

# An Analytical Survey of Provenance Sanitization

James Cheney and Roly Perera

{jcheney, rperera}@inf.ed.ac.uk  
School of Informatics, University of Edinburgh

**Abstract.** Security is likely to be a critical factor in the future adoption of provenance technology, because of the risk of inadvertent disclosure of sensitive information. In this survey paper we review the state of the art in secure provenance, considering mechanisms for controlling access, and the extent to which these mechanisms preserve provenance integrity. We examine seven systems or approaches, comparing features and identifying areas for future work.

## 1 Introduction

Automatically associating data with metadata describing its provenance has emerged as an important requirement in databases, scientific computing, and other domains that place a premium on reproducibility, accountability or trust [27]. Providing such metadata typically involves instrumenting a system with monitoring or logging that tracks how results depend on inputs and on other, perhaps untrustworthy, sources.

Publishing the entire provenance record associated with a computation is not always feasible or desirable. Disclosing certain information may violate security, privacy, or need-to-know policies, or expose sensitive intellectual property. Sometimes the complete provenance record may be too detailed for the intended audience, or may leak irrelevant implementation detail. But simply omitting some of the provenance information may leave it unable to certify the origins of the data product.

We refer to the general problem of ensuring that provenance solutions satisfy not only disclosure requirements but also security or privacy requirements as the problem of *provenance sanitization* or *provenance abstraction*. A number of approaches to provenance sanitization have been proposed recently [3,8,15,16,18], sometimes under other names such as *provenance views* or *provenance redaction*. These techniques have been developed mainly for scientific workflow systems, where provenance is viewed as a directed acyclic graph, as in the Open Provenance Model [28].

Existing approaches have several elements in common. Typically, an *obfuscation policy* specifies the aspects of the provenance which are to be hidden. A *disclosure policy* may additionally specify that certain other aspects of the provenance are to remain visible. *Sanitization* then involves transforming the provenance graph to obtain a view which satisfies both the obfuscation and the disclosure policies.

Few of the existing systems have been formally studied, and the security guarantees they actually provide are unclear. Some do provide formal guarantees, but are narrow in applicability or have other shortcomings. Moreover, many systems provide some form of security or confidentiality without considering the impact on the causal or explanatory role of provenance. In this paper we review the state of the art in



synchronisation which are beyond scope of the systems considered here. Some permit *composite* modules, i.e. modules that contain other modules.

A *provenance graph* is a directed, acyclic graph (DAG) recording the causal history of a data product. Often such a graph represents the (coarse-grained) *execution* of a software system, such as a workflow; more generally, provenance graphs can describe ad hoc processes or collaborations involving both human and software components. The nodes of the graph represent participants, actions and intermediate artifacts.

Fig. 1b shows a provenance graph that captures one possible execution of the workflow in Fig. 1a. The rectangular nodes, or *activities*, represent invocations of modules; the circular nodes  $d_1$  to  $d_9$ , sometimes called *entities*, record data values passed between modules. Moreover activities yield entities, and entities feed into activities; a graph that is partitioned in this way is called *bipartite*. Bipartiteness is just one of many possible design choices for graph-structured provenance; for example, one could add  $d_1, \dots, d_9$  as labels to the edges instead of using special nodes.

When a provenance graph represents a run of an iterative process, each module invocation must give rise to a distinct node, to maintain acyclicity. If necessary additional tags on the node label can be used to distinguish invocations of the same module.

*Sanitizing provenance graphs* The goal of provenance sanitization is to derive a *sanitized view* which hides or abstracts sensitive details of a provenance graph, whilst preserving some of its disclosure properties. Typically one wants the view itself to be a well-formed provenance graph. Fig. 2 below illustrates a simple provenance graph with two examples of views. On the right, tasks  $c_1$  and  $c_2$  have been abstracted into a single task  $c_3$ ; on the left, entities  $d_2$  and  $d_4$  and intermediate task  $c_2$  have been abstracted into a single entity  $d_5$ .

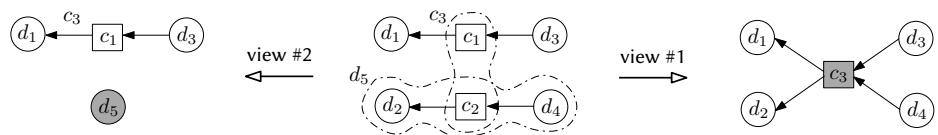


Fig. 2: Two possible views of a provenance graph

Both views are examples of *quotients*, arguably the simplest notion of graph view. One specifies a quotient of a graph  $G = (V, E)$  by giving a partitioning  $V' = \{V_1, \dots, V_n\}$  of its nodes. The partitioning induces another graph  $G' = (V', E')$  where there is an edge  $(V_i, V_j) \in E'$  iff there is an edge in  $G$  between a node of  $V_i$  and a node of  $V_j$ , for any  $i \neq j$ . In Fig. 2 the dotted border labeled  $c_3$  determines a partitioning if we consider each of the remaining nodes to inhabit a singleton partition; the dotted border labeled  $d_5$  determines a different partitioning, under a similar assumption.

Quotients are a natural forms of provenance view as they preserve *paths*, which represent relationships of direct or indirect dependency between nodes. If paths are preserved then related nodes are mapped to related nodes in the view; in other words, every dependency in the original graph gives rise to a dependency between the corresponding view nodes. Quotients preserve paths but not edges; for example the edges  $(d_4, c_2)$  and  $(c_2, d_2)$  have no counterpart in view #2 because all three nodes are mapped to  $d_5$ . Indeed edge-preservation, or *homomorphism*, is a stronger property than we use-

ally require for provenance sanitization, where dependency is assumed to be reflexive and transitive.

It can also be important to consider whether paths are *reflected*: whether nodes are related in the view *only if* there exist related nodes in the original graph which map to those nodes in the view. This too can be understood in terms of dependency, since it means that every reported dependency arises from a dependency between corresponding nodes in the original graph. Quotients do not in general reflect paths, because they coarsen the dependency relation: in view #1, for example,  $d_1$  now appears to depend on  $d_4$ , and  $d_2$  on  $d_3$ . This can be problematic if it violates cardinality constraints, such as a requirement that every artifact be generated by at most one activity [29].

## 4 Survey of techniques for provenance sanitization

In the ZOOM system of **Biton, Cohen-Boulakia and Davidson** [2,13], the user obtains a provenance view by first defining an abstract workflow view. A ZOOM workflow is a directed graph of atomic modules; a provenance graph is a DAG of invocations with edges labeled with runtime values. A workflow view is a quotienting which partitions the system into composite modules; for a given run of the workflow, the corresponding “quotient run” can then be obtained automatically by deriving invocations of each composite module from the invocations of its constituent modules.

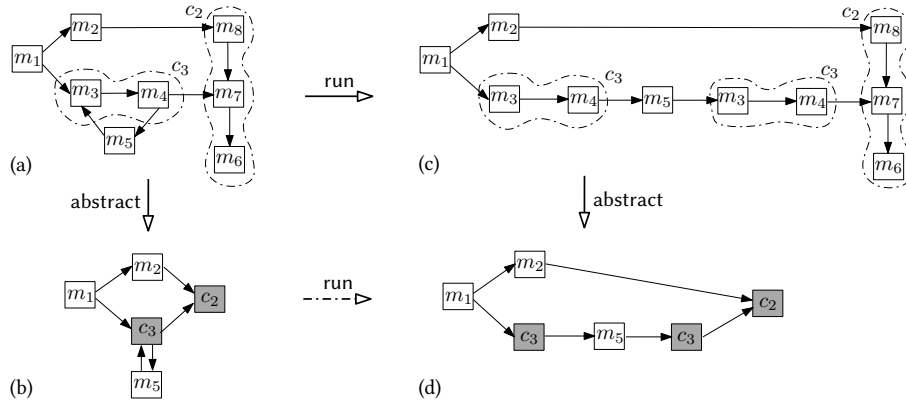


Fig. 3: ZOOM: deriving a provenance view from a workflow view

Fig. 3 illustrates the ZOOM approach. In Fig. 3a we see the original workflow with the partitioning identified by dashed borders labeled  $c_2$  and  $c_3$ . The modules  $m_1, m_2$  and  $m_5$  are assumed to be in singleton partitions. The induced workflow view is shown in Fig. 3b. Then, Fig. 3c shows an execution of the workflow with data labels omitted; here the dashed borders represent a partitioning of the *invocations* corresponding to invocations of the composite modules  $c_2$  and  $c_3$ . Fig. 3d shows the the corresponding quotient run where each node is mapped to its equivalence class.

ZOOM is not overtly motivated by security, but its views can be seen as abstracting away uninteresting parts of the graph while ensuring user-identified “relevant” parts remain visible. ZOOM is unique in respecting the semantic relationship between

program and provenance, as alluded to by the dotted run arrow relating Figs. 3b and 3d. Moreover being able to derive provenance views from *ex post facto* modularisations of a workflow is extremely powerful. However, it seems unlikely that their method for doing so (sketched only briefly in the papers) will generalise to workflows with non-trivial control flow or settings where submodules are shared by composite modules. In [13], most of the focus is on workflow views instead, in particular a method for deriving workflow views that preserve and reflect certain structural properties of the workflow, given a user-specified set of modules that are of interest.

The **security views** of **Chebotko, Chang, Lu, Fotouhi and Yang** [8] provide both access control and abstraction for scientific workflow provenance. Their workflows are DAGs with additional structure to model hierarchical tasks; the data channels of a composite task are those of its constituent tasks that cross the boundary of the composite task, relating composite tasks to the partitions of a quotient view. However, composite tasks are fixed features of the workflow rather than on-the-fly abstractions as in ZOOM, above. Being acyclic, workflows are unable to represent iteration.

To obtain a security view, one first specifies the accessibility of the various tasks and data channels, marking each element as accessible or inaccessible. Inheritance rules define the accessibility of an element if it is not given explicitly. Access control can be specified down to the level of individual ports; consistency constraints ensure that (for example) a data value inaccessible on one port is not accessible via another port. The access specification is then used to derive a provenance view from which inaccessible data values, tasks and channels have been removed.

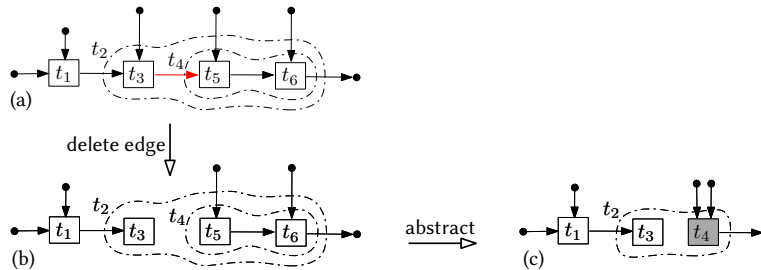


Fig. 4: Security views: combining abstraction with access control

Fig. 4a shows a run of a hierarchical workflow with two levels of composite task; both data nodes and ports have been elided for brevity. A node written as  $\bullet$  indicates an input or output. In Fig. 4b, the data channel between  $t_3$  and  $t_5$  has been deleted to conform to the access specification. Although dummy nodes, similar to the *surrogates* of Blaustein et al. below [3], may be added to the view to preserve well-formedness constraints, more general integrity requirements are not considered. For example once the edge between  $t_3$  and  $t_5$  has been deleted, the view no longer preserves dependencies, and so its ability to provide a full account of the output is compromised. Access control can however be combined with quotienting. In Fig. 4c the composite module  $t_4$  has been abstracted to a single node with two inputs, preserving the dependency structure of Fig. 4b, even though the latter view is unsound.

**Blaustein, Chapman, Seligman, Allen and Rosenthal** [3] present an approach based on *surrogates*. They define a *protected account* of a graph  $G$  to be any graph  $G'$ ,

along with a path-preserving function from the nodes of  $G'$  to the nodes of  $G$ . Since by definition every path in the view has an image in the original graph, a protected account necessarily reflects dependencies, but in general does not preserve them. Surrogates are a mechanism for publishing dependency information in a way that still protects sensitive nodes and edges.

Fig. 5a, adapted from [3], shows a typical graph with sensitive nodes and edges in red. Fig. 5b shows a protected account where  $e$  has been deleted and  $f$  replaced by a surrogate  $f'$ , shown with a dotted border, that hides its sensitive data (perhaps its identity). The view in Fig. 5c hides two more edges, breaking the indirect dependency between  $c$  and  $g$ . This is repaired in Fig. 5d by a surrogate edge (dotted arrow).

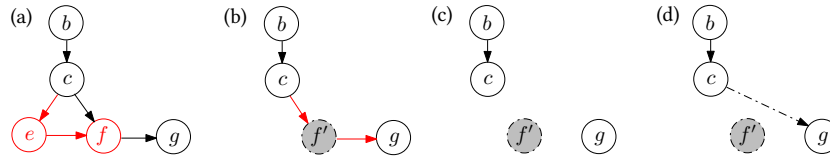


Fig. 5: Surrogates: provenance graph, plus three protected accounts

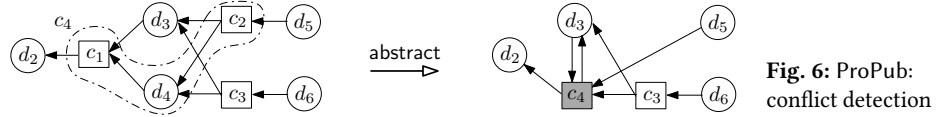
Blaustein et al.’s approach has three components: user privileges, which allow the graph provider to control graph access down to the level of individual ports; an algorithm for protecting graphs by deleting nodes and edges and adding surrogates; and metrics for analyzing disclosure and obfuscation properties of the resulting graph. For a given set of user privileges, their algorithm purportedly obtains a protected account which is “maximally informative”, according to a utility metric derived from the proportion of  $G$ -paths retained in  $G'$  plus the similarity of each node in  $G'$  to its counterpart in  $G$ . However definitions given are rather informal, and the theorems lack proofs, making this claim hard to evaluate.

Even when a protected account satisfies a particular obfuscation policy, an attacker may still be able to infer the original graph  $G$  from  $G'$ . To study this, Blaustein et al. introduce the notion of *opacity*, a measure of the difficulty of inferring an edge in  $G$  that is not present in  $G'$ , given a user-supplied model of the attacker. (The notion of opacity in the security literature [1] is somewhat different.)

The ProPub framework of **Dey, Zinn and Ludäscher** [18], based on Datalog, provides what the authors refer to as “policy-aware” provenance sanitization. A provenance query is expressed as a set of Datalog facts, asserting that the provenance for certain data items is to be disclosed, plus additional requirements relating to sanitization and disclosure. ProPub works directly with a provenance graph, which may not have been derived from an underlying workflow. A sanitization requirement might assert that certain data associated with a particular node is to be erased, that several nodes are to be abstracted into a single node, or that some nodes are to be deleted; a disclosure constraint might insist that a specific node is always retained in the view. In addition there will usually be global policies which hold across all queries (for example to outlaw “false dependencies” of the kind illustrated earlier in Fig. 2), as well as the usual well-formedness conditions such as acyclicity or bipartiteness.

A unique feature of ProPub is its ability to detect conflicts in the sanitization and disclosure requirements and to assist with their resolution. When conflicts arise, ProPub

uses a ranking scheme and various auto-correction strategies to resolve them, with the user also able to intervene to withdraw or modify a constraint in the light of the conflicts. For example in Fig. 6, adapted from [18], a naïve abstraction of three nodes into a single node  $c_4$  violates both acyclicity and bipartiteness:



In this case a possible resolution would be to include  $d_3$  into the abstraction as well, removing the cycle and restoring bipartiteness. Should applying a correction induce other conflicts, the process of conflict resolution continues. Only when a conflict-free variant of the query is obtained can a final sanitized view be derived. Any constraints rescinded during conflict resolution are reported alongside the sanitized view, providing a certain level of “meta-provenance”, also a unique feature amongst the systems considered here. For example, it might record that a spurious dependency was tolerated in order to accommodate an abstraction. ProPub’s logical foundation also means that the final view is guaranteed to have the chosen disclosure and security properties.

**Davidson et al.** [15,16] tackle a rather different problem with **provenance views**. Workflows are modelled as directed acyclic multigraphs (graphs with potentially more than one edge between any two nodes). Edges are labeled with identifiers called *attributes* which identify the port that the edge starts from; because workflows are acyclic, the semantics of a workflow can be given as a relation  $R$  over the set of all attributes, where each tuple consists of the data values that arise during a possible execution. (Equivalently, one can consider each tuple to be a labeling function assigning data values to ports.) In Fig. 7 below, adapted from [15], the workflow consists of three modules computing Boolean functions. Port  $a_4$  of  $m_1$  is consumed by both  $m_2$  and  $m_3$ . The relation  $R$  for this particular workflow is shown in the middle of Fig. 7. Effectively  $R$  is the natural join  $R_1 \bowtie R_2 \bowtie R_3$  of the relations  $R_1$ ,  $R_2$  and  $R_3$  capturing the *extension* (input-output mapping) of the modules individually.

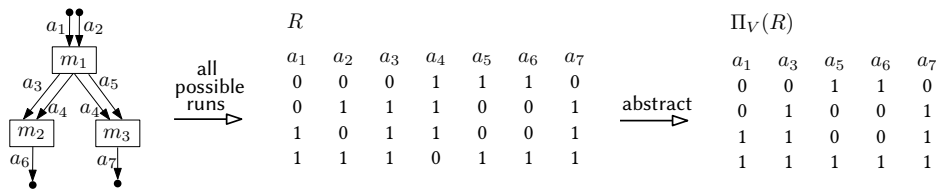


Fig. 7: Provenance views: hiding functional behaviour

Rather than hiding or abstracting parts of a particular run, Davidson et al. are interested in hiding the extension of a sensitive module  $m_i$ , namely the relation  $R_i$ , regardless of how many different executions the user observes. They classify modules as either public, whose behaviour is known *a priori*, or private, whose behaviour must be inferred by observing  $R$ . Their approach, which is quantitative, is based on an extension of  $\ell$ -diversity [24] which they call  $\Gamma$ -privacy. A view is specified by giving a set  $V$  of visible attributes. The relation  $\Pi_V(R)$ , the projection of  $R$  to  $V$  (Fig. 7, right),

defines the information that is publicly visible through  $V$ . For any positive natural number  $\Gamma$ , a private module is  $\Gamma$ -private with respect to  $V$  if for each input, the number of possible outputs from that module consistent with  $\Pi_V(R)$  is greater than  $\Gamma$ . With only this information, an attacker is unable to predict the output of the module for a given input with probability greater than  $1/\Gamma$ .

The first paper [15] studies some specific cases, including standalone private modules, multiple private modules, and heterogeneous workflows with a mixture of private and public modules where public modules can be “privatized” by renaming, so that their functional behaviour is no longer known. They show that standalone  $\Gamma$ -privacy is composable in a workflow consisting only of private modules. The authors also study the problem of finding minimum-cost views, given a cost function stating the penalty of being denied access to hidden attributes. The second paper [16] studies a more general solution for heterogeneous workflows, which involves *propagating* hiding, i.e. hiding attributes of public modules if they might disclose information about hidden attributes of private modules. They present a composability result generalizing the one for the all-private setting, to *single-predecessor* (that is, tree-like) workflows.

The privacy problem studied by Davidson et al. is interesting, but their work so far has a number of drawbacks. In particular, the PTIME bounds for the algorithms for mixed workflows [16] assume a fixed domain size, which in turn means that the size of relation  $R$  is treated as a constant. If we take the domain size  $d$  and number of attributes  $a$  into account, then the size of  $R$  is  $O(d^a)$ , so treating it as a constant may not be realistic. Moreover, it is also not always clear how to choose sensible values of  $\Gamma$ . For example, with a domain of  $1024 \times 1024$ , 8-bit grayscale images,  $\Gamma$  may need to be much higher than  $10^6$  to provide meaningful privacy, because changing a single grayscale pixel does not hide much information. (This criticism also pertains to other possibilistic definitions of security properties, such opacity [1] and obfuscation [9].) Techniques from quantitative information flow security [12], quantitative opacity [5] or differential privacy [19] may be relevant here.

The **provenance abstraction** approach of **Missier, Bryans, Gamble, Curcin and Danger** [26], implemented as ProvAbs, is based on graph quotienting and finding partitionings that satisfy both security needs and well-formedness constraints. Their provenance graphs follow the PROV model [29] and its associated constraints specification [10]. First, Missier et al. consider simple bipartite provenance graphs with node types representing activities and entities, and define three basic graph operations *pclose*, *extend* and *replace*. Intuitively, *pclose* takes a subgraph which is a candidate for replacement, and grows it until it is convex (there are no paths that lead out of the subgraph and back in again); *extend* further grows the subgraph until both its “input” nodes and its “output” nodes are homogeneous with respect to node type; and *replace* contracts such subgraphs to single nodes and adjusts edges to preserve paths.

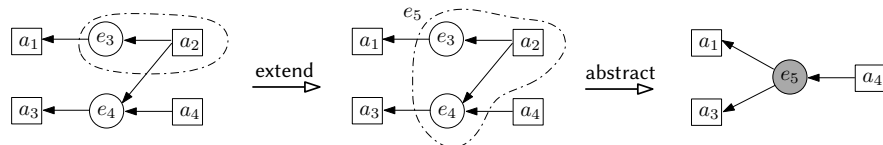


Fig. 8: ProvAbs: growing a partition so that abstraction preserves bipartiteness

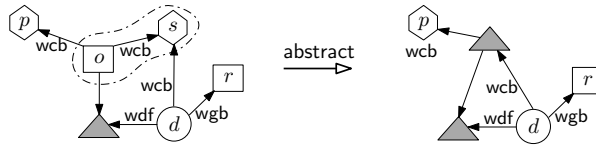


Fig. 8, adapted from [26], illustrates extend and replace. In Fig. 8a, the user selects activity  $a_2$  and entity  $e_3$  for abstraction. Replacing these two nodes by either an activity or an entity whilst preserving paths would violate bipartiteness. In Fig. 8b, extend is used to grow the target subgraph to include  $e_4$ , so that the output nodes of the target subgraph are uniformly entities. Replacing the subgraph by a single abstract entity  $e_5$  in Fig. 8c is now valid, although it coarsens the (transitive) dependencies by introducing a path between  $a_4$  and  $a_1$ .

Having shown how these transformations can be used to preserve basic validity constraints, Missier et al. go on to consider graphs which incorporate the PROV agent node type and associated relations such as attribution and delegation. They consider three cases of increasing sophistication. Grouping a homogeneous set of agents into a single abstract agent is relatively straightforward. Grouping agents and entities together is trickier; the type of the target abstract node (entity or agent) must be specified, and in order to maintain the type-correctness of certain relations between actions and agents (waw, “was associated with”) and between entities and agents (wat, “was attributed to”), the subgraph to be abstracted must be made larger. Finally, grouping arbitrary node types together presents the additional difficulty of agent-to-agent delegation edges (abo, “acted on behalf of”), which require similar treatment.

Like ProPub, a key feature of ProvAbs is that transformations operate directly on the provenance graph, and are thus more suited to situations where there is no underlying workflow. Missier et al. claim that their system avoids introducing spurious dependencies between nodes. However, their views are quotients, which in general over-approximate dependencies, so technically this claim is only correct for provenance applications where dependency is not required to be transitive.

The work of **Cadenhead, Khadilkar, Kantarcioglu and Thuraisingham** [7] on **provenance redaction** is also based on graph rewriting. Their provenance graphs are tripartite and conform to the Open Provenance Model’s labeled DAG format [28]. “Redacting”, or sanitizing, such a graph has two phases. First, the sensitive region  $G_Q$  (typically a single node or a path between two nodes) of the original graph  $G$  is isolated using a graph query  $Q$ . Then, this region of the graph is transformed according to an obfuscation policy expressed as rewrite rules. A rewrite rule has two components: a production rule  $r : L \rightarrow R$ , where  $L$  is matched against subgraphs of  $G_Q$ , plus an *embedding* specification, which determines how edges are to be connected to  $R$  once it has replaced  $L$ . The rewrites involve graph operations such as vertex contraction, edge contraction, path contraction and node relabeling.



**Fig. 9:** Provenance redaction: abstraction by edge contraction

In Fig. 9 above, adapted from [7], hexagons represent agents, rectangles represent processes, and circles represent artifacts. In the left graph, the gray triangle indicates an area of the graph that was previously redacted. On the right, a further subgraph is redacted by contracting the the wcb (“was controlled by”) edge relating a heart

operation  $o$  to the surgeon  $s$  who carried out the operation, and replacing the two nodes by another gray triangle.

Cadenhead et al.’s work is implementation-focused. Several formal definitions are given but not always made use of, and neither are their provenance or disclosure properties analyzed. One issue they do not appear to address, in contrast for example to Missier et al. (above), is preservation of basic well-formedness properties of the provenance graph. While edge contraction (as a particular kind of quotient) preserves dependencies, the interaction with tripartiteness is potentially problematic. For example in the view in Fig. 9, the new triangle has both an incoming *and* an outgoing wcb edge, because it subsumes both an agent and a process. Moreover, as the authors themselves point out, the obfuscation policy is only applied to a subgraph  $G_Q$  of the original graph  $G$ . Sensitive information available elsewhere in  $G$  will not be subject to the policy. Information flow techniques [17] may be relevant here.

## 5 Conclusions and future directions

We conclude our survey with a brief feature comparison, summarised in Table 1. The column headings refer to broad feature areas (discussed in more detail below); ● indicates reasonably comprehensive support for that feature, ○ little or no support, and ◐ somewhere in between. Necessarily this is a somewhat simplistic assessment.

System	Int	Dep	Acc	Qry	Sem	Form	Conf	Meta
ZOOM [2,13]	◐	●	◐	○	◐	○	○	○
Security views [8]	◐	◐	●	○	○	○	◐	○
Surrogates [3]	○	◐	●	○	○	○	◐	○
ProPub [18]	◐	◐	●	●	○	●	●	◐
Provenance views [15,16]	○	◐	○	◐	◐	○	○	○
ProvAbs [26]	●	●	●	◐	○	○	◐	○
Provenance redaction [7]	◐	●	●	●	○	○	○	○

Table 1: Feature comparison for the approaches surveyed

*Integrity.* We divide integrity features into basic integrity maintenance (**Int**) and integrity of causal or dependency structure (**Dep**). Even systems that make some effort to preserve the latter, such as provenance redaction, may in so doing violate low-level integrity constraints. In the future it seems likely that users will take low-level integrity for granted.

Preservation or reflection of dependency structure is more challenging because of the inherent tensions with obfuscation requirements. When arbitrary nodes or edges can be deleted, then the user may be responsible for repairing the damage, as with security views or surrogates. ProPub offers greater automation through conflict detection; ProvAbs and provenance redaction make safer (if simplistic) assumptions, by working mainly with quotient views.

*Sanitization.* Sanitization features range from explicit fine-grained access control (**Acc**), which all systems provide in some form or another, to query-based abstraction (**Qry**), as offered by ProPub and provenance redaction. Query-based systems typically subsume fine-grained access control, via fine-grained queries.

*Formal and semantic properties.* Few of the surveyed systems consider the problem of relating provenance views to the semantics of the underlying system (**Sem**). Instead, they operate directly on provenance graphs, without regard to how the graph was created. This is flexible, but means one cannot easily treat the provenance view as an (abstracted) account of how something was computed. ZOOM stands out in this respect, in relating provenance views to workflow views for simple kinds of workflow. On the other hand, this is a hard problem to solve in a general way.

Few existing systems provide formal guarantees of obfuscation or disclosure properties (**Form**). ProPub has the advantage of a solid logical foundation. The  $\Gamma$ -privacy of provenance views is a formal notion of (quantitative) opacity, but the goal is somewhat different from the other systems considered.

*Conflict detection and resolution.* As mentioned, ProPub stands out in being able to automatically detect conflicts between obfuscation and disclosure requirements (**Conf**), thanks to its logic-based approach. It is also the only system which makes conflict resolution an explicit and persistent part of the process, providing a certain level of “meta-provenance” for the sanitization process (**Meta**). If provenance security techniques are widely adopted, it seems likely that how provenance is manipulated to hide or reveal information will itself often be the point of interest (cf. “provenance of provenance” [29]).

Undoubtedly, controlling access to sensitive provenance metadata is of growing importance, and moreover we sometimes simply want to deliver provenance information at a particular level of detail. However, as the summary above highlights, current methods for provenance sanitization are immature. Future effort should focus on semantics, formal guarantees, and techniques for detecting and resolving conflicting policies.

*Acknowledgments* We are grateful to Jeremy Bryans, Brian Gamble, and Paolo Missier for comments on this paper. Effort sponsored by the Air Force Office of Scientific Research, Air Force Material Command, USAF, under grant number FA8655-13-1-3006. The U.S. Government and University of Edinburgh are authorized to reproduce and distribute reprints for their purposes notwithstanding any copyright notation thereon.

## References

1. R. D. Baillige and L. Mazaré. Using unification for opacity properties. In *Proceedings of WITS 2004*, pages 165–176, 2004.
2. O. Biton, S. Cohen-Boulakia, S. B. Davidson, and C. S. Hara. Querying and managing provenance through user views in scientific workflows. In *ICDE*, pages 1072–1081. IEEE, 2008.
3. B. T. Blaustein, A. Chapman, L. Seligman, M. D. Allen, and A. Rosenthal. Surrogate parent-hood: Protected and informative graphs. *PVLDB*, 4(8):518–527, 2011.
4. U. Braun, A. Shinnar, and M. Seltzer. Securing provenance. In *Proceedings of the 3rd conference on Hot topics in security*, pages 4:1–4:5, 2008.
5. J. Bryans, M. Koutny, and C. Mu. Towards quantitative analysis of opacity. In C. Palamidessi and M. Ryan, editors, *Trustworthy Global Computing*, volume 8191 of *Lecture Notes in Computer Science*, pages 145–163. Springer, 2013.

6. P. Buneman, A. P. Chapman, and J. Cheney. Provenance management in curated databases. In *SIGMOD 2006*, pages 539–550, 2006.
7. T. Cadenhead, V. Khadilkar, M. Kantarcioglu, and B. Thuraisingham. Transforming provenance using redaction. In *SACMAT*, pages 93–102, New York, NY, USA, 2011. ACM.
8. A. Chebotko, S. Chang, S. Lu, F. Fotouhi, and P. Yang. Scientific workflow provenance querying with security views. In *WAIM 2008*, pages 349–356, 2008.
9. J. Cheney. A formal framework for provenance security. In *CSF*, pages 281–293. IEEE, 2011.
10. J. Cheney, P. Missier, L. Moreau (eds.), and T. De Nies. Constraints of the PROV data model. W3C recommendation, W3C, April 2013.
11. S. Chong. Towards semantics for provenance security. In J. Cheney, editor, *TaPP 2009*. USENIX, 2009.
12. D. Clark, S. Hunt, and P. Malacaria. Quantitative analysis of the leakage of confidential data. *Electronic Notes in Theoretical Computer Science*, 59(3):238 – 251, 2002. QAPL 2001.
13. S. Cohen-Boulakia, O. Biton, S. Cohen, and S. Davidson. Addressing the provenance challenge using zoom. *Concurrency and Computation: Practice and Experience*, 20(5):497–506, Apr. 2008.
14. S. B. Davidson and J. Freire. Provenance and scientific workflows: Challenges and opportunities. In *Proceedings of SIGMOD 2008*, pages 1345–1350, New York, 2008. ACM.
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, T. Milo, and S. Roy. A propagation model for provenance views of public/private workflows. In *ICDT*, pages 165–176, New York, NY, USA, 2013. ACM.
17. D. E. Denning and P. J. Denning. Certification of programs for secure information flow. *Communications of the ACM*, 20(7):504–513, July 1977.
18. S. C. Dey, D. Zinn, and B. Ludäscher. ProPub: Towards a declarative approach for publishing customized, policy-aware provenance. In *SSDBM*, pages 225–243, 2011.
19. C. Dwork. Differential privacy. In *ICALP*, pages 1–12. Springer, 2006.
20. R. Hasan, R. Sion, and M. Winslett. Introducing secure provenance: problems and challenges. In *Proceedings of StorageSS 2007*, pages 13–18, New York, NY, USA, 2007. ACM.
21. R. Hasan, R. Sion, and M. Winslett. Preventing history forgery with secure provenance. *Trans. Storage*, 5:12:1–12:43, December 2009.
22. W. Lu, G. Miklau, and N. Immerman. Auditing a database under retention policies. *VLDB Journal*, 22(2):203–228, 2013.
23. J. Lyle and A. Martin. Trusted computing and provenance: better together. In *Proceedings of TAPP 2010*, Berkeley, CA, USA, 2010. USENIX Association.
24. A. Machanavajjhala, D. Kifer, J. Gehrke, and M. Venkitasubramaniam. L-diversity: Privacy beyond k-anonymity. *ACM Trans. Knowl. Discov. Data*, 1(1), Mar. 2007.
25. A. Martin, J. Lyle, and C. Namilkuo. Provenance as a security control. In *Proceedings of TaPP 2012*, pages 3–3, Berkeley, CA, USA, 2012. USENIX Association.
26. P. Missier, J. Bryans, C. Gamble, V. Curcin, and R. Danger. Provenance graph abstraction by node grouping. Technical Report CS-TR-1393, Newcastle University, 2013.
27. L. Moreau. The foundations for provenance on the web. *Foundations and Trends in Web Science*, 2(2–3), 2010.
28. L. Moreau, B. Clifford, J. Freire, J. Futrelle, Y. Gil, P. Groth, N. Kwasnikowska, S. Miles, P. Missier, J. Myers, B. Plale, Y. Simmhan, E. Stephan, and J. Van den Bussche. The OPM core specification (v1.1). *Future Generation Computer Systems*, 27(6):743–756, June 2011.
29. L. Moreau and P. Missier (eds.). PROV-DM: The PROV Data Model. W3C Recommendation REC-prov-dm-20130430, 2013.
30. J. Zhang, A. Chapman, and K. Lefevre. Do you know where your data’s been? — tamper-evident database provenance. In *Proceedings of SDM 2010*. Springer-Verlag, 2009.