Preserving Module Privacy in Workflow Provenance*

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

Sanjeev Khanna
University of Pennsylvania
sanjeev@cis.upenn.edu

Debmalya Panigrahi
CSAIL, MIT
debmalya@mit.edu

Sudeepa Roy
University of Pennsylvania
sudeepa@cis.upenn.edu

Technical Report MS-CIS-10-22
Department of Computer and Information Science
University of Pennsylvania

May 28, 2010

Abstract

We study the problem of providing workflow data provenance without revealing the functionality of any module. We develop a model that formalizes the notion of privacy of modules embedded in a workflow structure as a natural extension of privacy of standalone modules. Our model shows that by hiding a small amount of carefully chosen data, one can ensure privacy of all modules over an unbounded number of executions. The problem of identifying the smallest possible amount of such data is NP-hard, and in the full generality of our model it is in fact even hard to get a good approximation. However, we are able to design good approximation algorithms for optimizing the amount of hidden data when either the privacy model is slighted restricted or there is bounded sharing of data items among various modules.

*This research was supported in part by NSF grant IIS-0803524.

1 Introduction

Scientific workflow systems (e.g. myGrid/Taverna, Kepler, VisTrails, Pegasus (see [25, 7, 17])) are extensively used for specifying and executing in-silico experiments or processes. Such a system can be thought of as a program, producing a set of *final output data* from a set of *initial input data* in an execution. Internally, a workflow system comprises a set of *modules* (i.e. programs) with input and output data ports. An output port of a module is connected to a set of input ports of other modules (called *internal connections*) and/or has an *external connection* if it produces *final output data*. On the other hand, an input port of a module either receives input data from the output port of another module on an internal connection or has an external connection on which it receives initial input data. In addition to initial input data and final output data, an execution of a workflow also produces *intermediate data*, i.e. data produced by a module at an output port that does not have an external connection. Intermediate data produced by a module is passed along internal connections to input ports of other modules. *Parameters* can also be specified for modules and, for uniformity, are considered to be initial inputs as well. In this paper, we assume that the connections form an *acyclic* graph on the modules, which is common in many of the aforementioned workflow systems [17, 25].

Once specified, a workflow may be executed multiple times using different initial input data and generating much intermediate and final output data. To keep track of how the different output data were generated and ensure their reproducibility, many workflow systems are beginning to provide tools to capture and manage *data provenance*. Data provenance is defined as the set of dependencies between data (initial input, intermediate and final output) pertaining to an execution, including the module executions that produced them.

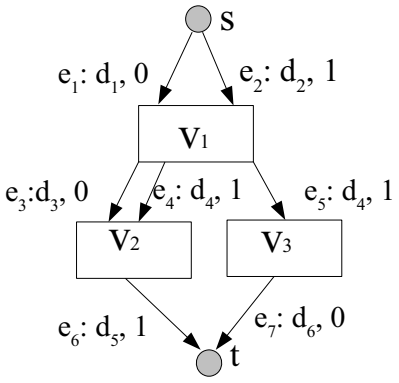

Figure 1: Example of a workflow execution

A simple example of a workflow is shown in Figure 1 which (for simplicity) uses boolean data and functions. There are two input data bits d_1 and d_2, and three modules v_1, v_2 and v_3 with underlying functions f_1, f_2 and f_3 defined as follows: f_1 computes two intermediate data bits d_3 and d_4, where $d_3 = d_1$ and $d_4 = d_1 \vee d_2$. f_2 produces one output data bit d_5 which is 1 iff $d_3 \neq d_4$, while the other output data bit $d_6 = 1 - d_4$ is produced by f_3. A source node s distributes initial input data to input ports (along e_1 and e_2), and a sink node t collects final output data from the output ports (along e_6 and e_7), corresponding to external connections. Note that the data bit d_4 produced by v_1 acts as input to both v_2 and v_3; in general, a data output by a module can act as input to multiple modules as well as (possibly) be a final output data. We call this phenomenon *data sharing*. Also, observe that a workflow may be a multigraph: in this example, v_1 sends two data bits d_3 and d_4 to v_2.

Figure 1 also shows a sample execution of the workflow when $d_1 = 0$ and $d_2 = 1$ are the initial inputs; then $d_3 = 0$, $d_4 = 1$, $d_5 = 1$ and $d_6 = 0$. In this execution, the provenance of $d_6 = 0$ includes data values $d_1 = 0, d_2 = 1$ and $d_4 = 1$, and modules v_1 and v_3. On the other hand, the provenance of $d_5 = 1$ includes data values $d_1 = 0, d_2 = 1, d_3 = 0$ and $d_4 = 1$, and modules v_1 and v_2.

Although provenance is important, if information about all intermediate data is repeatedly given for multiple executions of a workflow on different initial inputs, it may end up exposing partial or complete *functionality* of some of the modules. As an example, consider the module v_3. For input $d_1 = 0, d_2 = 1$, $d_4 = 1$ and $d_6 = 0$, while for

input $d_1 = 0, d_2 = 0$, $d_4 = 0$ and $d_6 = 1$. Clearly, provenance data for module v_3 for these two executions exposes the function f_3. However, workflow modules may involve proprietary code or algorithms, requiring that their functionality be kept *private*. Therefore the *workflow owner* would like to limit the amount of provenance data that she gives the user for any execution of a workflow to ensure *privacy of modules* in terms of their functionality. But for the workflow to be practically meaningful, it is unrealistic to hide the initial input or the final output data. In this paper, we achieve privacy of modules in a workflow by hiding a carefully chosen subset of intermediate data.

In Figure 1, suppose that the workflow owner hides bit d_4. Now, even if the workflow is executed an arbitrary number of times, the user gets no information about function f_3, and some partial information about functions f_1 and f_2. To decide whether f_1 and f_2 remain private, we need to quantify the notion of privacy of modules in a workflow. We will give a formal definition later, but for now, we informally say that a module is ε-private for some parameter $0 < \varepsilon < 1$ if given any input to the module, the user can guess the correct output with probability at most ε. It can be shown that hiding d_4 gives $\frac{1}{2}$-privacy for v_1, v_2 and v_3.

Clearly, identical privacy guarantees for modules can be achieved by hiding different subsets of intermediate data. For example, hiding both d_3 and d_4 gives $\frac{1}{2}$-privacy for modules v_1, v_2 and v_3, but since the bit d_4 is shared by the inputs of both v_2 and v_3, hiding only this bit gives the same privacy guarantee. Moreover, some data may be more important than others, or may be contain more information, necessitating the introduction of a cost function which assigns a cost to each data item in the workflow. The cost of an item indicates the utility lost to the user when a data is hidden. Assuming that all initial input and final output data should always be visible, the following question arises: *what is the minimum cost of intermediate data that can be hidden while guaranteeing that individual modules are ε-private for some given parameter $\varepsilon > 0$?* We call this the SECURE VIEW problem. In this paper, we define and offer algorithmic solutions to this problem.

The data model considered in this paper is general enough to capture most practical applications. We assume that each data d is a binary string that comes from a fixed (but unbounded, and possibly infinite) domain $D \subseteq \{0,1\}^*$. Therefore, a module with p input ports and q output ports takes as input p binary strings, computes a function on them and outputs q binary strings. For example, a module might read two input strings, interpret them as integers and output their sum and product. Then, the function computed by the module maps from the set $(\{0,1\}^*)^2$ to itself.

We also assume that the user gets to see all connections between modules in the workflow, only the *values* of some data are hidden. Thus the data values on *all* connections carrying those data are hidden in *all* executions of the workflow.

Related Work. To the best of our knowledge, the only previous work that deals with security in a workflow is [9]. The paper develops a framework to output a *partial* view of the workflow that conforms to a given set of access permissions on the connections and input/output ports. However, their treatment of the notion of privacy is somewhat informal, and no guarantees on the quality of the solution are provided. Furthermore, in our problem, hiding internal connections may lead to disconnected input and output ports, potentially disconnecting the workflow network. Another relative weakness of the approach [9] is that it reveals less provenance information than our mechanism of only hiding data values, since the dependency of visible data on the modules is unknown to the user.

A related domain that has received considerable attention is that of *privacy-preserving data mining* (see surveys [2, 29], and the references therein). Here, the goal is to hide individual data attributes while retaining the suitability of the data for mining patterns. For example, the technique of *anonymizing data* makes each record indistinguishable from a large enough set of other records in certain identifying attributes [28, 21, 3]. Privacy preserving approaches have been studied for *social networks* [19, 6, 26, 8], *auditing queries* [24, 22] and in other contexts. Another widely used technique is that of *data perturbation* where some noise (usually random) is added to the the output of a query or to the underlying database. This technique is often used in *statistical databases*, where a query computes some aggregate function on the dataset [1, 10]. Privacy in statistical databases is often quantified using the framework of *differential privacy*, which requires that the output distribution is *almost* invariant to the inclusion of any particular record (see [14, 18, 13, 15] and surveys [11, 12]). However, in the context of data provenance, the goal is to hide module functionality rather than particular data items, hence there does not seem to be an obvious extension of the notion of differential privacy to our context. Further, for provenance information to be useful, individual data items that are revealed must be accurate, especially for applications such as ensuring reproducibility of experiments. Thus, adding noise to provenance information renders it useless. It would be interesting to see if the notion of *privacy of functions* that we introduce in this paper lends itself to a new notion of *differential privacy of functions*.

Contributions. Our first contribution is to formalize the notion of ε-privacy of a module when it is a standalone entity (called the *standalone module privacy*) as well as when it is a component of a workflow (called the *in-network module privacy*), interacting with other modules. While standalone module privacy can be naturally captured in terms of input and output data at the module interface that needs to be hidden, the notion of in-network privacy of a module is inherently linked to the network topology of the workflow and functionality of other modules. Even so, we are able to show that the problem of ensuring in-network privacy of a workflow of modules essentially reduces to implementing the standalone privacy requirements for each module. Building on this connection, we develop algorithms and complexity results for the SECURE VIEW problem, defined as the problem of ensuring in-network privacy of all modules in a workflow by hiding the smallest amount of data. The standalone privacy requirement of each module in a workflow can be specified in the most general form as a list of pairs of input-output data sets (called *set constraints*) that need to be hidden to ensure ε-privacy. Alternately, a more succinct but less expressive representation is to specify a list of pairs of numbers (called *cardinality constraints*) for each module such that hiding at least as many input and output data as specified in any pair guarantees standalone privacy for that module. Both variants of the SECURE VIEW problem are easily shown to be NP-hard, and thus we focus on poly-time *approximation algorithms*[1] for these problems.

Our second key contribution is a linear-programming (LP) based approach to obtain an $O(\log n)$-approximation for the SECURE VIEW problem with cardinality constraints. Moreover, we show that this is the best possible approximation guarantee achievable in poly-time, modulo standard complexity-theoretic assumptions. In contrast, we show that the SECURE VIEW problem with set constraints is much harder to approximate: in particular, we show that for some $\varepsilon > 0$, it is $\ell_{\max}^{\varepsilon}$-hard to approximate under standard complexity assumptions, where $\ell_{\max}$ is the length of longest requirement list for any module. On the other hand, an $\ell_{\max}$-approximation is easily achievable in poly-time, again using an LP-based approach. Finally, we show that both variants of the SECURE VIEW problem becomes more tractable when the workflow has *bounded data sharing*, i.e. when a data in the workflow acts as input to at most γ modules for some constant γ. We give a $(\gamma+1)$-approximation algorithm for this situation, and show that the problem is APX-hard[2] to approximate even when there is *no* data sharing (i.e. $\gamma = 1$).

Organization. Section 2 defines the notions of standalone and in-network ε-privacy. We formalize the SECURE VIEW problem in Section 3, and then establish a connection between standalone and in-network privacy of modules in Section 4. We present our approximation results for the SECURE VIEW problem with cardinality and set constraints in Sections 5 and 6, respectively. In Section 6, we also present approximation results for the SECURE VIEW problem with bounded data sharing.

2 Module Privacy in a Workflow

In this section we formalize the notion of module privacy in a workflow, which we call *in-network module privacy*. We start with the simpler notion of *standalone module privacy*, which defines the privacy of a module when it appears alone (and not as part of a workflow). We also introduce the notion of hiding input and output data of a module in order to guarantee its privacy. We will assume throughout that all modules in a workflow are initially private, i.e. the user has no apriori knowledge about the functionality of any module prior to the workflow executions.

We start with some terminology that is used throughout the paper. An *index* k is a positive integer, and an *index set* I is a set of indices. Each data (initial, intermediate and final) in a workflow is assumed to have a unique index. Conversely, each index has a domain associated with it, namely the domain of the data it is associated with. We denote the domain of index k by D_k. A *data vector* x is said to be *defined on an index set* I, denoted by $x \in \prod_{k\in I} D_k$, if x contains $|I|$ strings, each of which is associated with a unique index $k \in I$ and belongs to D_k. Then, $x[k]$ denotes the string in x mapped to index k. For a data vector $x \in \prod_{k\in I} D_k$, and for any subset of indices $J \subseteq I$, $x|_J \in \prod_{k\in J} D_k$ denotes x *restricted to* the indices in J, i.e. $x|_J[k] = x[k]$ for all $k \in J$. Finally, for any $J \subseteq I$, the *J-partition of* I is a partition of ~~all data vectors in $\prod_{k\in I} D_k$ such that two data ve~~ctors $x, y \in \prod_{k\in I} D_k$ are in the same subset of the partition iff $x|_J = y|_J$.

[1] An algorithm is said to be a $\mu(n)$-*approximation algorithm* for some non-decreasing function $\mu(n) : \mathbb{N}^+ \to \mathbb{N}$ if on every input of size n it computes a solution where the value is within a factor of $\mu(n)$ of the optimal algorithm.

[2] A problem is said to be *APX-hard* if there exists an $\varepsilon_0 > 0$ such that a $(1+\varepsilon_0)$-approximation in poly-time would imply $P = NP$.

2.1 Standalone Privacy of a Module

A standalone module v with input index set P and output index set Q computes a function $f : \prod_{p\in P} D_p \to \prod_{q\in Q} D_q$. As a running example throughout this section, we use the following function, where each data is a single bit.

Example 1. Let v be a module with input bits x_1, x_2 and output bits x_3, x_4, x_5. Suppose v computes function $f : \{0,1\}^2 \to \{0,1\}^3$ where $x_3 = x_1 \vee x_2$, $x_4 = \overline{x_1 \wedge x_2}$ and $x_5 = \overline{x_1 \oplus x_2}$. The truth table of function f is given in Table 1. Let $P = \{1,2\}$ and $Q = \{3,4,5\}$ be the input and output index sets of v in this example. □

x_1	x_2	x_3	x_4	x_5
0	0	0	1	1
0	1	1	1	0
1	0	1	1	0
1	1	1	0	1

Table 1: Truth table of the function f given in Example 1.

Our goal is to ensure that for any input $x \in \prod_{p\in P} D_p$ to v, a user is not be able to guess the value of $f(x)$ with probability more than ε, for a given parameter $\varepsilon > 0$. To enforce privacy of module v, we will *hide* a set of input and output data for each execution of the module; the remaining data will be *visible*. The input and output index sets corresponding to the hidden data are called *hidden input and output index sets*, and denoted by P^h and Q^h respectively. Correspondingly, the *visible input and output index sets* are $P^v = P \setminus P^h$ and $Q^v = Q \setminus Q^h$. On fixing a particular data set $x^v \in \prod_{p\in P^v} D_p$ as the visible part of the input data, the user sees an arbitrary data vector from the set $\{f(x)|_{Q^v} : x^v = x|_{P^v}\}$ as the visible output. For instance, in Example 1, if $P^v = \{1\}$ and $Q^v = \{3,5\}$, and if the user sets x^v to 0, then she sees an arbitrary string in set $\{(0,1),(1,0)\}$, i.e. the visible output bits corresponding to an arbitrary row of Table 1 containing $x_1 = 0$. We allow the user to query the module an unbounded number of times on all settings of the visible input data. The *observed relation* for function f with respect to the hidden index sets P^h and Q^h is then defined as the set of all data that the user might see in an execution, i.e.

$$R_{f,P^h,Q^h} = \{(x|_{P^v}, y|_{Q^v}) : x \in \prod_{p\in P} D_p, y = f(x)\}.$$

Note that there may be multiple functions g (besides f) that would result in the same observed relation. For instance, in Example 1, we can partition the output rows into those corresponding to $x_1 = 0$ and $x_1 = 1$, and then permute the output rows inside each subset of the partition arbitrarily (keeping the input rows fixed), to obtain another function with the same observed relation. Such functions are called *consistent functions*.

Definition 1. *A function $g : \prod_{p\in P} D_p \to \prod_{q\in Q} D_q$ is said to be* consistent *with module v with respect to hidden index sets P^h, Q^h if $R_{g,P^h,Q^h} = R_{f,P^h,Q^h}$. The set of all consistent functions for v with respect to P^h, Q^h is denoted by F_{v,P^h,Q^h}.*

We are now ready to define standalone privacy of a module. Recall that intuitively, a function is ε-private if for any input x, the user cannot guess the output with probability $> \varepsilon$. Given an input x, the output can be $g(x)$ for any consistent function g; hence the size of the set of images of x under all consistent functions is a measure of the confidence of a user in deciding the output corresponding to x.[3]

Definition 2. *Given a parameter $\varepsilon > 0$ and hidden input and output index sets P^h and Q^h, a module v is said to be* (P^h, Q^h, ε)-private *if $|\{g(x) : g \in F_{v,P^h,Q^h}\}| \geq \frac{1}{\varepsilon}$ for every input string $x \in \prod_{p\in P} D_p$.*

In general, more than one pair of index sets (P^h, Q^h) can guarantee ε-privacy of a module. The privacy requirement (called a *requirement list*) can therefore be defined as a list of index sets (called *set constraints*) sufficient for guaranteeing privacy.

Definition 3. *A module v is said to be $\langle L, \varepsilon\rangle_{\text{SET}}$-private, where $L = \{(P_j, Q_j) : 1 \leq j \leq \ell\}$, if* for all $1 \leq j \leq \ell$, *v is (P_j, Q_j, ε)-private.*

[3] In fact, one can show that the privacy condition in Definition 2 is equivalent to the user not being able to guess $f(x)$ with probability $> \varepsilon$ for any input x.

In Example 1, let $L = \{(x_1, x_3), (\emptyset, \{x_3, x_4\})\}$[4] be a requirement list. Hiding any one input and one output bit, or two output bits, yields $\frac{1}{4}$-privacy of v; thus, v is $\langle L, \frac{1}{4}\rangle_{\text{SET}}$-private. On the other hand, if $L = \{(\{x_1, x_2\}, \emptyset)\}$, v is not $\langle L, \frac{1}{4}\rangle$-private.

One shortcoming of the above representation is that the requirement list could be exponential in the number of input and output data. However, for many functions, the privacy requirement can be succinctly represented by a list of pairs of numbers (called *cardinality constraints*), where the module is ε-private if the number of hidden input and output data are at least the numbers specified by a pair. For instance, in Example 1, v is $\frac{1}{4}$-private if at least 2 output bits or 1 input and 1 output bit are hidden. Thus, a possible requirement list for $\frac{1}{4}$-privacy is $\{(1,1),(0,2)\}$. On the other hand, since hiding both input bits is not sufficient, $(2,0)$ cannot be part of any such list. Formally, we define $\langle L, \varepsilon\rangle_{\text{CARD}}$-privacy for a standalone module as follows:

Definition 4. *A module v is said to be $\langle L, \varepsilon\rangle_{\text{CARD}}$-private, where $L = \{(\alpha_j, \beta_j) : 1 \le j \le \ell\}$, if* for all P^h, Q^h *satisfying* $|P^h| \ge \alpha_j$ and $|Q^h| \ge \beta_j$ *for some* $1 \le j \le \ell$, *v is (P^h, Q^h, ε)-private.*

Clearly, $\langle L, \varepsilon\rangle_{\text{CARD}}$-privacy is a special case of $\langle L, \varepsilon\rangle_{\text{SET}}$-privacy, and therefore might impose stricter privacy requirements on the module. However, the trade-off is that the size of a non-redundant list[5] in this formulation is at most the number of input or output data of the module.

2.2 Workflows: Networks of Modules

A workflow, also called a *network of modules*, is modeled by a directed acyclic multi-graph (DAG) $G(V \cup \{s,t\}, E)$ with a single source node s and a single sink node t. The vertex set $V = \{v_1, v_2, \ldots, v_n\}$ represents the n modules of the workflow, computing functions $f_1, f_2, \ldots, f_n$ respectively. The source node s acts as the distributor of the initial input data to different modules in the workflow, and the sink node t aggregates all the final output data. A module v_i takes as input the output data produced by one or more v_j, or the initial input data distributed by s, and sends its output data to the inputs of other v_k and/or to the sink node t (see Figure 1).

The set of all data in the network, denoted by B, comprises the input data to the network (i.e. output data from s) and all the output data from $v_1, v_2, \ldots, v_n$. As mentioned previously, we assign a unique *index* to each data in B. Each module v_i has a fixed subset of indices $P_i \subseteq B$ as input and a fixed subset of indices $Q_i \subseteq B$ as output; these are called its *input and output index sets*. Hence for each module v_i, $f_i : \prod_{k \in P_i} D_k \to \prod_{k \in Q_i} D_k$. The output index set of s and the input index set of t are denoted by Q_s and P_t respectively—they represent the initial input and the final output data of the workflow. The *intermediate data* $B \setminus (Q_s \cup P_t)$ is denoted by B^{int}. For example, in the workflow given in Figure 1, $Q_s = \{d_1, d_2\}$, $P_t = \{d_5, d_6\}$ and $B^{int} = \{d_3, d_4\}$ are the initial input, final output and intermediate data respectively.

The edges denote connections between modules in the workflow. Each edge $e \in E$ carries a particular data in B, defined by a mapping $\phi : E(G) \to B$. Note that due to data sharing ϕ might not be a one-one map. For example, in Figure1, $\phi(e_4) = \phi(e_5) = d_4$.

Consider any assignment $\rho : B \to \prod_{b \in B} D_b$ of the data in B. For an index set $A \subseteq B$, $\rho(A)$ is defined as the restriction of $\rho(B)$ to the index set A. A run of the workflow G represents an assignment of B in any particular execution of the workflow.

Definition 5. *A* run *of the workflow G is an assignment $\rho : B \to \prod_{b \in B} D_b$ of the data in B such that for all $1 \le i \le n$, $f_i(x_i) = y_i$ where $x_i = \rho(P_i)$ and $y_i = \rho(Q_i)$.*

For any set of candidate functions $\mathbf{g} = \langle g_1, g_2, \ldots, g_n\rangle$ for the modules $v_1, v_2, \ldots, v_n$, we extend the notion of a run ρ of G to $\rho_{\mathbf{g}}$, by requiring that in Definition 5, $y_i = g_i(x_i)$ for each $1 \le i \le n$.

2.3 In-Network Privacy of a Module

Recall that the initial input and final output data of the workflow (corresponding to the index set $Q_s \cup P_t$) cannot be hidden. For each data b in the *hidden index set* $B^h \subseteq B^{int}$, the value of $\rho(b)$ is not revealed for *any* run ρ. On the other hand, for each data b in the *visible index set* $B^v = B \setminus B^h$, the value of $\rho(b)$ is revealed for *every* run ρ. Then, $P_i^h = P_i \cap B^h$ and $P_i^v = B^v \cap P_i$ are the hidden and visible input index sets of v_i respectively. Similarly, $Q_i^h = Q_i \cap B^h$ and $Q_i^v = B^v \cap Q_i$ are the hidden and visible output index sets respectively.

[4] We will often denote a singleton set $\{x\}$ by x.

[5] In a non-redundant requirement list L, for any two pairs (α_1, β_1) and $(\alpha_2, \beta_2) \in L$, either $\alpha_1 < \alpha_2$ and $\beta_1 > \beta_2$, or $\alpha_1 > \alpha_2$ and $\beta_1 < \beta_2$.

Given a network G, a hidden index set B^h, and a sequence of candidate functions $\mathbf{g} = \langle g_1, \ldots, g_n \rangle$ for the modules $v_1, \ldots, v_n$, the *observed relation* for $\mathbf{g}$ with respect to B^h is defined as

$$R_{\mathbf{g},B^h} = \langle (\rho_{\mathbf{g}}(P_i^v), \rho_{\mathbf{g}}(Q_i^v)) : 1 \leq i \leq n \rangle$$

Definition 6. *Let G be a workflow where module v_i computes function $f_i : \prod_{k \in P_i} D_k \rightarrow \prod_{k \in Q_i} D_k$, $1 \leq i \leq n$. A sequence of functions*

$$\mathbf{g} = \langle g_i : 1 \leq i \leq n, g_i : \prod_{k \in P_i} D_k \rightarrow \prod_{k \in Q_i} D_k \rangle$$

is said to be consistent *for workflow G with respect to a set of hidden data B^h, if $R_{\mathbf{g},B^h} = R_{\mathbf{f},B^h}$. The set of all consistent sequence of functions $\mathbf{g}$ for G with respect to B^h is denoted by F_{G,B^h}.*

For example, suppose bit d_4 is hidden in the workflow depicted in Figure 1. Consider functions $g_1 = (d_1, \overline{d_1 \vee d_2})$, $g_2 = \overline{d_3 \oplus d_4}$ and $g_3 = d_4$. It can be verified that the visible bits produced by these functions are exactly identical to those produced by f_1, f_2, f_3, in any execution of the workflow. Therefore, $\mathbf{g} = \langle g_1, g_2, g_3 \rangle$ is a consistent sequence of functions for the workflow, with respect to the hidden bit d_4.

Finally, we define ε-privacy of a workflow. The underlying intuition is the same as in the standalone case, namely for *any* input to *any* module in the workflow, the user should not be able to guess the correct output of the module with probability $> \varepsilon$.

Definition 7. *Given a parameter $\varepsilon > 0$ and a set of hidden data B^h, network G is said to be $\langle B^h, \varepsilon \rangle$-*private*, if for each $1 \leq i \leq n$ and each input $x_i \in \prod_{k \in P_i} D_k$ to module v_i, $|\{g_i(x_i) : \mathbf{g} = \langle g_1, g_2, \ldots, g_i, \ldots, g_n \rangle, \mathbf{g} \in F_{G,B^h}\}| \geq \frac{1}{\varepsilon}$.*

3 The SECURE VIEW Problem

Having given a formal definition of module privacy, we now define the SECURE VIEW problem as that of minimizing the total cost of hidden data while guaranteeing the required in-network privacy of the modules.

Formally, an instance of the SECURE VIEW problem comprises a network of modules represented by a single-source, single-sink directed acyclic multi-graph $G = (V \cup \{s,t\}, E)$, where $V = \{v_1, v_2, \ldots, v_n\}$ is the set of modules, and s and t are respectively the source and sink nodes. We are also given a set of data items B and a mapping $\phi : E \rightarrow B$ that indicates the data (or equivalently, index) corresponding to an edge. The set of all intermediate data is denoted by B^{int}. Only intermediate data $b \in B^{int}$ can be hidden, and c_b is the cost of hiding data b. Cost of hiding data is assumed to be *additive*, i.e. the cost of hiding a set of data $B^h \subseteq B^{int}$ is $c(B^h) = \sum_{b \in B^h} c_b$. Then given a parameter $\varepsilon > 0$, and the goal is to find a minimum cost subset B^h such that G is $\langle B^h, \varepsilon \rangle$-private.

To make the problem tractable, the privacy requirement of the workflow is specified in terms of a *requirement list* L_i for each module $v_i \in V$. Depending on the composition of the requirement lists, there are two variants of the problem. In the SECURE VIEW *problem with set constraints*, for each $1 \leq i \leq n$, $L_i = \{(P_{ij}, Q_{ij}) : 1 \leq j \leq \ell_i\}$ is a list of pairs of subsets of input and output data of v_i such that v_i is $\langle L_i, \varepsilon \rangle_{\text{SET}}$-private. On the other hand, in the SECURE VIEW *problem with cardinality constraints*, for each $1 \leq i \leq n$, $L_i = \{(\alpha_{ij}, \beta_{ij}) : 1 \leq j \leq \ell_i\}$ is a list of pairs of numbers such that v_i is $\langle L_i, \varepsilon \rangle_{\text{CARD}}$-private.

Note that the requirement lists specify *local* conditions for ensuring privacy of standalone modules, whereas a feasible solution requires *global* guarantees on their in-network privacy in the workflow. In the next section, we show that satisfying the local conditions specified by the requirement list of a module not only guarantees its standalone privacy, but also its in-network privacy in *any* workflow. This equivalence of standalone and in-network privacy guarantees is somewhat surprising, because privacy properties of standalone modules are expected to weaken when placed in a workflow due to mutual interaction. This connection simplifies the definition of the SECURE VIEW problem with set constraints (resp., cardinality constraints). Now, a feasible solution is a data set $B^h \subseteq B^{int}$ such that for each $1 \leq i \leq n$, $B^h \supseteq (P_{ij} \cup Q_{ij})$ (resp., $|B^h \cap P_i| \geq \alpha_{ij}$ and $|B^h \cap Q_i| \geq \beta_{ij}$) for some $1 \leq j \leq \ell_i$) for some $1 \leq j \leq \ell_i$.

4 Relationship between Standalone and In-network Module Privacy

We now show that if a pair of hidden input and output data vectors guarantee ε-privacy for a standalone module, they also guarantee ε-privacy for the module in any workflow, irrespective of its network topology.

We will assume throughout that the modules $V = \{v_1, v_2, \ldots, v_n\}$ are indexed in a topologically sorted order, and that P_i and Q_i denote the input and output index set of module v_i respectively.

Theorem 1. *Given a parameter $\varepsilon > 0$, for each $1 \leq i \leq n$, let $P_i^h \subseteq P_i^{int}$ and $Q_i^h \subseteq Q_i^{int}$ be two subsets of the intermediate input and output data of v_i respectively such that v_i is $(P_i^h, Q_i^h, \varepsilon)$-private as a standalone module. Then the entire workflow is $\langle B^h, \varepsilon \rangle$-private where $B^h = \cup_i (P_i^h \cup Q_i^h)$.*

It is tempting to try to show that any choice of a consistent function for a standalone module with respect to a particular pair of hidden input and output data vectors remains consistent for the module even when it is placed in a workflow, provided the same data vectors are hidden in the workflow. The above theorem would then follow immediately. However, it turns out that this is not true. In fact, the example below shows that the number of consistent functions for a module might decrease *doubly exponentially* (in the number of inputs) when the module is placed in a workflow.

Example 2. In this example, each data is a single bit. Let v be a module with k input and k output bits such that the underlying function reverses the input bits, i.e. $f(x_1, x_2, \ldots, x_k) = (x_k, x_{k-1}, \ldots, x_1)$. Let G be a workflow that comprises two copies of v (call them v_1 and v_2) joined in series. Thus G computes the identity map.

Note that, ε-privacy of v can be ensured by hiding any $\ell = \lceil \log(\frac{1}{\varepsilon}) \rceil$ output bits. (In fact, since f is a one-one function, hiding ℓ input bits of v also guarantees ε-privacy.) We compute the number of consistent functions of v with a particular set of ℓ hidden output bits. Given an input to v, the visible output bits are fixed for *any* consistent function; however, the hidden output bits can have arbitrary values. Thus, any input is mapped to one of $L = 2^\ell$ different outputs, and any consistent function is an arbitrary combination of the mappings for individual inputs. Then the number of consistent functions for v with ℓ hidden output bits is L^{2^k}.

In workflow G, the output bits of v_1 (i.e. the input bits of v_2) are the only intermediate data. Suppose the same ℓ output bits of v_1 as for the standalone v are hidden (this guarantees ε-privacy of both v_1 and v_2 as standalone modules). To compute the number of consistent functions for v_1 in the workflow with these hidden bits, we partition the set of 2^k different initial input strings into $2^k/L$ groups, where all L strings in a group have the same values of visible intermediate bits. Since both v_1 and v_2 are one-one functions, the workflow maps each such group to L distinct final outputs that are always visible. Therefore, v_1 has to map the strings in each such group to L *distinct* intermediate strings, implying that any consistent function for v_1 is necessarily one-one. Such a one-one function permutes the L intermediate strings corresponding to a group of L input strings. Thus, the total number of such functions is $(L!)^{2^k/L} \simeq ((2\pi L)^{1/2L}(L/e))^{2^k}$. The number of consistent functions for module v therefore decreases by a factor of $\left(\frac{(2\pi L)^{1/2L}}{e}\right)^{2^k} < 2^{-2^k}$ for any $L \geq 2$, i.e. $\varepsilon \leq \frac{1}{2}$. □

Theorem 1 shows that even though the number of consistent functions of a module can decrease substantially when the module is placed in a workflow as shown above, the privacy guarantees remain unchanged. In proving Theorem 1, our main technical result is Lemma 1, which states that given a set of hidden input and output data of a module, the set of possible outputs for a particular input to the module remains unchanged when the module is placed in a workflow, provided the same data are also hidden in the workflow. We continue to use P_i^h, Q_i^h to denote the indices of the hidden input and output data to v_i, and, $P_i^v = P_i \setminus P_i^h$, $Q_i^v = Q_i \setminus Q_i^h$ to denote the corresponding visible indices.

Lemma 1. *Consider any module v_i. Let X and Y be subsets of the P_i^v-partition of $\prod_{k \in P_i} D_k$ and the Q_i^v-partition of $\prod_{k \in Q_i} D_k$, and let x and y be any two data vectors in X and Y respectively. If there exists a $w \in X$ such that $z = f_i(w) \in Y$, then there exists a sequence of functions $\mathbf{g} = \langle g_j : 1 \leq j \leq n \rangle$ with $g_i(x) = y$ that is consistent for network G with respect to any set of hidden indices $B^h \supseteq P_i^h \cup Q_i^h$.*

First we show how this lemma implies Theorem 1. Let $x \in X$ and $y \in Y$ be two data vectors, where X and Y are subsets as defined in the lemma. There exists a consistent function g with $y = g(x)$ for v_i as a standalone module iff $(x|_{P_i^v}, y|_{Q_i^v}) \in R_{f_i, P_i^h, Q_i^h}$ is an observed input-output pair. Then, there must exist $w \in X, z \in Y$ (i.e. $x|_{P_i^v} = w|_{P_i^v}$ and

$y|_{Q_i^v} = z|_{Q_i^v}$) such that $f_i(w) = z$. The lemma claims that in this case, there exists a consistent sequence of functions $\mathbf{g} = \langle g_j : 1 \leq j \leq n \rangle$ for workflow G with $g_i(x) = y$, provided P_i^h and Q_i^h are also hidden in the workflow. Thus, the number of images, under consistent functions, of any input to v_i does not decrease when v_i is placed in a workflow, and Theorem 1 follows.

Before giving a formal proof of the lemma, we will give some intuition about our proof strategy. Consider a simple special case, where all the modules are connected in a chain. Assume that only output data of any module v_i are hidden, i.e. $P_i^h = \emptyset$. Then, $w = w|_{P_i^v} = x|_{P_i^v} = x$. Note that for $\mathbf{g}$ to be a consistent sequence of functions for the workflow, the visible data values in any execution must remain identical to that for the actual sequence of functions $\mathbf{f} = \langle f_1, f_2, \ldots, f_n \rangle$. However, the values of the hidden data might be different for an execution of the workflow with module functions $\mathbf{g}$ and $\mathbf{f}$. As a first try, let $\mathbf{g}$ be exactly identical to $\mathbf{f}$, except that $g_i(x) = y$ instead of z. Since $y, z \in Y$, they have exactly the same visible data values. Thus, $\mathbf{g}$ does not violate consistency in the output data of v_i. However, suppose that the visible parts of $f_{i+1}(y)$ and $f_{i+1}(z)$ do not match. Then, $\mathbf{g}$ violates the consistency requirement at the output of v_{i+1} for an execution with x as the input to v_i. To avoid this, we modify g_{i+1} (which is simply f_{i+1} at this point) by adding a filter σ_1 at the input of v_{i+1}. σ_1 is an identity map except that $\sigma_1(y) = z$. Thus, the new g_{i+1} is the composition of σ_1 with f_{i+1}, i.e. $\sigma_1 \cdot f_{i+1}$. The previous violation is successfully avoided, but this modification creates a different consistency violation. Consider an execution where the input to v_i is some x' such that $f_i(x') = y$. For this execution, the output of v_i was originally $f_{i+1}(y)$, but is now $g_{i+1}(y) = f_{i+1}(z)$. These two data vectors differ in their visible parts, and therefore $\mathbf{g}$ violates the consistency property. To fix this, we add a filter σ_2 at the output of v_i. σ_2 is an identity map except that $\sigma_2(y) = z$ and $\sigma_2(z) = y$. Then, the new g_i is $f_i \cdot \sigma_2$. Correspondingly, we replace the filter σ_1 at the input of v_{i+1} by a copy of σ_2. Then, the new definition of g_{i+1} is $\sigma_2 \cdot f_{i+1}$. It can be verified that $\mathbf{g}$ is now a consistent sequence of functions.

However, the general case introduces additional complications. First, input data of v_i might be hidden requiring us to add filters at the input of v_i. Further, due to data sharing, some of the hidden input data of v_i might also act as input to other modules—we need to add filters to the inputs of those modules as well. Finally, only a part of the output of a module v_i may act as input to another module v_j. In this case, v_j cannot distinguish between two different executions where the output of module v_i are distinct data vectors y and y' respectively, if y and y' are identical when restricted to the input index set of v_j. To handle this situation, we will make our filters more fine-grained, as shown below.

To formally prove the lemma, we introduce some additional notation to capture the notion of input and output filters that we described above. Let $x \in \prod_{i \in I} D_i$ and $a, b \in \prod_{j \in J} D_j$ for two index sets I and J. Then, $y = \textsc{Flip}_{a,b}(w) \in \prod_{i \in I} D_i$ is defined as

$$y[i] = \begin{cases} b[i] & \text{if } i \in J \text{ and } x[i] = a[i] \\ a[i] & \text{if } i \in J \text{ and } x[i] = b[i] \\ x[i] & \text{otherwise.} \end{cases}$$

Note that $\textsc{Flip}_{a,b}(\textsc{Flip}_{a,b}(x)) = x$. In the proof of the lemma, we will also use the notion of *function flipping*, which is defined as follows.

Definition 8. *Consider a function $f : \prod_{p \in P} D_p \rightarrow \prod_{q \in Q} D_q$. Let $a, b \in \prod_{i \in I} D_i$ for some index set I. Then, $\forall x \in \prod_{p \in P} D_p$, $\textsc{Flip}_{f,a,b}(x) = \textsc{Flip}_{a,b}(f(\textsc{Flip}_{a,b}(x)))$.*

Proof of Lemma 1. Let $a, b \in \prod_{k \in P_i \cup Q_i} D_k$ be defined as

$$a[k] = \begin{cases} x[k] & \text{if } k \in P_i \\ y[k] & \text{if } k \in Q_i \end{cases} \text{ and } b[k] = \begin{cases} w[k] & \text{if } k \in P_i \\ z[k] & \text{if } k \in Q_i. \end{cases}$$

(Note that $P_i \cap Q_i = \phi$.) Then, for each $1 \leq j \leq n$, we define $g_j = \textsc{Flip}_{f_j,a,b}$.

First, we claim that g_i maps x to y as desired. This holds since $g_i(x) = \textsc{Flip}_{f_i,a,b}(x) = \textsc{Flip}_{a,b}(f_i(\textsc{Flip}_{a,b}(x)))$ $= \textsc{Flip}_{a,b}(f_i(w)) = \textsc{Flip}_{a,b}(z) = y$.

To prove the consistency of $\mathbf{g} = \langle g_1, g_2, \ldots, g_n \rangle$ for G w.r.t. any set of hidden data $B^h \supseteq P_i^h \cup Q_i^h$, we need to show that for any run ρ, the visible data are identical for $\rho_{\mathbf{f}}$ and $\rho_{\mathbf{g}}$. We prove by induction on j the visible output data of every module v_j are identical in the two runs.

Let $c_{j,f}, c_{j,g}$ be the input data and $d_{j,f}, d_{j,g}$ the output data of module v_j for $\rho_{\mathbf{f}}$ and $\rho_{\mathbf{g}}$ respectively, where the initial input data is the same for both runs. Then, we prove that $d_{j,g} = \textsc{Flip}_{a,b}(d_{j,f})$. Since $x, w \in X$, they differ only in the

indices corresponding to hidden data; similarly, for $y,z \in Y$. Therefore the data vectors a and b only differ in hidden indices. Then if the above claim is true, for every j, $d_{j,g}$ and $d_{j,f}$ are the same on the visible indices, and therefore the visible data for both runs are the same.

Note that if the inductive hypothesis holds for all $j' < j$, then $c_{j,g} = \text{FLIP}_{a,b}(c_{j,f})$, since the modules are listed in a topological order. Thus,

$$\begin{aligned} d_{j,g} &= g_j(c_{j,g}) = \text{FLIP}_{f_j,a,b}(\text{FLIP}_{a,b}(c_{j,f})) \\ &= \text{FLIP}_{a,b}(f_j(\text{FLIP}_{a,b}(\text{FLIP}_{a,b}(c_{j,f})))) \\ &= \text{FLIP}_{a,b}(f_j(c_{j,f})) = \text{FLIP}_{a,b}(d_{j,f}). \quad \square \end{aligned}$$

5 The SECURE VIEW Problem with Cardinality Constraints

In this section, we give an $O(\log n)$-approximation algorithm for the SECURE VIEW problem with cardinality constraints. In addition, we show that this problem is $\Omega(\log n)$-hard under standard complexity-theoretic assumptions, even if the cost of hiding each data is identical and the requirement list of every module in the workflow contains exactly one pair of numbers.

5.1 $O(\log n)$-Approximation Algorithm

Our algorithm is based on rounding the fractional relaxation (called the LP relaxation) of the integer linear program (IP) for this problem presented in Figure 2.

$$\text{Minimize } \sum_{b \in B^{int}} c_b x_b \quad \text{subject to}$$

$$\sum_{j=1}^{\ell_i} r_{ij} \geq 1 \quad \forall 1 \leq i \leq n \tag{1}$$

$$\sum_{b \in P_i} y_{bij} \geq r_{ij}\alpha_{ij} \quad \forall 1 \leq i \leq n,\ \forall 1 \leq j \leq \ell_i \tag{2}$$

$$\sum_{b \in Q_i} z_{bij} \geq r_{ij}\beta_{ij} \quad \forall 1 \leq i \leq n,\ \forall 1 \leq j \leq \ell_i \tag{3}$$

$$\sum_{j=1}^{\ell_i} y_{bij} \leq x_b, \quad \forall 1 \leq i \leq n, \forall b \in P_i^{int} \tag{4}$$

$$\sum_{j=1}^{\ell_i} z_{bij} \leq x_b, \quad \forall 1 \leq i \leq n, \forall b \in Q_i^{int} \tag{5}$$

$$y_{bij} \leq r_{ij}, \quad \forall 1 \leq i \leq n,\ \forall 1 \leq j \leq \ell_i,\ \forall b \in P_i^{int} \tag{6}$$

$$z_{bij} \leq r_{ij}, \quad \forall 1 \leq i \leq n,\ \forall 1 \leq j \leq \ell_i,\ \forall b \in Q_i^{int} \tag{7}$$

$$x_b, r_{ij}, y_{bij}, z_{bij} \in \{0,1\} \tag{8}$$

Figure 2: An IP for the SECURE VIEW problem with cardinality constraint

Recall that each module v_i has a requirement list $L_i = \{(\alpha_{ij}, \beta_{ij}) : 1 \leq j \leq \ell_i\}$. A feasible solution must ensure that for each $1 \leq i \leq n$, there exists a j such that at least α_{ij} input data and β_{ij} output data of v_i are hidden.

In this IP, $x_b = 1$ if data b is hidden, and $r_{ij} = 1$ if at least α_{ij} input data and β_{ij} output data of module v_i are hidden. Then, $y_{bij} = 1$ (resp., $z_{bij} = 1$) if both $r_{ij} = 1$ and $x_b = 1$, i.e. if data b contributes to satisfying the input requirement α_{ij} (resp., output requirement β_{ij}) of module v_i. Let us first verify that the IP indeed solves the SECURE VIEW problem with cardinality constraints. For each module v_i, constraint (1) ensures that for some $1 \leq j \leq \ell_i$, $r_{ij} = 1$. In conjunction

with constraints (2) and (3), this ensures that for some $1 \leq j \leq \ell_i$, (i) at least α_{ij} input data of v_i have $y_{bij} = 1$ and (ii) at least β_{ij} output data of v_i have $z_{bij} = 1$. But, constraint (4) (resp., constraint (5)) requires that whenever $y_{bij} = 1$ (resp., $z_{bij} = 1$), data b be hidden, i.e. $x_b = 1$, and a cost of c_b be added to the objective. Thus the set of hidden data satisfy the privacy requirement of each module v_i and the value of the objective is the cost of the hidden data. Note that constraints (6) and (7) are also satisfied since y_{bij} and z_{bij} are 0 whenever $r_{ij} = 0$. Thus, the IP represents the SECURE VIEW problem with cardinality constraints.

One can write a simpler IP for this problem, where the summations are removed from constraints (4) and (5), and constraints (6) and (7) are removed altogether. To see the necessity of these constraints, consider the LP relaxation of the IP, obtained by replacing constraint (8) with $x_b, r_{ij}, y_{bij}, z_{bij} \in [0,1]$.

Suppose constraints (6) and (7) were missing from the IP, and therefore from the LP as well. For a particular $1 \leq i \leq n$, it is possible that a fractional solution to the LP has $r_{ij} = 1/2$ for two distinct values j_1 and j_2 of j, where $\alpha_{ij_1} > \alpha_{ij_2}$ and $\beta_{ij_1} < \beta_{ij_2}$. But constraint (2) (resp., constraint (3)) can now be satisfied by setting $y_{bij_1} = y_{bij_2} = 1$ (resp., $z_{bij_1} = z_{bij_2} = 1$) for $\alpha_{ij_1}/2$ input data (resp., $\beta_{ij_2}/2$ output data). However, $(\alpha_{ij_1}/2, \beta_{ij_2}/2)$ might not satisfy the privacy requirement for i, forcing an integral solution to hide some data b with $x_b = 0$. This will lead to an unbounded integrality gap.

Now, suppose constraints (2) and (3) did not have the summation. For a particular $1 \leq i \leq n$, it is possible that a fractional solution to the LP has $r_{ij} = 1/\ell_i$ for all $1 \leq j \leq \ell_i$. Constraint (2) (resp., constraint (3)) can then be satisfied by setting $y_{bij} = 1/\ell_i$ for all $1 \leq \ell_i$, for $\max_j\{\alpha_{ij}\}$ distinct input data (resp., $\max_j\{\beta_{ij}\}$ distinct output data). Correspondingly, $x_b = 1/\ell_i$ for those data b. If all the α_{ij}s and β_{ij}s for different $1 \leq j \leq \ell_i$ have similar values, it would mean that we are satisfying the privacy constraint for v_i paying an ℓ_i fraction of the cost of an integral solution. This can be formalized to yield an integrality gap of $\max_i\{\ell_i\}$, which could be n. Introducing the summation in the LP precludes this possibility.

We round the fractional solution to the LP relaxation using Algorithm 1. For each $1 \leq j \leq \ell_i$, let P_{ij}^{min} and Q_{ij}^{min} be the α_{ij} input and β_{ij} output data of v_i with minimum cost. Then, B_i^{min} represents $P_{ij}^{min} \cup Q_{ij}^{min}$ of minimum cost.

Algorithm 1 Rounding algorithm of LP relaxation of the IP given in Figure 2, **Input**: An optimal fractional solution $\{x_b | b \in B^{int}\}$, **Output**: A feasible SECURE VIEW solution B^h

1: Initialize $B^h = \phi$.
2: For each $b \in B^{int}$, include b in B^h with probability $\min\{1, 16 x_b \log n\}$.
3: For each module v_i whose privacy requirement is not satisfied by B^h, add B_i^{min} to B^h.
4: Return B^h.

Analysis. We can assume wlog that the requirement list L_i for each $v_i \in V$ is non-redundant, i.e. for all $1 \leq j_1 \neq j_2 \leq \ell_i$, either $\alpha_{ij_1} > \alpha_{ij_2}$ and $\beta_{ij_2} < \beta_{ij_1}$, or $\alpha_{ij_1} < \alpha_{ij_2}$ and $\beta_{ij_2} > \beta_{ij_1}$. We can thus assume that for each module v_i, the list L_i is sorted in increasing order on the values of α_{ij} and in decreasing order on the values of β_{ij}. The following lemma shows that step 2 satisfies the privacy requirement of each module with high probability

Lemma 2. *Let v_i be any module in V. Then with probability at least $1 - 2/n^2$, there exists a $1 \leq j \leq \ell_i$ such that $|P_i^h| \geq \alpha_{ij}$ and $|Q_i^h| \geq \beta_{ij}$.*

Proof. Given a fractional solution to the LP relaxation, let $1 \leq m \leq \ell_i$ be the index corresponding to the *median* $(\alpha_{ij}, \beta_{ij})$, satisfying $\sum_{j=1}^{m-1} r_{ij} < 1/2$ and $\sum_{j=1}^{m} r_{ij} \geq 1/2$. We will show that after step 2, at least α_{im} input data and β_{im} output data is hidden with probability at least $1 - 2/n^2$ for module v_i.

We partition the set of intermediate data B^{int} into two sets: the set of data deterministically included in B^h, $B^{det} = \{b : x_b \geq 1/16 \log n\}$, and the set of data probabilistically rounded, $B^{prob} = B^{int} \setminus B^{det}$. Also, let $B^{round} = B^h \setminus B^{det}$ be the set of intermediate data that are actually hidden among B^{prob}. For each module v_i, let $P_i^{det} = B_{det} \cap P_i^{int}$ and $Q_i^{det} = B^{det} \cap Q_i^{int}$ be the set of hidden input and output data in B^{det} respectively. Let the size of these sets be $\alpha_i^{det} = |P_i^{det}|$ and $\beta_i^{det} = |Q_i^{det}|$. Also, let $P_i^{prob} = B^{prob} \cap P_i^{int}$ and $Q_i^{prob} = B^{prob} \cap Q_i^{int}$. Finally, let $P_i^{round} = B^{round} \cap P_i^{int}$ and $Q_i^{round} = B^{round} \cap Q_i^{int}$. We show that for any module v_i, $|P_i^{round}| \geq \alpha_{im} - \alpha_i^{det}$ and $|Q_i^{round}| \geq \beta_{im} - \beta_i^{det}$ with probability at least $1 - 1/n^2$.

First we show that $\sum_{b\in P_i^{prob}} x_b \geq (\alpha_{im} - \alpha_i^{det})/2$. Constraint (2) implies that $\sum_{b\in P_i^{int}} y_{bij} \geq r_{ij}\alpha_{ij}$, while constraint (6) ensures that $\sum_{b\in P_i^{det}} y_{bij} \leq r_{ij}\alpha_i^{det}$. Combining these, we have

$$\sum_{b\in P_i^{prob}} y_{bij} \geq r_{ij}(\alpha_{ij} - \alpha_i^{det}). \tag{9}$$

From constraint (4), we have

$$\sum_{b\in P_i^{prob}} x_b \geq \sum_{b\in P_i^{prob}} \sum_{j=1}^{\ell_i} y_{bij} \geq \sum_{j=m}^{\ell_i} \sum_{b\in P_i^{prob}} y_{bij}.$$

Then, from Eqn. (9),

$$\sum_{b\in P_i^{prob}} x_b \geq \sum_{j=m}^{\ell_i} r_{ij}(\alpha_{ij} - \alpha_i^{det}) \geq (\alpha_{im} - \alpha_i^{det}) \sum_{j=m}^{\ell_i} r_{ij}.$$

Finally, using constraint (1), we conclude that

$$\sum_{b\in P_i^{prob}} x_b \geq \frac{\alpha_{im} - \alpha_i^{det}}{2}. \tag{10}$$

Similarly, since $\sum_{j=1}^{m} r_{ij} \geq 1/2$ and the list L_i is sorted in decreasing order of β_{ij}, it follows that

$$\sum_{b\in Q_i^{prob}} x_b \geq \frac{\beta_{im} - \beta_i^{det}}{2}. \tag{11}$$

Next we show that $|P_i^{round}| \geq \alpha_{im} - \alpha_i^{det}$ with probability $\geq 1 - 1/n^2$. Each $b \in B^{prob}$ is independently included in B^{round} with probability $16x_b \log n$. Hence, by Eqn. (10),

$$E[|P_i^{round}|] = \sum_{b\in P_i^{prob}} 16x_b \log n \geq 8(\alpha_{im} - \alpha_i^{det}) \log n.$$

Using Chernoff bound[6], $|B_{round} \cap P_i^{int}| \leq \alpha_{im} - \alpha_i^{det}$ with probability at most $1/n^2$. Similarly, using Eqn. (11), $|Q_i^{round}| \leq \beta_{im} - \beta_i^{det}$ with probability at most $1/n^2$. The lemma follows by using union bound over the failure probabilities. □

Using Lemma 2, we prove the following theorem.

Theorem 2. *Algorithm 1 gives a feasible solution with expected cost $O(\log n)$ times the optimal.*

Proof. Using union bound over the set of modules in the above lemma, we conclude that with probability at least $1 - 2/n$, the solution produced by the rounding algorithm after step 2 is feasible. By linearity of expectation, the cost of the rounded solution at this stage is at most $16 \log n$ times that of the LP solution, and therefore $O(\log n)$ times that of the optimal cost. If all modules are not satisfied after step 3, the cost of the data added to B^h in step 3 is at most $O(n)$ times the optimal. However, this happens with probability at most $2/n$; thus the expected total cost of the solution produced by this algorithm remains $O(\log n)$ times the optimal cost. □

[6] If X is sum of independent boolean random variables with $E[X] = \mu$, then $\Pr[X \leq \mu(1-\varepsilon)] \leq e^{\frac{-\mu\varepsilon^2}{2}}$ (see, for instance, [23]).

5.2 $\Omega(\log n)$-Hardness

The following theorem shows that Algorithm 1 produces an optimal answer upto a constant factor.

Theorem 3. *It is not possible to approximate the* SECURE VIEW *problem with cardinality constraints within a factor of $o(\log n)$ unless* NP $\subseteq$ DTIME$(n^{O(\log\log n)})$ *even if the maximum list size $\ell_{\max} = 1$ and each data has unit cost.*

We give a reduction from the minimum set cover problem to this version of the SECURE VIEW problem where $\ell_{\max} = 1$ and each data has unit cost. Since set cover is hard to approximate within a factor of $o(\log n)$ unless NP $\subseteq$ DTIME$(n^{O(\log\log n)})$ [16, 20], the hardness result of the SECURE VIEW problem with cardinality constraints follows under the same assumption.

An instance of the set cover problem consists of an input universe $U = \{u_1, u_2, \ldots, u_n\}$, and a set of its subsets $\mathbf{S} = \{S_1, S_2, \ldots, S_m\}$, i.e. each $S_i \subseteq U$. The goal is to find a set of subsets $\mathbf{T} \subseteq \mathbf{S}$ of minimum size (i.e. $|\mathbf{T}|$ is minimized) subject to the constraint that $\cup_{S_i \in \mathbf{T}} S_i = U$.

We create an instance $G = (V \cup \{s,t\}, E)$ of the SECURE VIEW problem, where the set of modules V contains a module v_i corresponding to each element $u_i \in U$, and an additional module z. There is a single edge (s,z) from s to z, a set of edges $\{e_{ij} : S_i \ni u_j\}$ from z to each v_j, and a single edge e_j from each v_j to the sink node t. The edge (s,z) uniquely carries data b_s, and each edge e_j uniquely carries data b_j for $1 \leq j \leq n$. All edges $\{e_{ij} : j \in S_i\}$ carry the same data c_i, $1 \leq i \leq m$. Note that only $\{c_i : 1 \leq i \leq m\}$ are intermediate data, and therefore can be hidden; the cost of hiding each such data is 1. The privacy requirement for z is any single data carried by one of its outgoing edges, while that for each v_i is any single data carried by one of its incoming edges. Note that the maximum list size $\ell_{\max}$ is 1.

If the minimum set cover problem has a cover of size k, hiding the data corresponding to the subsets selected in the cover produces a solution of cost k for this instance of the SECURE VIEW problem. Conversely, if a solution to the SECURE VIEW problem hides a set of k data in $\{c_i : 1 \leq i \leq m\}$, selecting the corresponding sets produces a cover of k sets.

6 The SECURE VIEW Problem with Set Constraints

We now consider the SECURE VIEW problem with set constraints. Let $\ell_{\max}$ be the maximum size of the requirement list of a module. We show the following theorem.

Theorem 4. *Unless* NP $\subseteq$ DTIME$(n^{\text{polylog } n})$*, the* SECURE VIEW *problem with set constraints cannot be approximated to within a factor of $\ell_{\max}^{\varepsilon}$ for some $\varepsilon > 0$. The hardness result holds even when $\ell_{\max}$ is a (sufficiently large) constant, and each data has unit cost. Further, it is possible to get a factor $\ell_{\max}$-approximation in polynomial time.*

We give an $\ell_{\max}$-approximation algorithm for this problem in the appendix.

6.1 $\ell_{\max}^{\varepsilon}$-Hardness

The hardness result in Theorem 4 is obtained by a reduction from the *minimum label cover* problem [5]. An instance of the minimum label cover problem consists of a bipartite graph $H = (U, W, E_H)$, a label set L, and a non-empty relation $R_{uw} \subseteq L \times L$ for each edge $(u,w) \in E_H$. A feasible solution is a label assignment to the vertices, $A : U \cup V \rightarrow 2^L$, such that for each edge (u,w), there exist $\ell_1 \in A(u), \ell_2 \in A(w)$ such that $(\ell_1, \ell_2) \in R_{uw}$. The objective is to find a feasible solution that minimizes $\sum_{u \in U \cup W} |A(u)|$.

Unless NP $\subseteq$ DTIME$(n^{\text{polylog } n})$, the label cover problem is $|L|^{\varepsilon}$-hard to approximate for some constant $\varepsilon > 0$ [5, 27]. The instance of the SECURE VIEW problem in the reduction has $\ell_{max} = |L|^2$. Theorem 4 follows immediately.

Given an instance of the label cover problem as defined above, we create an instance $G = (V \cup \{s,t\}, E)$ of the SECURE VIEW problem (refer to Figure 3). For each edge $(u,w) \in E_H$, there is a module $x_{uw} \in V$. In addition, V has another module z. As shown in Figure 3, there are three kinds of edges in V: (i) a single edge e_z from the source node s to z, (ii) a set of $|2L|$ parallel edges $e_{uw,u,\ell}$ from z to each x_{uw} indexed by one of u or w and a label $\ell \in L$, and (iii) a single e_{uw} edge from each x_{uw} to the sink node t. Edge e_z and e_{uw}, for each $(u,w) \in E$, uniquely carry data b_z and b_{uw} respectively, whereas data $b_{u,\ell}$ is carried by all edges $e_{uw,u,\ell}$. Note that $b_{u,\ell}$ are the only intermediate data in the workflow; the cost of hiding each such data is 1. The requirement list of z contains singleton subsets of

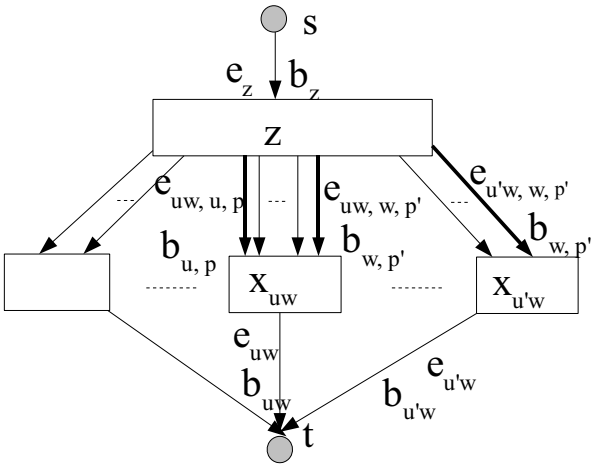

Figure 3: Reduction from label cover: bold edges correspond to $(p, p') \in R_{uw}$.

each intermediate data, while that of x_{uw}, for each $(u,w) \in E$, contains pairs of data corresponding to members of the relation R_{uw}, i.e $L_{uw} = \{(b_{u,\ell_1}, b_{w,\ell_2}) : (\ell_1, \ell_2) \in R_{uw}\}$.

The following lemma proves the correctness of the reduction.

Lemma 3. *The label cover instance H has a solution of cost k iff the* SECURE VIEW *instance G has a solution of cost k.*

Proof. Let $A : U \cup W \to 2^L$ be a solution of the label cover instance H with total cost $k = \sum_{u \in U \cup W} |A(u)|$. We create a solution B^h for the SECURE VIEW instance G as follows: for each $u \in U \cup W$, and $\ell \in L$, add $b_{u,\ell}$ to B^h iff $\ell \in A(u)$. We claim that B^h is a feasible solution for G. For each $u \in U \cup W$, $A(u)$ is non-empty; hence, the requirement of z is trivially satisfied. Since A is a valid label cover solution, for each $(u,w) \in E_H$, there exists $\ell_1 \in A(u)$ and $\ell_2 \in A(w)$ such that $(\ell_1, \ell_2) \in R_{uw}$. Hence for the same pair (ℓ_1, ℓ_2), $(b_{u,\ell_1}, b_{w,\ell_2}) \in L_{x_{uw}}$, and both $b_{u,\ell_1}, b_{w,\ell_2} \in B^h$. This satisfies the requirement for all $x_{uw} \in V$.

Conversely, let B^h be a solution of the SECURE VIEW instance G, where $|B^h| = k$. Note that B^h can include only the intermediate data. For each $u \in U \cup W$, we define $A(u) = \{\ell | b_{u,\ell} \in B^h\}$. Clearly, $\sum_{u \in U \cup W} |A(u)| = k$. For each $x_{uw} \in V$, the requirement of x_{uw} is satisfied by B^h; hence there exist $\ell_1, \ell_2 \in L$ such that $b_{u,\ell_1}, b_{w,\ell_2} \in B^h$. This implies that for each edge $(u,w) \in E_H$, there exist $\ell_1 \in A(u)$ and $\ell_2 \in A(w)$, where $(\ell_1, \ell_2) \in R_{uw}$, thereby proving feasibility. □

Remark. If $N = |U| + |W|$, the label cover problem is also known to be $\Omega(2^{\log^{1-\gamma} N})$-hard to approximate for all constant $\gamma > 0$, unless NP $\subseteq$ DTIME($n^{\text{polylog } n}$) [5, 27]. Thus, the SECURE VIEW problem with set constraints is $\Omega(2^{\log^{1-\gamma} n})$-hard to approximate as well, for all constant $\gamma > 0$, under the same complexity assumption.

6.2 $\ell_{\max}$-Approximation Algorithm

Here we give an $\ell_{\max}$-approximation algorithm for the SECURE VIEW problem with set constraints as claimed in Theorem 4. The algorithm rounds the solution given by LP relaxation of the following integer program:

Minimize $\sum_{b \in B^{int}} c_b x_b$ subject to

$$\sum_{j=1}^{\ell_i} r_{ij} \geq 1 \qquad \forall 1 \leq i \leq n \tag{12}$$

$$x_b \geq r_{ij} \qquad \forall b \in P_{ij} \cup Q_{ij},\ \forall 1 \leq i \leq n \tag{13}$$

$$x_b, r_{i,j} \in \{0,1\} \tag{14}$$

The LP relaxation is obtained by changing Eqn. (13) to

$$x_b, r_{ij} \in [0,1]. \tag{15}$$

The rounding algorithm outputs $B^h = \{b : x_b \geq 1/\ell_{\max}\}$.

Since the maximum size of a requirement list is $\ell_{\max}$, for each i, there exists a j such that in the solution of the LP, $r_{ij} \geq 1/\ell_i \geq 1/\ell_{\max}$. Hence there exists at least one $1 \leq j \leq \ell_i$ such that $P_{ij} \subseteq P_i^h, Q_{ij} \subseteq Q_i^h$. Since $c(B^h)$ is most $\ell_{\max}$ times the cost of LP solution, this algorithm gives an $\ell_{\max}$-approximation.

7 The SECURE VIEW Problem with Bounded Data Sharing

The SECURE VIEW problem becomes substantially easier to approximate if the workflow has *bounded data sharing*, i.e. when every data d produced by some module is either a final output data or is an input data to at most γ other modules. Though the problem remains NP-hard even with this restriction, the following theorem shows that it is possible to approximate it within a constant factor when γ is a constant.

Theorem 5. *There is a $(\gamma+1)$-approximation algorithm for both versions of the* SECURE VIEW *problem when each data acts as input to at most γ modules. On the other hand, both versions of the problem remain APX-hard even when there is* no *data sharing (i.e. $\gamma = 1$), each data has unit cost, and $\ell_{\max}$ is 2.*

First, we give a $(\gamma+1)$-approximation algorithm for the SECURE VIEW problem with set constraints, where each data is shared by at most γ edges. This also implies an identical approximation factor for the cardinality version. Then, we show that the cardinality version of the problem is APX-hard, i.e. there exists a constant $c > 1$ such that it is NP-hard to obtain a c-approximate solution to the problem. The set version is therefore APX-hard as well.

7.1 $(\gamma+1)$-Approximation Algorithm

Recall that the input to the problem includes a requirement list $L_i = \{(P_{ij}, Q_{ij}) : 1 \leq j \leq \ell_i\}$ for each module v_i. Let (P_{ij^*}, Q_{ij^*}) be a minimum cost pair for module v_i, i.e. $c(P_{ij^*} \cup Q_{ij^*}) = \min_{j=1}^{\ell_i} c(P_{ij} \cup Q_{ij})$. The algorithm greedily satisfies the (P_{ij^*}, Q_{ij^*}) requirement for each module v_i, i.e. the set of hidden data $B^h = \bigcup_{1 \leq i \leq n} (P_{ij^*} \cup Q_{ij^*})$.

Note that each intermediate data is an input to at most γ modules. In any optimal solution, assume that each terminal module of every hidden edge carrying this data pays its cost. Then, the total cost paid by the modules is at most $\gamma+1$ times the cost of the optimal solution. On the other hand, the total cost paid by any module is at least the cost of the edges incident on the module that are hidden by the algorithm. Thus, the solution of the algorithm has cost at most $\gamma+1$ times the optimal cost.

A very similar greedy algorithm with the same approximation factor can be given to the SECURE VIEW problem with cardinality constraints.

7.2 APX-Hardness

We give a reduction from the *minimum vertex cover* problem in cubic graphs to the SECURE VIEW problem with cardinality constraints. An instance of the vertex cover problem consists of an undirected graph $G'(V', E')$. The goal is to find a subset of vertices $S \subseteq V'$ of minimum size $|S|$ such that each edge $e \in E'$ has at least one endpoint in S.

Given an instance of the vertex cover problem, we create an instance of the SECURE VIEW problem $G(V \cup \{s,t\}, E)$ (see Figure 4). For each edge $(u,v) \in E'$, there is a module x_{uv} in V; also there is a module y_v for each vertex $v \in V'$. In addition to these, V contains a single module z. For each x_{uv}, the edge set E includes an edge (s, x_{uv}) and edges (x_{uv}, y_u) and (x_{uv}, y_v). There is an edge (y_v, z) in E for every $y_v \in V$. Finally, there is an edge (z,t) in E.

There is no data sharing in the workflow, and hiding a data is equivalent to hiding the edge carrying it. So, we do not introduce an explicit data set, and consider hiding edges instead of data. Note that only edges of the form (x_{uv}, y_u) can be hidden, since they are the only ones carrying intermediate data; the cost of hiding each such edge is 1.

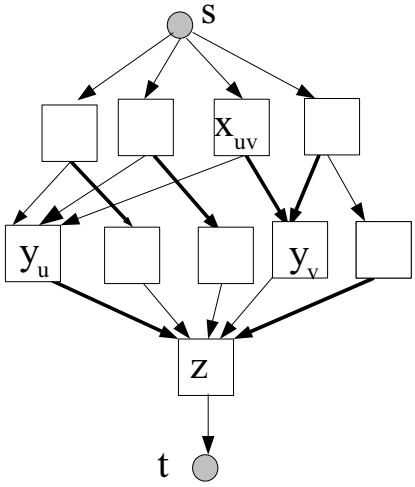

Figure 4: Reduction from vertex cover, the dark edges show a solution with cost $|E'| + k$, $k =$ size of a vertex cover in G'

Finally, we define the requirement list for each module in V. For each x_{uv}, $L_{uv} = \{(0,1)\}$, i.e. the requirement for x_{uv} is any single outgoing edge. For each y_v, $L_v = \{(d_v, 0), (0,1)\}$, where d_v is the degree of the v in G'. Hence the requirement of the vertex y_v is either all of its incoming edges, or a single outgoing edge. For vertex z, $L_z = \{(1,0)\}$, i.e. hiding any incoming edge suffices.

Lemma 4. *The vertex cover instance G' has a solution of size k if and only if the* SECURE VIEW *instance G has a solution of cost $m' + k$, where $m' = |E'|$ is the number of edges in G'.*

Proof. Let $S \subseteq V'$ be a vertex cover of G' of size k. We create a create a set of hidden edges B^h for the SECURE VIEW problem as follows: for each $v \in S$, add (y_v, z) to B^h. Further, for each x_{uv}, if $u \notin S$, add (x_{uv}, y_u) to B^h, otherwise add (x_{uv}, y_v). We claim that B^h is a feasible solution for G. Clearly, the requirement is satified for each x_{uv}, since one outgoing edge is hidden; the same holds for all y_v such that $v \in S$. Assuming E' to be non-empty, any vertex cover is of size at least one. Hence at least one incoming edge to z is hidden. Finally, for every y_v such that $v \notin S$, all its incoming edges are hidden; if not, S is not a vertex cover. Hence B^h satisfies the requirement of all vertices in V. Since we hide exactly one outgoing edge from all x_{uv}, and exactly one outgoing edge from all y_v where $v \in S$, the cost of the solution is $m' + k$.

Now assume that we have a solution $B^h \subseteq B$ of the SECURE VIEW instance with cost $|B^h| = k'$. We can assume, wlog, that for each x_{uv} exactly one outgoing edge is included in B^h; if both (x_{uv}, y_u) and (x_{uv}, y_v) are in B^h, we arbitrarily select one of u or v, say u, and replace the edge (x_{uv}, y_u) in B^h with the edge (y_u, t) to get another feasible solution without increasing the cost. We claim that the set $S \subseteq V'$ of vertices v such that $(y_v, z) \in B^h$ forms a vertex cover. For any edge $(u,v) \in E'$, if $(x_{uv}, y_u) \notin B^h$, then $(y_u, z) \in B^h$ to satisfy the requirement of y_u, and therefore $u \in S$; otherwise, $v \in S$ by the same argument. Hence S is a vertex cover. Since each vertex x_{uv} has exactly one outgoing edge in B^h, $|S| = k' - m'$. □

To complete the proof, note that if G' were a cubic graph, i.e. the degree of each vertex is at most 3, then the size of any vertex cover $k \geq m'/3$. It is known that vertex cover in cubic graphs is APX-hard [4]; hence so is the SECURE VIEW problem with cardinality constraints and no data sharing. An exactly identical reduction shows that the SECURE VIEW problem with set constraints and no data sharing is APX-hard as well.

8 Conclusions

We developed a model of privacy for module functionality in workflows where data are made public over repeated workflow executions. Our model motivates a natural optimization problem, the SECURE VIEW problem, which seeks to identify the smallest amount of intermediate data that needs to be hidden so that the functionality of every module is kept private. We developed algorithms and hardness results that characterize the complexity of the SECURE VIEW problem.

Our work suggests several promising directions for future exploration. One natural direction is to understand the connection between the network topology and the tractability of the SECURE VIEW problem. In the presence of unrestricted data sharing, our hardness results hold even for very simple network topologies (special cases of series-parallel graphs). However, when there is no data sharing, our initial work suggests that the problem becomes poly-time solvable for some interesting and natural classes of graphs (e.g., series-parallel graphs). A more ambitious future direction is to develop models of privacy when some of the modules in the workflow are public, that is, the underlying code is publicly available. In the absence of any restriction on the behavior of public modules, it is not difficult to construct workflows where the presence of such modules provably compromises privacy of proprietary modules. Thus handling privacy in this more general setting requires identifying natural restrictions on the behavior of public modules.

References

[1] Nabil R. Adam and John C. Worthmann. Security-control methods for statistical databases: a comparative study. *ACM Comput. Surv.*, 21(4):515–556, 1989.

[2] Charu C. Aggarwal and Philip S. Yu. Privacy-preserving data mining: Models and algorithms. 2008.

[3] Gagan Aggarwal, Tomás Feder, Krishnaram Kenthapadi, Samir Khuller, Rina Panigrahy, Dilys Thomas, and An Zhu. Achieving anonymity via clustering. In *PODS '06: Proceedings of the twenty-fifth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 153–162, New York, NY, USA, 2006. ACM.

[4] Paola Alimonti and Viggo Kann. Hardness of approximating problems on cubic graphs. In *CIAC '97: Proceedings of the Third Italian Conference on Algorithms and Complexity*, pages 288–298, London, UK, 1997. Springer-Verlag.

[5] Sanjeev Arora, L Babai, Jacques Stern, and Z. Sweedyk. The hardness of approximate optimia in lattices, codes, and systems of linear equations. In *IEEE Symposium on Foundations of Computer Science*, pages 724–733, 1993.

[6] Lars Backstrom, Cynthia Dwork, and Jon M. Kleinberg. Wherefore art thou r3579x?: anonymized social networks, hidden patterns, and structural steganography. In *WWW*, pages 181–190, 2007.

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

[8] A. Campan and T. M. Truta. A clustering approach for data and structural anonymity in social networks. In *Proceedings of the 2nd ACM SIGKDD International Workshop on Privacy, Security, and Trust in KDD (PinKDD'08), in Conjunction with KDD'08*.

[9] A. Chebotko, Seunghan Chang, Shiyong Lu, F. Fotouhi, and Ping Yang. Scientific workflow provenance querying with security views. *Web-Age Information Management, 2008. WAIM '08. The Ninth International Conference on*, pages 349–356, July 2008.

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

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

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

[13] Cynthia Dwork and Jing Lei. Differential privacy and robust statistics. In *STOC*, pages 371–380, 2009.

[14] Cynthia Dwork, Frank Mcsherry, Kobbi Nissim, and Adam Smith. Calibrating noise to sensitivity in private data analysis. In *In Proceedings of the 3rd Theory of Cryptography Conference*, pages 265–284. Springer, 2006.

[15] Cynthia Dwork, Moni Naor, Omer Reingold, Guy N. Rothblum, and Salil P. Vadhan. On the complexity of differentially private data release: efficient algorithms and hardness results. In *STOC*, pages 381–390, 2009.

[16] Uriel Feige. A threshold of ln n for approximating set cover. *J. ACM*, 45(4):634–652, 1998.

[17] Juliana Freire, Cláudio T. Silva, Steven P. Callahan, Emanuele Santos, Carlos Eduardo Scheidegger, and Huy T. Vo. Managing rapidly-evolving scientific workflows. In *IPAW*, volume 4145 of *LNCS*, pages 10–18. Springer, 2006.

[18] Arpita Ghosh, Tim Roughgarden, and Mukund Sundararajan. Universally utility-maximizing privacy mechanisms. In *STOC*, pages 351–360, 2009.

[19] Aleksandra Korolova, Rajeev Motwani, Shubha U. Nabar, and Ying Xu. Link privacy in social networks. In *CIKM '08: Proceeding of the 17th ACM conference on Information and knowledge management*, pages 289–298, New York, NY, USA, 2008. ACM.

[20] Carsten Lund and Mihalis Yannakakis. On the hardness of approximating minimization problems. *J. ACM*, 41(5):960–981, 1994.

[21] Ashwin Machanavajjhala, Daniel Kifer, Johannes Gehrke, and Muthuramakrishnan Venkitasubramaniam. L-diversity: Privacy beyond k-anonymity. *ACM Trans. Knowl. Discov. Data*, 1(1):3, 2007.

[22] Rajeev Motwani, Shubha U. Nabar, and Dilys Thomas. Auditing sql queries. In *ICDE*, pages 287–296, 2008.

[23] Rajeev Motwani and Prabhakar Raghavan. Randomized algorithms. *ACM Comput. Surv.*, 28(1):33–37, 1996.

[24] Shubha U. Nabar, Bhaskara Marthi, Krishnaram Kenthapadi, Nina Mishra, and Rajeev Motwani. Towards robustness in query auditing. In *VLDB '06: Proceedings of the 32nd international conference on Very large data bases*, pages 151–162. VLDB Endowment, 2006.

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

[26] Vibhor Rastogi, Michael Hay, Gerome Miklau, and Dan Suciu. Relationship privacy: output perturbation for queries with joins. In *PODS*, pages 107–116, 2009.

[27] Ran Raz. A parallel repetition theorem. *SIAM Journal on Computing*, 27:763–803, 1998.

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

[29] Vassilios S. Verykios, Elisa Bertino, Igor Nai Fovino, Loredana Parasiliti Provenza, Yucel Saygin, and Yannis Theodoridis. State-of-the-art in privacy preserving data mining. *SIGMOD Rec.*, 33(1):50–57, 2004.