all rows $f(x_1, \ldots, x_n) = 0$. Hence for equivalent inputs with respect to $H$, the outputs are equivalent and vice versa. Therefore $m_i$ is UD-safe with respect to $H$.

Now suppose $f$ is satisfiable, then there is at least one assignment of $x_1, \ldots, x_n$, such that $f(x_1, \ldots, x_n) = 1$. In this row, for $y = 0, z = 1$. However for all assignments of $x_1, \ldots, x_n$, whenever $y = 1, z = 1$. So for output $z = 1$, all inputs producing $z$ are not equivalent with respect to the visible attribute $y$, therefore $m_i$ is not upstream-safe and hence not UD-safe.

Upper Bound to Find All UD-safe Solutions

The lower bounds studied for the second step of the four step optimization show that for a public module $m_j$, it is not possible to have poly-time algorithms (in $|A_j|$) even to decide if a given subset $H \subseteq A_j$ is UD-safe, unless $P = NP$. Here we present Algorithm 1 that finds all UD-safe solutions of $m_j$ is time exponential in $k_j = |A_j|$, assuming that the maximum domain size of attributes $\Delta$ is a constant.

Time complexity. The outer for loop runs for all possible subsets of $A_j$, i.e. $2^{k_j}$ times. The inner for loop runs for maximum $\Delta[|I_j \setminus H|]$ times (this is the maximum number of such tuples $x^+$), whereas the check if $H$ is a valid downstream-safe subset takes $O(\Delta[|I_j \setminus H|])$ time. Here we ignore the time complexity to check equality of tuples which will take only polynomial in $|A_j|$ time and will be dominated by the exponential terms. For the upstream-safety check, the number of $(x^+, y^+)$ pairs are at most $\Delta[|I_j \setminus H|]$, and to compute the distinct number of $x^+, y^+$ tuples from the pairs can be done in $O(\Delta^2[|I_j \setminus H|])$ time by a naive search; the time complexity can be improved by the use of a hash function. Hence the total time complexity is dominated by $2^{k_j} \times O(\Delta[|I_j \setminus H|]) \times O(\Delta^2[|I_j \setminus H|] + \Delta^2[|I_j \setminus H|]) = O(2^{k_j} \Delta^{3k}) = O(2^{4k_j})$. By doing a tighter analysis, the multiplicative factor in the exponents can be improved, however, we make the point here that the algorithm runs in time exponential in $k_j = |A_j|$.

Correctness. The following lemma proves the correctness of Algorithm 1.

Lemma 4. Algorithm 1 adds $H \subseteq A_j \to U_j$ if and only if $m_j$ is UD-safe with respect to $H$.

Proof. (if) Suppose $H$ is a UD-safe subset for $m_j$. Then $V$ is downstream-safe, i.e. for equivalent inputs with respect to the visible attributes $I_j \setminus H$, the projection of the output on the visible attributes $O_j \setminus H$ will be the same, so $H$ will pass the downstream-safety test.

Since $H$ is UD-safe, $H$ is also upstream-safe. Clearly, by definition, $n_1 \geq n_2$. Suppose $n_1 > n_2$. Then there are two $x^+_1$ and $x^+_2$ that pair with the same $y^+$. By construction, $x^+_1$ and $x^+_2$ (and all input tuples $x$ to $m_j$ that project on these two tuples) have different value on the visible input attributes $I_j \setminus H$, but they map to outputs $y$s that have the same value on visible output attributes $O_j \setminus H$. Then $H$ is not upstream-safe, which is a contradiction. Hence $n_1 = n_2$, and $H$ will also pass the test for upstream-safety and be included in $U_j$.

(only if) Suppose $H$ is not UD-safe, then it is either not upstream-safe or not downstream-safe. Suppose it is not downstream-safe. Then for at least one assignment $x^+$, the values of $y$ generated by the assignments $x^-$ will not be equivalent with respect to the visible output attributes, and the downstream-safety test will fail.
Algorithm 1 Algorithm to find all UD-safe solutions $U_j$ for a public module $m_j$

1: – Set $U_j = \emptyset$.
2: for every subset $H$ of $A_j$ do
3:   */Check if $H$ is downstream-safe */
4:   for every assignment $x^+$ of the visible input attributes in $I_j \setminus H$ do
5:     – Check if for every assignment $x^-$ of the hidden input attributes in $I_j \cap V$, whether the value of
6:       $\Pi_{O_j \setminus H}(m_j(x))$ is the same, where $\Pi_{I_j \setminus H}(x) = x^+$ and $\Pi_{I_j \cap H}(x) = x^-$
7:     if not then
8:       – $H$ is not downstream-safe.
9:       – Go to the next $H$.
10:   else
11:     – $H$ is downstream-safe.
12:     – Let $y^+ = \Pi_{O_j \setminus H}(m_j(x))$ = projection of all such tuples that have projection = $x^+$ on the visible input attributes
13:     – Label this set of input-output pairs $(x, m_j(x))$ by $(x^+, y^+)$. 
14: end if
15: */Check if $H$ is upstream-safe */
16: – Consider the pairs $(x^+, y^+)$ constructed above.
17: – Let $n_1$ be the number of distinct $x^+$ values, and let $n_2$ be the number of distinct $y^+$ values/
18: if $n_1 = n_2$ then
19: – $H$ is upstream-safe.
20: – Add $H$ to $U_j$.
21: else
22: – $H$ is not upstream-safe.
23: – Go to the next $H$.
24: end if
25: end for
26: end for
27: return The set of subsets $U_j$.

Suppose $H$ is downstream-safe but not upstream-safe. Then there are Then there are two $x_1^+$ and $x_2^+$ that pair with the same $y^+$. This makes $n_1 > n_2$, and the upstream-safety test will fail. □

B.2 Correctness of Optimal Algorithm for Chain Workflows

Recall that after renumbering the modules, $m_1, \cdots, m_k$ denote the modules in the public closure $C$ of a private module $m_i$. The following lemma shows that $Q[j, \ell]$ correctly stores the desired value: 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 $A_j \cap H^{j|\ell} = U_{j|\ell} \subseteq U_j$. Recall that we use the simplified notations $S$ for the safe subset $S_{i|\ell}$ of $m_i$, $C$ for public closure $C(S_{i|\ell})$, and $H$ for $H_i$.

**Lemma 5.** For $1 \leq j \leq k$, the entry $Q[j, \ell]$, $1 \leq \ell \leq p_j$, stores the minimum cost of the hidden attributes $H^{j|\ell}$ such that $\cup_{x=1}^j A_x \supseteq H^{j|\ell} \supseteq S$, $A_j \cap H^{j|\ell} = U_{j|\ell}$, and every module $m_x, 1 \leq x \leq j$ in the chain is UD-safe with respect to $A_x \setminus H^{j|\ell}$.
The following proposition will be useful to prove the lemma.

**Proposition 3.** For a public module \( m_j \), for two \( UD-safe \) hidden subsets \( U_1, U_2 \subseteq A_j \), if \( U_1 \cap O_j = U_2 \cap O_j \), then \( U_1 = U_2 \).

The proof of the proposition is simple, and therefore is omitted.

**Proof of Lemma 5**

**Proof.** We prove this by induction from \( j = 1 \) to \( k \). The base case follows by the definition of \( Q[1, \ell] \), for \( 1 \leq \ell \leq p_1 \). Here the requirements are \( A_1 \supseteq H^{1\ell} \supseteq S \), and \( H^{1\ell} = U_{1\ell} \). So we set the cost at \( Q[1, \ell] \) to \( c(U_{1\ell}) = c(H^{1\ell}) \), if \( U_{1\ell} \supseteq S \).

Suppose the hypothesis holds until \( j - 1 \), and consider \( j \). Let \( H^{j\ell} \) be the minimum solution s.t. \( A_j \cap H^{j\ell} = U_{j\ell} \) and satisfies the other conditions of the lemma.

First consider the case when there is no \( q \) such that \( U_{j-1, q} \cap O_{j-1} = U_{j, \ell} \cap I_j \), where we set the cost to be \( \infty \). If there is no such \( q \), i.e. for all \( q \leq p_{j-1} \), then clearly there cannot be any solution \( H^{j\ell} \) that contains \( U_{j, \ell} \) and also guarantees \( UD-safe \) properties of all \( x < j \) (in particular for \( x = j - 1 \)). In that case the cost of the solution is indeed \( \infty \).

Otherwise (when such a \( q \) exists), let us divide the cost of the solution \( c(H^{j\ell}) \) into two disjoint parts:

\[ c(H^{j\ell}) = c(H^{j\ell} \cap O_j) + c(H^{j\ell} \setminus O_j) \]

We argue that \( c(O_j \cap H^{j\ell}) = c(O_j \cap U_{j\ell}) \). \( A_j \cap H^{j\ell} = U_{j\ell} \). Then \( O_j \cap U_{j\ell} = O_j \cap A_j \cap H^{j\ell} = O_j \cap H^{j\ell} \), since \( O_j \subseteq A_j \). Hence \( c(O_j \cap H^{j\ell}) = c(O_j \cap U_{j\ell}) \). This accounts for the cost of the first part of \( Q[j, \ell] \).

Next we argue that \( c(H^{j\ell} \setminus O_j) \) is minimum cost \( Q[j-1, q] \), \( 1 \leq q \leq p_j \), where the minimum is over those \( q \) where \( U_{j-1, q} \cap O_{j-1} = U_{j, \ell} \cap I_j \). Due to the chain structure of \( C \), \( O_j \cap \bigcup_{i=1}^{j-1} A_i = \emptyset \), and \( O_j \cup \bigcup_{i=1}^{j-1} A_i = \bigcup_{i=1}^{j-1} A_i \). Since \( \bigcup_{i=1}^{j-1} A_i \supseteq H^{j\ell} \), \( H^{j\ell} \setminus O_j = H^{j\ell} \cap \bigcup_{i=1}^{j-1} A_i \).

Consider \( H' = H^{j\ell} \cap \bigcup_{i=1}^{j-1} A_i \). By definition of \( H^{j\ell} \), \( H' \) must satisfy the \( UD-safe \) requirement of all \( 1 \leq x \leq j - 1 \). Further, \( \bigcup_{i=1}^{j-1} A_i \supseteq H' \), \( A_j \cap H^{j\ell} = U_{j, \ell} \), hence \( U_{j, \ell} \subseteq H^{j\ell} \).

We are considering the case where there is a \( q \) such that

\[ U_{j-1, q} \cap O_{j-1} = U_{j, \ell} \cap I_j \quad (11) \]

Therefore

\[ U_{j-1, q} \cap O_{j-1} \subseteq U_{j, \ell} \subseteq H^{j\ell} \]

We claim that if \( q \) satisfies \((11)\), then \( A_{j-1} \cap H' = U_{j-1, q} \). Therefore, by induction hypothesis, \( Q[j-1, \ell] \) stores the minimum cost solution \( H' \) that includes \( U_{j-1, q} \), and part of the the optimal solution cost \( c(H^{j\ell} \setminus O_j) \) for \( m_j \) is the minimum value of such \( Q[j-1, q] \).

So it remains to show that \( A_{j-1} \cap H' = U_{j-1, q} \cap O_{j-1} = U_{j-1, y} \cap O_{j-1} = H^{j\ell} \cap O_{j-1} = H^{j\ell} \cap I_j = (A_j \cap U_{j, \ell}) \cap I_j \), i.e.

\[ U_{j-1, y} \cap O_{j-1} = U_{j, \ell} \cap I_j \quad (12) \]

From \((11)\) and \((12)\),

\[ U_{j-1, q} \cap O_{j-1} = U_{j-1, y} \cap O_{j-1} \]

since both \( U_{j-1, q}, U_{j-1, y} \subseteq U_{j-1} \), from Proposition 3 \( U_{j-1, q} = U_{j-1, y} \). This completes the proof of the lemma. \( \square \)
B.3 Optimal Algorithm for Tree Workflows

Here we prove Theorem 4 for tree workflows.

**Optimal algorithm for tree workflows** Similar to the algorithm for chain workflows, to obtain an algorithm of time polynomial in $L$ for tree workflows, for a given module $m_i$, we can go over all choices of safe subsets $S_{i\ell} \in S_i$ of $m_i$, compute the public-closure $C(S_{i\ell})$, and choose a minimal cost subset $H_i = H_i(S_{i\ell})$ that satisfies the $\text{UD-safe}$ property of all modules in the public-closure. Then, output, among them, a subset having the minimum cost. Consequently, it suffices to explain how, given a safe subset $S_{i\ell} \in S_i$, one can solve, in PTIME, the problem of finding a minimum cost hidden subset $H_i$ that satisfies the $\text{UD-safe}$ property of all modules in a subgraph formed by a given $C(S_{i\ell})$.

To simplify notations, the given safe subset $S_{i\ell}$ will be denoted below by $S$, the closure $C(S_{i\ell})$ will be denoted by $C$, and the output hidden subset $H_i$ will be denoted by $H$. Our PTIME algorithm uses dynamic programming to find the optimal $H$.

First note that since $C$ is the public-closure of (some) output attributes for a tree workflow, $C$ is a collection of trees all of which are rooted at the private module $m_i$. Let us consider the tree $T$ rooted at $m_i$ with subtrees in $C$, (note that $m_i$ can have private children that are not considered in $T$). Let $k$ be the number of modules in $T$, and the modules in $T$ be renumbered as $m_i,m_1,\cdots,m_k$, where the private module $m_i$ is the root, and the rest are public modules.

Now we solve the problem by dynamic programming as follows. Let $Q$ be an $k \times L$ two-dimensional array, where $Q[j,\ell]$, $1 \leq j \leq k$, $1 \leq \ell \leq p_j$ denotes the cost of minimum cost hidden subset $H^{j\ell}$ that (i) satisfies the $\text{UD-safe}$ condition for all public modules in the subtree of $T$ rooted at $m_j$, that we denote by $T_j$; and, (ii) $H^{j\ell} \cap A_j = U_{j\ell}$. (recall that $I_j O_j , A_j$ is the set of input, output and all attributes of $m_j$ respectively); the actual solution can be stored easily by standard argument. The algorithm is described below.

- **Initialization for leaf nodes.** The initialization step handles all leaf nodes $m_j$ in $T$. For a leaf node $m_j$, $1 \leq \ell \leq p_j$,

$$Q[j,\ell] = c(U_{j,\ell})$$

- **Internal nodes.** The internal nodes are considered in a bottom-up fashion (by a post-order traversal), and $Q[j,\ell]$ is computed for a node $m_j$ after its children are processed.

For an internal node $m_j$, let $m_i, \cdots, m_k$ be its children in $T$. Then for $1 \leq \ell \leq p_j$,

1. Consider $\text{UD-safe}$ subset $U_{j,\ell}$.
2. For $y = 1$ to $x$, let $U^y = U_{j,\ell} \cap I_i$. Since there is no data sharing, $U^y$-s are disjoint
3. For $y = 1$ to $x$,

$$k^y = \arg\min_k Q[i,y,k] \quad \text{where the minimum is over}$$
$$1 \leq k \leq p_i \quad \text{s.t.} \quad U_{i,k} \cap I_i = U^y$$
$$= \bot \quad \text{(undefined), \quad if there is no such } k$$

4. $Q[j,\ell]$ is computed as follows.

$$Q[j,\ell] = \infty \quad \text{if } \exists y, 1 \leq y \leq x, \quad k^y = \bot$$
$$= c(I_j \cap U_{j,\ell}) + \sum_{y=1}^{x} Q[i,y,k^y] \quad \text{(otherwise)}$$
- **Final solution for \( S \).** Now consider the private module \( m_i \) that is the root of \( T \). Recall that we fixed a safe solution \( S \) of \( m_i \) for doing the analysis. Let \( m_i, \ldots, m_i \) be the children of \( m_i \) in \( T \) (which are public modules). Similar to the step before, we consider the min-cost solutions of its children which exactly match the hidden subset \( S \) of \( m_i \).

1. Consider safe subset \( S \) of \( m_i \).
2. For \( y = 1 \) to \( x \), let \( S^y = S \cap I_i \). Since there is no data sharing, again, \( S^y \)'s are disjoint
3. For \( y = 1 \) to \( x \),

\[
k^y = \arg\min_k Q[i_y,k] \quad \text{where the minimum is over} \quad 1 \leq k \leq p_i \quad \text{s.t.} \quad U_{i,k} \cap I_i \supseteq S^y
\]

\[
= \bot \quad \text{(undefined)}, \quad \text{if there is no such} \ k
\]

4. The cost of the optimal \( H \) (let us denote that by \( c^* \)) is computed as follows.

\[
c^* = \infty \quad \text{if} \ \exists y, 1 \leq y \leq x, \ k^y = \bot
\]

\[
= \sum_{y=1}^x Q[i_y,k^y] \quad \text{(otherwise)}
\]

It is not hard to see that the trivial solution of \( \text{UD-safe} \) subsets that include all attributes of the modules gives a finite cost solution by the above algorithm.

Lemma 6 stated and proved below shows that \( Q[j, \ell] \) correctly stores the desired value. Given this lemma, the correctness of the algorithm easily follows. For hidden subset \( H \supseteq S \) in the closure, for every public child \( m_i \) of \( m_i \), \( H \cap I_i \supseteq S \cap I_i = S^y \). Further, each such \( m_i \) has to be \( \text{UD-safe} \) with respect to \( H^{j,\ell} \).

In other words, for each \( m_i \), \( H \cap I_i \) must equal \( U_{i,k} \cap I_i \) for some \( 1 \leq k \leq p_i \). The last step in our algorithm (that computes \( c^* \)) tries to find such a \( k \) that has the minimum cost \( Q[i_y,k^y] \), and the total cost \( c^* \) of \( H \) is \( \sum m_j Q[i_y,k^y] \) where the sum is over all children of \( m_i \) in the tree \( T \) (the trees rooted at \( m_i \) are disjoint, so the optimal cost \( c^* \) is sum of those costs). This proves Theorem 4 for tree workflows.

**\( Q[j, \ell] \) stores correct values.** The following lemma shows that the algorithm stores correct values in \( Q[j, \ell] \) for all public modules \( m_j \) in the closure \( C \).

**Lemma 6.** For \( 1 \leq j \leq k \), let \( T_j \) be the subtree rooted at \( m_j \) and let \( \text{Att}_j = \bigcup_{m_j \in T_j} A_q \). The entry \( Q[j, \ell] \), \( 1 \leq \ell \leq p_j \), stores the minimum cost of the hidden attributes \( H^{j,\ell} \subseteq \text{Att}_j \) such that \( A_j \cap H^{j,\ell} = U_{j,\ell} \), and every module \( m_q \in T_j \) is \( \text{UD-safe} \) with respect to \( A_q \cap H^{j,\ell} \).

To complete the proof of Theorem 4 for tree workflows, we need to prove Lemma 6, that we prove next.

**Proof.** We prove the lemma by an induction on all nodes at depth \( h = H \) down to \( 1 \) of the tree \( T \), where depth \( H \) contains all leaf nodes and depth \( 1 \) contains the children of the root \( m_i \) (which is at depth 0).

First consider any leaf node \( m_j \) at height \( H \). Then \( T_j \) contains only \( m_j \) and \( \text{Att}_j = A_j \). For any \( 1 \leq \ell \leq p_j \), since \( \text{Att}_j = A_j \supseteq H^{j,\ell} \) and \( A_j \cap H^{j,\ell} = U_{j,\ell} \). In this case \( H^{j,\ell} \) is unique and \( Q[j, \ell] \) correctly stores \( c(U_{j,\ell}) = c(H^{j,\ell}) \).

Suppose the induction holds for all nodes up to height \( h + 1 \), and consider a node \( m_j \) at height \( h \). Let \( m_i, \ldots, m_i \) be the children of \( m_j \) which are at height \( h + 1 \). Let \( H^{j,\ell} \) be the min-cost solution, which is partitioned into two disjoint component:
Let us partition the subset $I_j$ as stated in Theorem 4 by a reduction from 3SAT.

**B.4 Proof of NP-hardness for DAG Workflows**

First we argue that $c(H^j \cap I_j) = c(U_{j, i})$. $A_j \cap H^j = U^j$. Then $I_j \cap U^j = I_j \cap A_j \cap H^j = I_j \cap H^j$, since $I_j \subseteq A_j$. Hence $c(I_j \cap H^j) = c(I_j \cap U^j)$. This accounts for the cost of the first part of $Q[j, \ell]$.

Next we analyze the cost $c(H^j \setminus I_j)$. This cost comes from the subtrees $T_i, \ldots, T_h$, which are disjoint due to the tree structure and absence of data-sharing. Let us partition the subset $H^j \setminus I_j$ into disjoint parts $(H^j \setminus I_j) \cap \text{Att}_y$, $1 \leq y \leq x$. Below we prove that $c((H^j \setminus I_j) \cap \text{Att}_y) = Q[i, k^y]$, $1 \leq y \leq x$, where $k^y$ is computed as in the algorithm. This will complete the proof of the lemma.

To see this, let $H' = (H^j \setminus I_j) \cap \text{Att}_y$. Clearly, $\text{Att}_y \supseteq H'$. Every $m_q \in T_j$ is UD-safe with respect to $A_q \cap H^j$. If also $m_q \in T_i$, then $A_q \cap H' = A_q \cap H^j$, and therefore all $m_q \in T_i$ are also UD-safe with respect to $H'$. In particular, $m_q$ is UD-safe with respect to $H'$, and therefore $A_q \cap H' = U_{i, k^y}$ for some $k^y$, since $U_{i, k^y}$ was chosen as the UD-safe set by our algorithm.

Finally, we argue that $c(H') = c(H_{i, k^y})$, where $H_{i, k^y}$ is the min-cost solution for $m_i$ among all such subsets. This follows from our induction hypothesis, since $m_i$ is a node at depth $h + 1$. Therefore, $c(H') = c(H_{i, k^y}) = Q[i, k^y]$, i.e. $c((H^j \setminus I_j) \cap \text{Att}_y) = Q[i, k^y]$ as desired. This proves the lemma. 

**B.4 Proof of NP-hardness for DAG Workflows**

Here we prove NP-hardness for arbitrary DAG workflows as stated in Theorem 4 by a reduction from 3SAT.

Given a CNF formula $\psi$ on $n$ variables $z_1, \ldots, z_n$ and $m$ clauses $\psi_1, \ldots, \psi_m$, we construct a graph as shown in Figure 7. Let variable $z_i$ occurs in $m_i$ different clauses (as positive or negative literals). In the
figure, the module $p_0$ is the single-private module ($m_0$), having a single output attribute $a$. The rest of the modules are the public modules in the public-closure $C(\{a\})$.

For every variable $z_i$, we create $m_i + 2$ nodes: $p_i, y_i$ and $x_{i,1}, \cdots, x_{i,m_i}$. For every clause $\psi_j$, we create 2 modules $C_j$ and $f_j$.

The edge connections are as follows:

1. $p_0$ sends its single output $a$ to $p_1$.
2. For every $i = 1$ to $n − 1$, $p_i$ has two outputs; one is sent to $p_{i+1}$ and the other is sent to $y_i$. $p_n$ sends its single output to $y_n$.
3. Each $y_i$, $i = 1$ to $n$, sends two outputs to $x_{i,1}$. The blue outgoing edge from $y_i$ denotes positive assignment of the variable $z_i$, whereas the red edge denotes negative assignment of the variable $z_i$.
4. Each $x_{i,j}$, $i = 1$ to $n$, $j = 1$ to $m_i − 1$, sends two outputs (blue and red) to $x_{i,j+1}$. In addition, if $x_{i,j}$, $i = 1$ to $n$, $j = 1$ to $m_i$ sends a blue (resp. red) edge to clause node $C_k$ if the variable $z_i$ is a positive (resp. negative) in the clause $C_k$ (and $C_k$ is the $j$-th such clause containing $z_i$).
5. Each $C_j$, $j = 1$ to $m$, sends its single output to $f_j$.
6. Each $f_j$, $j = 1$ to $m − 1$, sends its single output to $f_{j+1}$, $f_m$ outputs the single final output.

The UD-safe sets are defined as follows:

1. For every $i = 1$ to $n − 1$, $p_i$ has a single UD-safe set: hide all its inputs and outputs.
2. Each $y_i$, $i = 1$ to $n$, has three UD-safe choices: (1) hide its unique input and blue output, (2) hide its unique input and red output, (3) hide its single input and both blue and red outputs.
3. Each $x_{i,j}$, $i = 1$ to $n$, $j = 1$ to $m_i$, has three choices: (1) hide blue input and all blue outputs, (2) hide red input and all red outputs, (3) hide all inputs and all outputs.
4. Each $C_j$, $j = 1$ to $m$, has choices: hide the single output and at least of the three inputs.
5. Each $f_j$, $j = 1$ to $m$, has the single choice: hide all its inputs and outputs.

Cost. The outputs from $y_i$, $i = 1$ to $n$ has unit cost, the cost of the other attributes is 0. The following lemma proves correctness of the construction.

**Lemma 7.** There is a solution of single-module problem of cost $= n$ if and only if the 3SAT formula $\psi$ is satisfiable.

**Proof.** (if) Suppose the 3SAT formula is satisfiable, so there is an assignment of the variables $z_i$ that makes $\Psi$ true. If $z_i$ is set to TRUE (resp FALSE), choose the blue (resp. red) outgoing edge from $y_i$. Then choose the other edges accordingly: (1) choose outgoing edge from $p_0$, (2) choose all input and outputs of $p_i$, $i = 1$ to $n$; (3) if blue (resp. red) input of $x_{i,j}$ is chosen, all its blue (resp. red) outputs are chosen; and, (4) all inputs and outputs of $f_j$ are chosen. Clearly, all these are UD-safe sets by construction.

So we have to only argue about the clause nodes $C_j$. Since $\psi$ is satisfied by the given assignment, there is a literal $z_i \in C_j$ (positive or negative), whose assignment makes it true. Hence at least one of the inputs to $C_j$ will be chosen. So the UD-safe requirements of all the UD-safe clauses are satisfied. The total cost
of the solution is \( n \) since exactly one output of the \( y_i \) nodes, \( i = 1 \) to \( n \), have been chosen.

(only if) Suppose there is a solution to the single-module problem of cost \( n \). Then each \( y_i \) can choose exactly one output (at least one output has to be chosen to satisfy UD-safe property for each \( y_i \), and more than one output cannot be chosen as the cost is \( n \)). If \( y_i \) chooses blue (resp. red) output, this forces the \( x_{i,j} \) nodes to select the corresponding blue (resp. red) inputs and outputs. No \( x_{i,j} \) can choose the UD-safe option of selecting all its inputs and outputs as in that case finally \( y_i \) be forced to select both outputs which will exceed the cost. Since \( C_j \) satisfies UD-safe condition, this in turn forces each \( C_j \) to select the corresponding blue (resp. red) inputs.

If the blue (resp. red) output of \( y_i \) is chosen, the variable is set to TRUE (resp. FALSE). By the above argument, at least one such red or blue input will be chosen as input to each \( C_j \), that satisfies the corresponding clause \( \psi_j \).

\[ \square \]

### C General Workflows

In this section we discuss the privacy theorem for general workflows as outlined in Section 5. First we define directed-path and downward-closure as follows (similar to public path and public-closure).

**Definition 12.** A module \( m_1 \) has a directed path to another module \( m_2 \), if there are modules \( m_{i_1}, m_{i_2}, \ldots, m_{i_j} \) such that \( m_{i_1} = m_1, m_{i_j} = m_2 \), and for all \( 1 \leq k < j \), \( O_{i_k} \cap I_{i_{k+1}} \neq \emptyset \).

An attribute \( a \in A \) has a directed path from to module \( m_j \), if there is a module \( m_k \) such that \( a \in I_k \) and \( m_k \) has a directed path to \( m_j \).

**Definition 13.** Given a private module \( m_i \) and a set of hidden output attributes \( h_i \subseteq O_i \) of \( m_i \), the downward-closure of \( m_i \) with respect to \( h_i \), denoted by \( D(h_i) \), is the set of modules \( m_j \) (both private and public) such that there is a directed path from some attribute \( a \in h_i \) to \( m_j \).

Also recall downstream-safety (D-safety) defined in Definition 8 which says that for equivalent inputs to a module with respect to hidden attributes, the outputs must be equivalent. We prove the following theorem in this section:

**Theorem 5. (Privacy Theorem for General workflows)** Let \( W \) be any 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 \( \Gamma \)-standalone-privacy of \( m_i \), (ii) each private and public module \( m_j \) in the downward-closure \( D(h_j) \) is D-safe with respect to \( A_j \cap H_j \), and (iii) \( H_i \subseteq O_i \cup \bigcup_{j: m_j \in D(h_i)} A_j \). Then the workflow \( W \) is \( \Gamma \)-private with respect to \( H = \bigcup_{i} H_i \) is private to \( H_i \).

In the proof of Theorem 2 from Lemma 1, we used the fact that for single-predecessor workflows, for two distinct private modules \( m_i, m_k \), the public-closures and the hidden subsets \( H_i, H_j \) are disjoint. However, it is not hard to see that this is not the case for general workflows, where the downward-closure and the subsets \( H_i \) may overlap. Further, the D-safe property is not monotone (hiding more output attributes will maintain the D-safe property, but hiding more input attributes may destroy the D-safe property). So we need to argue that the D-safe property is maintained when we take union of \( H_i \) sets in the workflow which is formalized by the following lemma.

**Lemma 8.** If a module \( m_j \) is D-safe with respect to sets \( H_1, H_2 \subseteq A_j \), then \( m_j \) is D-safe with respect to \( H = H_1 \cup H_2 \).
Given two equivalent inputs $x_1 \equiv_H x_2$ with respect to $H = H_1 \cup H_2$, we have to show that their outputs are equivalent: $m_j(x_1) \equiv_H m_j(x_2)$. Even if $x_1, x_2$ are equivalent with respect to $H$, they may not be equivalent with respect to $H_1$ or $H_2$. In the proof we construct a new tuple $x_3$ such that $x_1 \equiv_{H_1} x_3$, and $x_2 \equiv_{H_2} x_3$. Then using the $D$-$\text{safe}$ properties of $H_1$ and $H_2$, we show that $m_j(x_1) \equiv_H m_j(x_3) \equiv_V m_j(x_2)$. The formal proof is given below.

**Proof.** Let $H = H_1 \cup H_2$. Let $x_1$ and $x_2$ be two input tuples to $m_j$ such that $x_1 \equiv_H x_2$. i.e.

$$\Pi_{I_j \setminus H}(x_1) = \Pi_{I_j \setminus H}(x_2) \quad (13)$$

For $a \in I_j$, let $x_3[a]$ denote the value of $a$-th attribute of $x_3$ (similarly $x_1[a], x_2[a]$). From $(13)$, for $a \in I_j \setminus H$, $x_1[a] = x_2[a]$. Let us define a tuple $x_3$ as follows on four disjoint subsets of $I_j$:

$$x_3[a] = \begin{cases} x_1[a] & \text{if } a \in I_j \cap (H_1 \cap H_2) \\ x_2[a] & \text{if } a \in I_j \cap (H_2 \setminus H_1) \\ x_1[a] = x_2[a] & \text{if } a \in I_j \setminus H \end{cases}$$

For instance, on attribute set $I_j = \langle a_1, \ldots, a_5 \rangle$, let $x_1 = \langle 2, 3, 6, 7 \rangle$, $x_2 = \langle 4, 5, 6, 7 \rangle$, $H_1 = \{a_1, a_2\}$ and $H_2 = \{a_2, a_3\}$, $H = \{a_1, a_2, a_3\}$ (in $x_1, x_2$, the hidden attribute values in $H$ are underlined). Then $x_3 = \langle 4, 3, 2, 6, 7 \rangle$.

(1) First we claim that, $x_1 \equiv_{H_1} x_3$, or,

$$\Pi_{I_j \setminus H_1}(x_1) = \Pi_{I_j \setminus H_1}(x_3) \quad (14)$$

Partition $I_j \setminus H_1$ into two disjoint subsets, $I_j \cap (H_2 \setminus H_1)$, and, $I_j \setminus (H_1 \cup H_2) = I_j \setminus H$. From the definition of $x_3$, for all $a \in I_j \cap (H_2 \setminus H_1)$ and all $a \in I_j \setminus H$, $x_1[a] = x_3[a]$. This shows $(14)$.

(2) Next we claim that, $x_2 \equiv_{H_2} x_3$, or,

$$\Pi_{I_j \setminus H_2}(x_2) = \Pi_{I_j \setminus H_2}(x_3) \quad (15)$$

Again partition $I_j \setminus H_2$ into two disjoint subsets, $I_j \cap (H_1 \setminus H_2)$, and, $I_j \setminus (H_1 \cup H_2) = I_j \setminus H$. From the definition of $x_3$, for all $a \in I_j \cap (H_1 \setminus H_2)$ and all $a \in I_j \setminus H$, $x_2[a] = x_3[a]$. This shows $(15)$. $(14)$ and $(15)$ can also be verified from the above example.

(3) Now by the condition stated in the lemma, $m_j$ is $D$-$\text{safe}$ with respect to $H_1$ and $H_2$. Therefore, using $(14)$ and $(15)$, $m_j(x_1) \equiv_{H_1} m_j(x_3)$ and $m_j(x_2) \equiv_{H_2} m_j(x_3)$ or,

$$\Pi_{O_j \setminus H_1}(m_j(x_1)) = \Pi_{O_j \setminus H_1}(m_j(x_3)) \quad (16)$$

and

$$\Pi_{O_j \setminus H_2}(m_j(x_2)) = \Pi_{O_j \setminus H_2}(m_j(x_3)) \quad (17)$$

Since $O_j \setminus H = O_j \setminus (H_1 \cup H_2) \subseteq O_j \setminus H_1$, from $(16)$

$$\Pi_{O_j \setminus H}(m_j(x_1)) = \Pi_{O_j \setminus H}(m_j(x_3)) \quad (18)$$
Similarly, $O_j \setminus H \subseteq O_j \setminus H_2$, from (17)

$$\Pi_{O_j \setminus H}(m_j(x_2)) = \Pi_{O_j \setminus H}(m_j(x_3))$$

From (18) and (19),

$$\Pi_{O_j \setminus H}(m_j(x_1)) = \Pi_{O_j \setminus H}(m_j(x_2))$$

In other words, the output tuples $m_j(x_1), m_j(x_2)$, that are defined on attribute set $O_j$,

$$m_j(x_1) \equiv_H m_j(x_2)$$

Since we started with two arbitrary input tuples $x_1 = x_2$, this shows that for all equivalent input tuples the outputs are also equivalent. In other words, $m_j$ is $D$-safe with respect to $H = H_1 \cup H_2$.

Along with this lemma, two other simple observations will be useful.

**Observation 3.**
1. Any module $m_j$ is $D$-safe with respect to $\emptyset$ (hiding nothing maintains downstream-safety property).
2. If $m_j$ is $D$-safe with respect to $H$, and if $H'$ is such that $H \subseteq H'$, but $I_j \setminus H' = I_j \setminus H$, then $m_j$ is also $D$-safe with respect to $H'$ (hiding more output attributes maintains downstream-safety property).

### C.1 Main Lemma for Privacy Theorem for General Workflows

The following lemma is the crucial component in the proof of Theorem 5 and is analogous to Lemma 1 for single-predecessor workflows.

**Lemma 9.** Consider a standalone private module $m_i$, a set of hidden attributes $h_i$, any input $x$ to $m_i$, and any candidate output $y \in \text{OUT}_{x,m_i,h_i}$ of $x$. Then $y \in \text{OUT}_{x,W,H}$ when $m_i$ belongs to an arbitrary (general) 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 downward-closure $D(h_i)$ is $D$-safe with respect to $A_j \cap H_i$.

**Proof.** We fix a module $m_i$, an input $x$ to $m_i$, a set of safe hidden attributes $h_i$, and a candidate output $y \in \text{OUT}_{x,m_i,h_i}$ for $x$. For simplicity, let us refer to the set of modules in $D(h_i)$ by $D$. We will show that $y \in \text{OUT}_{x,W,H}$ where the hidden attributes $H_i$ satisfies the conditions in the lemma. In the proof, we show the existence of a possible world $R' \in \text{worlds}(R, H)$, such that if $\Pi_{l_i}(t) = x$ for some $t \in R'$, then $\Pi_{O_i}(t) = y$. Since $y \in \text{OUT}_{x,m_i,h_i}$, by Lemma 3, $y \equiv_{h_i} z$ where $z = m_i(x)$.

We will use the \text{FLIP} function used in the proof of Lemma 1 (see Appendix A.2). We redefine the module $m_l$ to $\tilde{m}_l$ as follows. For an input $u$ to $m_i$, $\tilde{m}_i(u) = \text{FLIP}_{x,z}(m_i(u))$. All other public and private modules are unchanged, $\tilde{m}_j = m_j$. The required possible world $R'$ is obtained by taking the join of the standalone relations of these $\tilde{m}_j$, $j \in [n]$.

First note that by the definition of $\tilde{m}_i$, $\tilde{m}_i(x) = y$ (since $\tilde{m}_i(x) = \text{FLIP}_{x,z}(m_i(x)) = \text{FLIP}_{x,z}(z) = y$, from Observation 1). Hence if $\Pi_{l_i}(t) = x$ for some $t \in R'$, then $\Pi_{O_i}(t) = y$.

Next we argue that $R' \in \text{worlds}(R, H_i)$. Since $R'$ is the join of the standalone relations for modules $\tilde{m}_j$, $R'$ maintains all functional dependencies $I_j \rightarrow O_j$. Also none of the public modules are unchanged, hence for any public module $m_j$ and any tuple $t$ in $R'$, $\Pi_{O_j}(t) = m_j(\Pi_{l_j}(t))$. So we only need to show that the projection of $R$ and $R'$ on the visible attributes are the same.

Let us assume, wlog. that the modules are numbered in topologically sorted order. Let $l_0$ be the initial input attributes to the workflow, and let $p$ be a tuple defined on $l_0$. There are two unique tuples $t \in R$ and...
t' ∈ R' such that \( \Pi_I(t) = \Pi_I(t') = p \). Note that any intermediate or final attribute \( a ∈ A \setminus I_0 \) belongs to \( O_j \), for a unique \( j ∈ [1, n] \) (since for \( j ≠ l \), \( O_j ∩ O_l = \emptyset \)). So it suffices to show that \( t, t' \) projected on \( O_j \) are equivalent with respect to visible attributes for all module \( j, j = 1 \) to \( n \).

Let \( c_{j,m}, c_{j,m̂} \) be the values of input attributes \( I_j \) and \( d_{j,m}, d_{j,m̂} \) be the values of output attributes \( O_j \) of module \( m_j \), in \( t ∈ R \) and \( t' ∈ R' \) respectively on initial input attributes \( p \) (i.e. \( c_{j,m} = \Pi_I(t), c_{j,m̂} = \Pi_I(t') \), \( d_{j,m} = \Pi_O(t) \) and \( d_{j,m̂} = \Pi_O(t') \)). We prove by induction on \( j = 1 \) to \( n \) that

\[
d_{j,m̂} = \Pi_{H_j} d_{j,m} \quad \text{if } j = i \text{ or } m_j ∈ D \tag{22}
\]

\[
d_{j,m̂} = d_{j,m} \quad \text{otherwise} \quad \text{ (23)}
\]

If the above is true for all \( j \), then \( \Pi_{O_j}(t) = \Pi_{H_j} \Pi_{O_j}(t) \), along with the fact that the initial inputs \( p \) are the same, this implies that \( t ≡_{H_j} t' \).

**Proof of (22) and (23).** The base case follows for \( j = 1 \). If \( m_1 ≠ m_i \) (\( m_j \) can be public or private), then \( I_1 ∩ O_1 = \emptyset \), so for all input \( u \), \( \hat{m}_j(u) = m_j(\text{FLIP}_{y,z}(u)) = m_j(u) \). Since the inputs \( c_{1,m̂} = c_{1,m} \) (both projections of initial input \( p \) on \( I_1 \)), the outputs \( d_{1,m̂} = d_{1,m} \). This shows (23). If \( m_1 = m_i \), the inputs are the same, and by definition of \( \hat{m}_1, d_{1,m̂} = \hat{m}_1(c_{1,m̂}) = \text{FLIP}_{y,z}(m_1(c_{1,m})) = \text{FLIP}_{y,z}(d_{1,m}). \) Since \( y, z \) only differ in the hidden attributes, by the definition of the FLIP function \( d_{1,m̂} = \Pi_{H_1} d_{1,m}. \) This shows (22). Note that the module \( m_1 \) cannot belong to \( D \) since then it will have predecessor \( m_i \) and cannot be the first module in topological order.

Suppose the hypothesis holds until \( j - 1 \), consider \( m_j \). There will be three cases to consider.

(i) If \( j = i \), for all predecessors \( m_k \) of \( m_i \) \( (O_k ∩ I_i ≠ \emptyset), k ≠ i \) and \( m_k ∉ D \), since the workflow is a DAG. Therefore from (23), using the induction hypothesis, \( c_{i,m̂} = c_{i,m}. \) Hence \( d_{i,m̂} = \hat{m}_i(c_{i,m̂}) = \text{FLIP}_{y,z}(m_i(c_{i,m})) = \text{FLIP}_{y,z}(d_{i,m}). \) Again, \( y, z \) are equivalent with respect to \( H_i \), so \( d_{i,m̂} = d_{i,m} \). This shows (22) in the inductive step.

(ii) If \( j ≠ i \) \((\hat{m}_j = m_j) \) and \( m_j ∉ D \), then \( m_j \) does not get any of its inputs from any module in \( D \), or any hidden attributes from \( m_i \) (then by the definition of \( D, m_j ∈ D \)). Using IH, from (23) and from (22), using the fact that \( y, z \) are equivalent on visible attributes, \( c_{j,m̂} = c_{j,m}. \) Then \( d_{j,m̂} = m_j(c_{j,m̂}) = m_j(c_{j,m}) = d_{j,m} \). This shows (23) in the inductive step.

(iii) If \( j ≠ i \), but \( m_j ∈ D \), \( m_j \) can get all its inputs either from \( m_i \), from other modules in \( D \), or from modules not in \( D \). Using the IH from (22) and (23), \( c_{j,m̂} ≡_{H_j} c_{j,m}. \) Since \( m_j ∈ D \), by the condition of the lemma, \( m_j \) is \( D\text{-safe} \) with respect to \( H_i \). Therefore the corresponding outputs \( d_{j,m̂} = m_j(c_{j,m̂}) \) and \( d_{j,m} = m_j(c_{j,m}) \) are equivalent, or \( d_{j,m̂} = d_{j,m} \). This again shows (22) in the inductive step.

Hence the IH holds for all \( j = 1 \) to \( n \) and this completes the proof of the lemma.

**C.2 Proof of Theorem 5**

Finally, we prove Theorem 5 using Lemmas 8 and 9.

**Proof of Theorem 5** We argue that if \( H_i \) satisfies the conditions in Theorem 5 then \( H'_i = \bigcup_{c_{m_i}} \) is private \( H_i \) satisfies the conditions in Lemma 9. The first two conditions are easily satisfied by \( H'_i \): (i) \( h_i ⊆ H_i ⊆ H'_i \) and (ii) \( h_i ⊆ O_i \). So we need to show (iii), i.e. all modules in the downward-closure \( D(h_i) \) are \( D\text{-safe} \) with respect to \( A_j ∩ H'_1 \).
From the conditions in the theorem, each module \( m_j \in D(h_i) \) is \( D \)-safe with respect to \( A_j \cap H_i \). We show that for any other private module \( m_k \neq m_i, m_j \) is also \( D \)-safe with respect to \( A_j \cap H_k \). There may be three such cases as discussed below.

**Case-I:** If \( m_j \in D(h_k) \), by the \( D \)-safety conditions in the theorem, \( m_j \) is \( D \)-safe with respect to \( A_j \cap H_k \).

**Case-II:** If \( m_j \notin D(h_k) \) and \( m_j \neq m_k \), for any private module \( m_k \neq m_i \), \( A_j \cap H_k = \emptyset \) (since \( H_k \subseteq O_k \cup \bigcup_{\ell \in D(h_i)} A_\ell \) from the theorem). From Observation 3, \( m_j \) is \( D \)-safe with respect to \( A_j \cap H_k \).

**Case-III:** If \( m_j \notin D(h_k) \) but \( m_j = m_k \) (or \( j = k \)), then \( H_k \cap A_j \subseteq O_j \) (again since \( H_k \subseteq O_k \cup \bigcup_{\ell \in D(h_i)} A_\ell \) and \( O_k = O_j \)). From Observation 3, \( m_j \) is \( D \)-safe with respect to \( A_j \cap H_k \). This is because \( H_k \cap A_j \subseteq O_j \), since \( O_j \cap I_j = \emptyset \), \( I_j \cap (A_j \cap H_k) = \emptyset \). Hence from the same observation, \( m_j \) is \( D \)-safe with respect to \( A_j \cap H_k \).

Hence \( m_j \) is \( D \)-safe with respect to \( A_j \cap H_i \) and for all private modules \( m_k, m_k \neq m_i, m_j \) is \( D \)-safe with respect to \( A_j \cap H_k \). By Lemma 6, then \( m_j \) is \( D \)-safe with respect to \((A_j \cap H_i) \cup (A_j \cap H_k) = A_j \cap (H_i \cup H_k) \). By a simple induction on all private modules \( m_k, m_j \) is \( D \)-safe with respect to \( A_j \cap (\bigcup_{k \text{ private}} H_k) = A_j \cap H'_i \). Hence \( H'_i \) satisfies the conditions stated in the lemma. The rest of the proof follows by the same argument as in the proof of Theorem 2.