Secure Provenance for Data Preservation Repositories

Isuru Suriarachchi, School of Informatics and Computing, Indiana University

***Abstract*—Importance of research data preservation and management has been accepted by the scientists all around the world. Interest and investment in data preservation projects has become higher than ever before. Already there are number of well-known research data repositories for different types of research data. Data preservation, sharing, discovery and reuse are the key features which are common across all such repositories. Data provenance is used to track lineage or processing history of a particular data product. Capturing provenance has been identified as an important step in any scientific application. Therefore, data preservation repositories are also utilizing provenance practices mainly to enhance data discovery. However, in some situations, the complete provenance information about datasets cannot be published in preservation repositories due to various possible reasons. Therefore, such repositories should facilitate mechanisms to control the amount of provenance information exposed for outside people. In this paper, we identify the scenarios in which the conflicts between obfuscation and disclosure of provenance exists in the context of data preservation repositories. We propose a secure provenance model which is capable of preserving provenance integrity while satisfying obfuscation requirements. We build our design based on SEAD [1] repository.**

***Keywords*—Secure provenance, Provenance integrity, Data preservation, Policy.**

I. Introduction

REPRODUCIBILITY of research experiments has been an issue for a long time due to lack of accessibility to input and output data, source codes and configurations used in those experiments. Research data preservation repositories have been introduced to solve this problem by preserving and managing research datasets. Good examples of such already available repositories are DataONE [2], Data Conservancy [3] and SEAD [1]. A research dataset in the context of preservation repositories can include any data related to the research including inputs, outputs, source codes, scripts, configuration parameters etc. A researcher can create a dataset by including all important data and upload it to a preservation repository. Inside the repository, the dataset is identified as a *Research Object* [4] and it can go through various management steps. Researchers may share their datasets with other researchers in the same group. And also, they may improve the dataset by editing related metadata. In some cases, external data curators may curate the dataset or convert them into different formats so that different research groups can make use of it. Finally, when the dataset is ready to be published, a unique Digital Object Identifier (DOI) is assigned for the dataset and it is archived with a publicly available URI. Most of the preservation repositories provide comprehensive search facilities to support discovery of published datasets. Researchers can search and find relevant datasets and reuse those for new experiments. They may also upload their results as new datasets at the end of the experiments.

Provenance is defined as the information about entities, activities, and people involved in producing a piece of data. Provenance can be used for many purposes like reproducibility, derivation of ownership, assessment of trustworthiness and failure tracing. Most of these usages are applicable in the context of data preservation repositories as well. Therefore, capturing provenance related to preserved data sets has been identified as a vital task in such repositories. We identify two different kinds of provenance capture for preservation repositories based on our SEAD [1] experience. As pointed out above, a dataset can go through number of curation steps performed by different researchers while it resides in the repository. We refer capture of provenance related to these curation steps as *curation time provenance* in this paper. Curation time provenance is important to track which events were performed on a given dataset by whom. Once a dataset is published, it can be downloaded and reused by some researcher. If a derived version of the same dataset is re-uploaded later, a provenance relationship can be inferred between these two data sets. We refer capture of such provenance among published datasets as *published object provenance*. Published object provenance plays a vital role in data discovery. For example, SEAD makes the provenance information available as a graph along with the published dataset. When a researcher searches for datasets, she can see the entire lineage of the dataset by looking at the provenance graph and make decisions based on that.

Even though the disclosure of provenance as much as possible is useful for the researchers who use preserved data, data owners may not be willing to expose some provenance information in some situations. For example, datasets related to human subjects may contain data about private information of people. In such cases, researchers cannot make the datasets publicly available. However, they may still want to preserve the datasets and derive publishable datasets using those original datasets. When publishing those derived datasets, provenance related to the original confidential data set may need to be hidden in the provenance graph. In some other cases, researchers may want to hide the names of agents (people or organizations) involved in producing certain datasets due to policies. Likewise, there can be many restrictions in publishing provenance related to the datasets in preservation repositories. Therefore, repositories should be capable of obfuscating provenance information based on the needs of data owners.

Disclosing maximum amount of provenance information while enforcing obfuscation is a challenge as it can lead to many conflicts. In this paper, we identify the scenarios

in which the conflicts between disclosure and obfuscation exists in data preservation repositories. Based on SEAD project and its provenance collection system Komadu [5], we build a design to preserve provenance integrity while dealing with conflicts introduced by applying obfuscation requests on provenance graphs. Cheney and Perera [6] highlights the capability of ProPub [7] to automatically detect and resolve conflicts introduced by applying user customizations on workflow provenance traces. ProPub uses a logic-based approach to easily handle conflicts. In our work, we try to adapt some of the concepts used in ProPub for preservation repository provenance.

Remainder of the paper is organized as follows. In section II we discuss SEAD and Komadu provenance collection system in more details to set the background. Then in section III we identify provenance disclosure requirements and the need of obfuscation in the context of data preservation. We present our design based on SEAD in section IV and conclude in section V.

II. Background

The provenance related concepts and features that we introduce in this paper are applicable in data preservation repositories in general. However, we build our secure provenance design based on the SEAD architecture. SEAD project is still under active development. Therefore, some features that we discuss in this paper are not yet implemented in SEAD. In this section we discuss the SEAD architecture and the Komadu [5] provenance collection system in detail before we move onto our secure provenance design.

A. SEAD

SEAD [1] can be used by scientists to easily discover, manage, share and preserve research data in sustainability science. SEAD repository consists of three major components: *Project Spaces* or *Active Curation Repository*, *Virtual Archive* and *Research Network*. These components are interconnected and can be used in different stages in the dataset life cycle.

SEAD Project Spaces (PS) can be used to manage active data. This component is a secure team-controlled space for managing data. A group of researchers can collaborate through a SEAD project space. Researchers can be added to and removed from a project space. And also, different roles can be assigned for different people within a project. Researchers can upload any kind of data into the project in any format. Collections can be created to group a set of data files together. SEAD PS can automatically extract metadata from the uploaded files. Data curation is a main feature of PS where the researchers can add or edit metadata on data files and collections through the user interface. Relationships between datasets can also be added by the researchers who own the datasets. In addition to that, SEAD PS supports number of social features like commenting, view counts, download counts etc.

SEAD Virtual Archive (VA) acts as the data preservation layer in SEAD. Researchers can insert datasets into VA either by pulling data collections from SEAD PS or by uploading a local data collection in BagIt [8] format. A data collection in VA is also referred to as a research object (RO). While the collection is in VA, different data curators can edit metadata related to the data in the collection. In addition to that, they may perform advanced curation steps like converting data into different data types. Once the collection is ready to be published, it can be deposited into an institutional repository for long term preservation. SEAD VA acts as a federation layer across multiple institutional repositories where the destination repository is selected according to the publishing researcher. Once an RO is published, a DOI is assigned for it. SEAD VA offers an easy to use search functionality for data discovery where it provides results by searching through all connected institutional repositories.

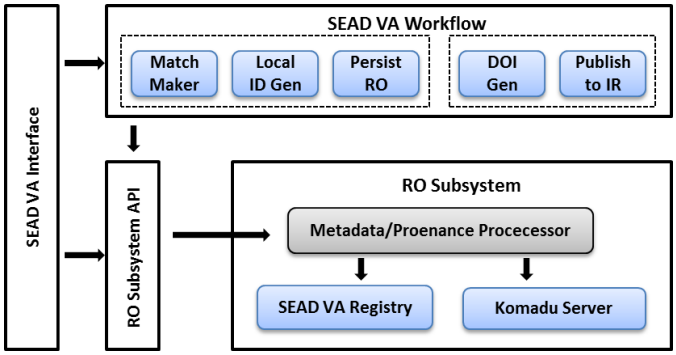


Fig. 1: SEAD Virtual Archive Architecture

SEAD VA captures both curation time provenance and published object provenance. Figure 1 shows the current high level architecture of SEAD VA. When an RO is ingested into VA either by pulling from SEAD PS or by uploading a BagIt package, it goes through the SEAD VA workflow. RO Subsystem is responsible of handling metadata and provenance. SEAD VA Registry component stores all metadata about each data item in the research object. Komadu is the provenance collection system which is used to store all types of provenance information in SEAD VA. When a curator edits metadata in some collection, those actions are stored as provenance information in Komadu. When a new RO is uploaded, SEAD VA checks whether it is a derived version of some already existing RO in the system. If such a relationship is found, that is also stored in Komadu. By processing all such relationships, Komadu generates a lineage graph for each RO which is displayed along with the RO when searched by some researcher. By looking at the lineage graph, the researcher can get an idea about the origin of the dataset and see what datasets contributed in deriving the current dataset.

SEAD Research Network maintains profiles of number of researchers in sustainability science. Researchers can link their publications and research data sets with their profiles in the network. All the co-authors of a publication can be linked with the publication page and that creates a research network.

B. Komadu

Komadu [5] is a W3C PROV [9] compliant provenance collection system which is used by SEAD VA. Figure 2

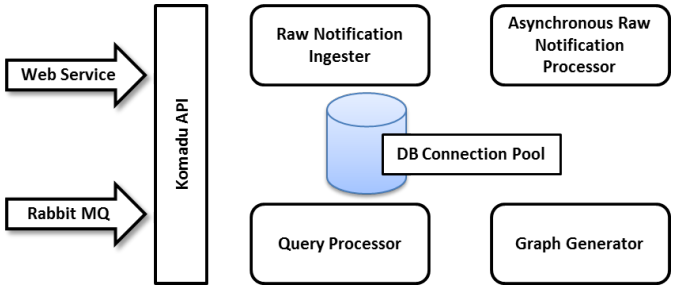


Fig. 2: Komadu Architecture

shows the high level architecture of Komadu. To collect provenance, a client can send runtime notifications into the system using either the Web Service API or the the Rabbit MQ [10] messaging API. Incoming raw notifications are stored in the database as it is by the Raw Notification Ingester. The Asynchronous Raw Notification Processor module runs periodically and separates Activities, Entities, Agents and their relationships by processing the raw notifications. When a client issues a query to retrieve a provenance graph, Query Processor accepts the request and generates the graph using Graph Generator.

C. SEAD Features to be Implemented

As the SEAD project is still under active development, there are number of useful features which are not implemented yet. The Komadu system also can be improved with more important features. In this section, we discuss some of those features which will be implemented in the future. We build our secure provenance model for SEAD by considering the complete set of features to make sure we do not have to change the model when new features will be added in the future.

As described above, currently only SEAD VA collects provenance related to data collections. However, once a dataset is uploaded into SEAD PS, there can be number of curation steps done by researchers to enhance metadata. Capturing provenance related to these curation steps is important to check the complete curation history. This feature will be implemented by connecting PS component with Komadu. When a curator (agent in provenance terms) does some edit step on a data item (entity in provenance terms), it will be recorded in Komadu by adding an agent-entity relationship. When the dataset is pulled into VA, it will use the same set of identifiers for all entities in the dataset. Therefore, provenance captured in VA can be added to the provenance captured in PS.

Missier et al. [11] presents a way of stitching provenance traces generated by related workflows. That work has been done by the Scientific Workflows and Provenance Working Group [12] of DataONE [2]. We adapt the concept of stitching provenance traces into SEAD by letting researchers to upload their W3C PROV compliant provenance traces along with their datasets. For example, if a dataset contains an output data file which was generated by a workflow engine like Kepler [13], the researcher can upload the provenance trace generated by the workflow with the data file. The uploaded provenance trace is ingested into Komadu which combines it with in-repository provenance. As both PS and VA accepts datasets, this feature has to be implemented in both those components.

To support the above feature in SEAD, Komadu should be able to accept provenance traces and stitch them with the existing provenance information. Currently this feature is not supported by Komadu. This can be implemented by splitting the incoming W3C PROV compliant provenance trace into activities, entities, agents and their relationships and storing them in the same way Komadu stores processed notifications. When the first query for a particular root node comes in, the graph generator will build the graph from the scratch by reading the database. Therefore, the provenance information created by splitting the ingested graph will automatically be considered.

In addition to that, both Web Services and RabbitMQ communication channels for Komadu are not secured currently. Securing the channels and authenticating users are must have features for our secure provenance architecture presented in the next sections. User authentication in Komadu can be implemented by integrating a user store as a backend component and authenticating the requests against the users in the user store. For Web Services channel, we propose the Username Token over HTTPS and for RabbitMQ channel, we propose SASL authentication [14].

III. Disclosure and Obfuscation

There are two main motivations for provenance disclosure in preservation repositories. First is to enhance data discovery. Provenance history of a dataset plays an important role in conveying more information to the researchers who are searching for data. A complete provenance trace related to a published dataset should contain details about previous datasets which has been used to derive the current dataset and curation time provenance of each dataset in the graph. Researchers can check the origin and decide on the quality, reliability and trustworthiness of a dataset by looking at its provenance history. Second is to track provenance related to curation activities before a dataset is published to outside researchers. By looking at the curation provenance trace, researchers can see the edit history of each data item in the dataset.

Even though disclosing provenance as much as possible is useful for users, there are number of situations where provenance information must be obfuscated in the context of data preservation repositories. Sometimes researchers cannot share research data with external researchers due to privacy policies and other restrictions. Such situations are common in biomedical, social and behavioural sciences. But still they may be interested in preserving these datasets by uploading those into preservation repositories. In such cases, those datasets should not be searchable and provenance information related to those datasets should not be exposed to researchers out of the group. In some other cases, researchers may derive publishable datasets using confidential datasets. In such situations, once the derived dataset is published, its provenance trace should not expose sensitive provenance information of the original dataset. Therefore, the researcher should be able to control the

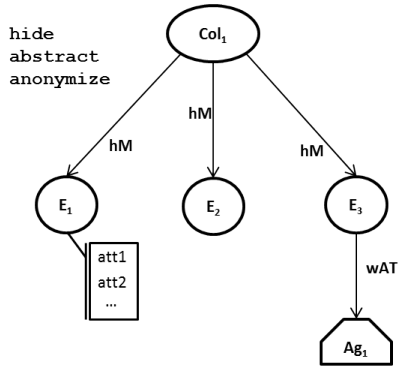


Fig. 3: Curation Time Provenance Window

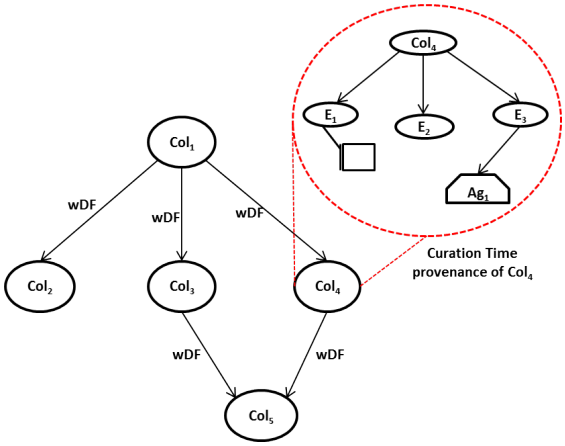


Fig. 4: Provenance Graph for a Published Data Collection

amount of provenance information to be published by using the tools provided by the repository.

As discussed above, allowing researchers to upload provenance traces with their datasets is important. However, researchers may want to hide some parts of the uploaded provenance traces due to various reasons. Those traces may contain sensitive information about the systems by which those are generated or may be too detailed for preservation needs. Therefore the researcher should be able to restrict the amount of information submitted to the preservation repository using some tool.

Considering these obfuscation scenarios, we adapt the following customization operations defined in [7] to be available in our SEAD secure provenance implementation. Researchers can use these operations to control the amount of provenance information published related to their datasets.

- **Hide:** Completely remove a node or an edge in the published provenance graph. In case of a node, all edges connected to the hidden node are also removed.
- **Anonymize:** Show only the existence of a node. Hide all attributes of the node. In case of a collection entity, do not allow to see the sub entities.
- **Abstract:** Create a single node by merging a selected set of nodes. Do not inherit attributes of the original nodes to the abstract node.

IV. Secure Provenance for SEAD

Here we present our secure provenance model based on SEAD. We use some of the concepts used in ProPub [7] here. However, in ProPub the entire provenance graph is owned by the user of the system. User can do any customizations to the provenance graph to have enough obfuscation. And also, the system can allow the user to modify her customization requests in case of a conflict. But in SEAD, different nodes of a published object provenance trace are owned by different researchers. Customization requests are submitted by the researcher before the dataset is published. System generates the provenance graph later when some user searches for data. Therefore, the system cannot alter any customization requests or get feedback from the user.

TABLE I: Customization Requests

`hide(N)`	Completely hide node N
`anonymize(N)`	Show only the existence of node N, hide details
`hide_edge(`N_1`, `N_2`)`	Completely hide the edge between nodes N_1 and N_2
`abstract(N, G)`	Replace all nodes mapped to group G with node G

A. Provenance Related Features

In the proposed model, both SEAD PS and VA comes with a curation time provenance window which displays provenance information related to unpublished data collections. As shown in Figure 3 each data item in the collection is shown as a separate entity which is a member (*hadMember* PROV relation) of the collection entity. The curation steps performed on individual entities by researchers (Agents) are also displayed. When a researcher submits an existing provenance graph with a data collection, that is also merged with in-repository provenance and displayed on the curation time provenance window.

SEAD VA displays a complete provenance graph for each published data collection. Each node in the graph represents a data collection. Edges represent the derivation relationships among collections. User can expand a node to see curation time provenance for a particular data collection. Figure 4 shows an example provenance graph for a published data collection. In this sample, Col_1 is the collection for which the graph was generated. Three other collections have been used to derive (*wasDerivedFrom* PROV relation) Col_1. If the user clicks on any collection, she will be shown the curation time provenance graph as shown for Col_4 in Figure 4.

Curation time provenance windows on both SEAD PS and SEAD VA allows users to make customization requests. Customization requests can be made either on the entire collection entity or on any internal node or edge of the graph. Table I shows the set of customization requests supported by SEAD. For example, if the owner of Col_1 in Figure 3 wants to hide the entire collection in any published provenance graphs, she can select the Col_1 node and select `hide` from the operations. In

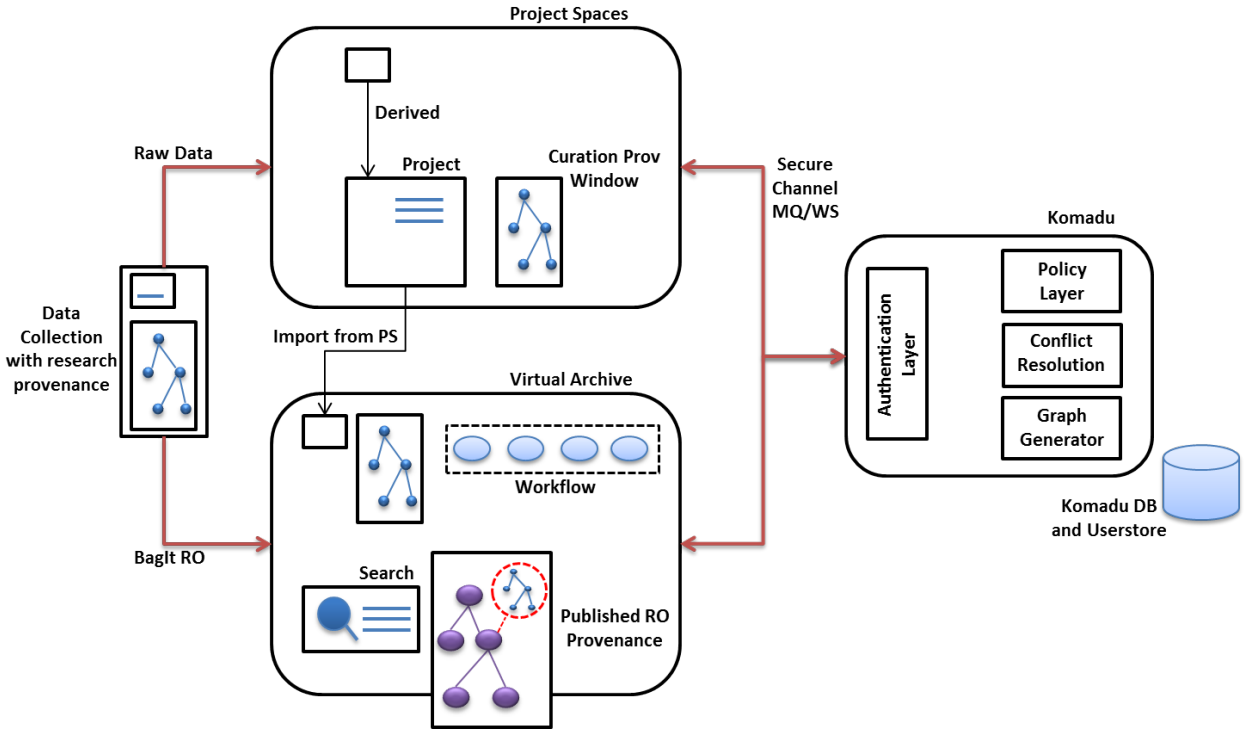


Fig. 5: New SEAD Architecture for Secure Provenance

that case, if some dataset (say Col_2) derived from Col_1 will be published in the future, provenance of Col_1 will not be shown on the provenance graph for Col_2.

SEAD adheres to a set of policies which makes sure the customized provenance graphs are correct and complete. These policies are set at the system level and can not be altered by users. When the graph is generated, customization requests from the users are applied on the original graph first. Then the system checks whether the customized graph adheres the set of polices.

NWC: No Write Conflict. Any entity cannot have more than one *wasGeneratedBy* relationships. In other words, only one activity can generate a given entity.

NC: No Cycles. There cannot be any cycles in the provenance graph. SEAD supports only directed acyclic graphs.

NFD: No False Dependencies. The customized graph cannot have any transitive dependencies which were not there in the original graph.

NFI: No False Independencies. The customized graph cannot have any independent nodes which were transitively dependent in the original graph.

Whenever there are conflicts between customization requests and polices, SEAD internally tries to repair those. When the conflicts cannot be repaired, the priority is given to customization requests as SEAD must honour privacy requirements of the researchers. In such cases, some polices have to be violated and the violated polices are displayed along with the customized graph.

B. Architecture

Figure 5 shows the new SEAD architecture which supports secure provenance. As mentioned above, researchers can upload raw data into SEAD PS or they can upload data in BagIt [8] format into SEAD VA. In both cases, they are given the option of including W3C PROV compliant provenance graphs in their submission.

When a dataset is submitted to SEAD PS, relevant entities for the collection and sub entities are created in Komadu. Provenance information related to all curation steps performed on the dataset are also captured by creating relationships in Komadu. Provenance graphs submitted by the researcher are also sent to Komadu and stitched to the same provenance graph. Curation provenance window in PS displays the complete provenance graph by pulling from Komadu. Researchers in the same project group are allowed to submit customization requests using the curation provenance window as shown in Figure 3. Customization requests are stored in Komadu to be used in future graph generation for published ROs. Researchers may create derived datasets in PS. For example to create a publishable dataset out of a private dataset. In such cases, that derivation relationship also stored in Komadu.

Curators in SEAD VA with proper permission to access collections in PS can import data collections into VA. In that case, all entity identifiers are passed to VA such that VA can pull provenance information from Komadu. When some researcher uploads a BagIt package into VA, new entities are created in Komadu. In both cases, the curation provenance

TABLE II: Additional operations in Komadu API

`addPolicy(P)`	Add a set of policies P to be adhered
`hide(N, t)`	Hide node N of type t (Agent, Entity or Activity)
`anonymize(N, t)`	Drop all attributes of node N of type t (Agent, Entity or Activity)
`hide_relationship(`N_1`, `N_2`, t)`	Hide the edge between nodes N_1 and N_2 of type t (derivation, attribution etc.)
`abstract(N, G)`	Replace node N with node G, drop attributes of N
`getActivityGraph(i, c)`	Return graph for activity i, with customized true or false
`getEntityGraph(i, c)`	Return graph for entity i, with customized true or false
`getAgentGraph(i, c)`	Return graph for agent i, with customized true or false

window in VA displays the complete provenance graph. Like in PS, curators in VA can submit customization requests using the curation provenance window in VA. When the research object is published, internal identifiers for entities are included in the OAI-ORE [15] resource map. If the published research object is re-uploaded as a new research object (most probably after using for some other research) after modifications, VA identifies that the new research object is a derived version of the original published research object and creates the relationship in Komadu. When a scientist searches and selects a published data collection on SEAD VA, she can see the provenance graph for the published data collection as shown in Figure 5. The purple dots represent published collections. Users can click on those to expand and see curation time provenance as described above using Figure 4. This provenance graph honours all customization requests made by the owners of the displayed collections.

Both SEAD PS and VA communicate with Komadu using the secure Web Services channel. Users of PS and VA are managed at the SEAD level and not propagated into Komadu. SEAD communicates with Komadu as a single system user. The new operations added to the existing Komadu API are listed in Table II. The set of policies adhered by SEAD is passed to Komadu on SEAD initialization. All customization requests coming from curation provenance windows are passed into Komadu and stored in the relational database. Provenance graphs for curation windows and published research objects are created by passing the entity id to Komadu. In case of published object graphs, SEAD VA instruct Komadu to return the customized graph instead of the full graph. When a customized graph generation request comes in, Komadu *Graph Generator* generates the graph by applying customization requests. Then the *Conflict Resolution* layer checks whether the customized graph adheres to SEAD graph policies. If not, the graph is repaired to make sure there are no conflicts between customizations are polices. If there are conflicts which cannot be repaired, priority is given to customization requests and the graph is returned with the set of breached policies.

C. Graph Generation

In our secure provenance model for SEAD, we include the graph customization and conflict resolution modules in Komadu as those are general features which add more value for a provenance repository. This makes sure that these features are not limited to SEAD and can be used by any Komadu user.

Algorithm 1 Algorithm for applying customization requests

```
1:  // Input: Full provenance graph (V, E)
2:  // Output: Customized provenance graph (V, E)
3:
4:  for (v in V) do
5:      if (v.hide == true) then
6:          V.remove(v);
7:          R = get_related_edges(v, E);
8:          for (r in R) do
9:              E.remove(r);
10:         end for
11:     end if
12:     if (v.anonymize == true OR v.abstract == true) then
13:         remove_attributes(v);
14:         if (v.collection == true) then
15:             remove_members(v, V, E);
16:         end if
17:     end if
18:     if (v.abstract == true) then
19:         i = v.abstractNodeId();
20:         g = V.get(i);
21:         if (g == null) then
22:             g = new_node(i);
23:             V.add(g);
24:         end if
25:         V.remove(v);
26:         repair_edges(v, g, E);
27:     end if
28: end for
29:
30: for (e in E) do
31:     if (is_hidden_edge(e) == true)  then
32:         E.remove(e);
33:     end if
34: end for
```

In the existing Komadu implementation, the *Graph Generator* module is responsible of generating provenance graphs. Following the standard practice, it treats a graph *G* as a set of vertices *V* and a set of edges *E* where *G = (V, E)*. In our new design, we extend the *Graph Generator* module to apply customization requests on the complete provenance graph when a customized graph is requested. As mentioned above, the customization requests are stored in the same relational

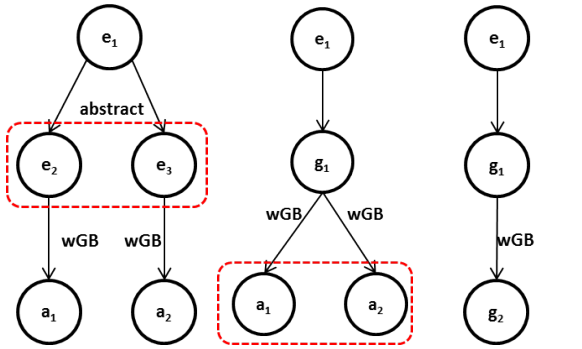
TABLE III: Policy validation rules

Transitive closures for G and G'
`tcdep'(X,Y) :- dep'(X,Y)`
`tcdep'(X,Y) :- tcdep'(X,Z), tcdep'(Z,Y)`
`tcdep(X,Y) :- dep(X,Y)`
`tcdep(X,Y) :- tcdep(X,Z), tcdep(Z,Y)`
Conditions for policy violations
`wc(X,Y) :- wGB'(A,X), wGB'(A,Y), not X=Y`
`cycle(X,Y) :- tcdep'(X,Y), tcdep'(Y,X), not X=Y`
`fi(X,Y) :- tcdep(X,Y), not tcdep'(X,Y)`
`fd(X,Y) :- tcdep'(X,Y), not tcdep(X,Y)`

database which is used to store provenance information in Komadu. Therefore, when the database is queried during the complete graph generation process, customization requests are also read and loaded into Java objects as properties. Komadu generates the full provenance graph first using the existing algorithm and then applies customizations. Algorithm 1 shows the customization algorithm which is used to apply customizations on the full provenance graph. The algorithm first loops through all vertices and looks for customizations. If there is a `hide` request (line 5) on current vertex, first it is removed from the set of vertices. The `get_related_edges` operation returns the subset *R* of edges out of all edges *E* which are connected to vertex *v*. Then all those edges are removed from the graph. If there is an `anonymize` request or an `abstract` request (line 12) on current vertex *v*, all attributes are removed from it. In addition to that, If *v* is a collection, all member nodes and their edges are removed to make sure that node is not further expandable. The subroutine `remove_members` takes care of that (line 15). If there is an `abstract` request (line 18) on current vertex, a new node *g* is created if it does not already exist in *V*. Original vertex *v* is removed from *V*. Then the subroutine `repair_edges` moves all edges connected to *v* to new node *g* (line 26). Finally the algorithm loops through all edges and removes the edges to be hidden (line 30).

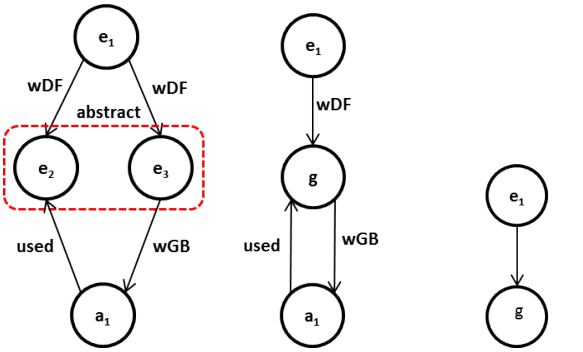
D. Conflict Resolution

The *Graph Generator* module returns both the complete provenance graph and the customized provenance graph. We refer the complete graph by *G* and customized graph by *G'*. The *Conflict Resolution* layer in Komadu gets both *G* and *G'*. First it checks whether there are any integrity violations in *G'* by applying the set of polices described above. Here we use the transitive closure technique used in ProPub [7] to detect dependency related policy violations. Table III shows the set of policy validation logic rules based on transitive closure. Komadu *Conflict Resolution* layer applies the rules on top of *G'* to detect any policy violations. Notation `tcdep'(X,Y)` denotes that node `X` and node `Y` are transitively dependent in *G'*. Notation `wGB'(A,X)` denotes that entity `X` *wasGeneratedBy* activity `A` in *G'*. After applying the logic rules on *G'*, if there are no policy violations found, *Conflict Resolution* layer immediately returns *G'* as the final customized provenance



(a) Full (b) Customized (c) Repaired

Fig. 6: Write Conflict Scenario

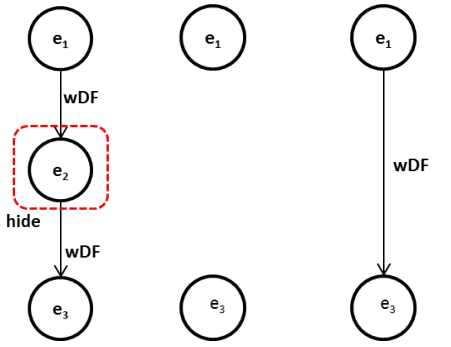


(a) Full (b) Customized (c) Repaired

Fig. 7: Cycle Scenario

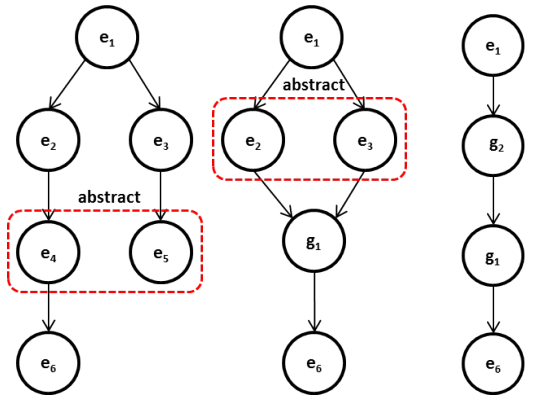
graph. If there are policy violations, it tries to repair them without losing much information exposed by the graph.

Figure 6 shows how Komadu handles a Write Conflict scenario. Original full provenance graph is shown in Figure 6a and the Write Conflict created by `abstract` user request is shown in Figure 6b where node g_1 is generated by two activities a_1 and a_2. The *Conflict Resolution* layer repairs the conflict by abstracting a_1 and a_2 with g_2 as shown in Figure 6c. Figure 7 shows how Cycles are handled. As shown in Figure 7a Cycles are also created by `abstract` customization request. In such cases, one node involved in the Cycle is an abstracted node. As shown in Figure 7c, other node participating in the Cycle (a_1) is also included in the abstraction node to resolve the Cycle. This is called *swallowing*. Figure 8 shows how False Independencies are created by `hide` customization request. In this case, the conflict is resolved by adding a dependency between e_1 and e_3 to re-introduce the dependency which was there in the original graph. However, the *Conflict Resolution* layer does not add this new edge if there is a `hide_edge` customization request between e_1 and e_3. In that case, the conflict cannot be resolved. Figure 9 shows how False Dependencies are handled. Such scenarios are



(a) Full (b) Customized (c) Repaired

Fig. 8: False Independency Scenario



(a) Full (b) Customized (c) Repaired

Fig. 9: False Dependency Scenario

occurred by abstracting two different dependency paths. In Figure 9b, e_3 is dependent on e_6 which was not the case in the original graph. As shown in Figure 9c, Komadu tries to resolve this by abstracting e_2 and e_3. This can create further False dependencies in more complex graphs. In such cases, the original conflict is considered unsolvable.

By applying the conflict resolution strategies on *G'*, Komadu generates a new graph which we refer by *G"*. As the next step, Komadu re-applies the policy validation rules in III on *G"*. If there are no policy violations, *G"* is returned as the final customized graph. If there are new conflicts introduced by the conflict resolution process, Komadu does not try to resolve them further as recursive application of the conflict resolution process removes lot of information from the provenance graph. Therefore, Komadu selects the graph with minimal policy violations out of *G'* and *G"* and returns it with the set of violated policies. However, Komadu always makes sure that all customization requests are satisfied in the returned provenance graph.

V. Conclusion and Future Work

In this paper, we have presented a secure provenance model for preservation repositories. First we identified the scenarios in which secure provenance is important in data preservation. Disclosure of maximum available provenance information is important for researchers in data discovery. However, there are many cases where data producing researchers and curators want to obfuscate provenance information due to various reasons. Based on SEAD architecture, we have presented our secure provenance model to satisfy both ends while preserving provenance integrity. We used a policy-based approach to resolve conflicts introduced by user requested customizations on provenance graphs. Our model always honours all customization requests to make sure the privacy of research data is preserved.

Implementing this secure provenance model on SEAD as a future work will immensely help the researchers who are using the system. Researchers who have sensitive data will not hesitate to use the system as the data and provenance security is always preserved.

References

[1] Sustainable Environment Actionable Data. [Online]. Available: http://sead-data.net/

[2] Data Observation Network for Earth. [Online]. Available: https://www.dataone.org/

[3] Data Conservancy. [Online]. Available: http://dataconservancy.org/

[4] S. Bechhofer, J. Ainsworth, J. Bhagat, I. Buchan, P. Couch, D. Cruickshank, D. D. Roure, M. Delderfield, I. Dunlop, M. Gamble, C. Goble, D. Michaelides, P. Missier, S. Owen, D. Newman, and S. Sufi, "Why linked data is not enough for scientists," in *Proceedings of the 2010 IEEE Sixth International Conference on e-Science*, ser. ESCIENCE '10. Washington, DC, USA: IEEE Computer Society, 2010, pp. 300–307. [Online]. Available: http://dx.doi.org/10.1109/eScience.2010.21

[5] Komadu Provenance Collection Tool. [Online]. Available: http://d2i.indiana.edu/provenance_komadu

[6] J. Cheney and R. Perera, "An analytical survey of provenance sanitization," *CoRR*, vol. abs/1405.5777, 2014.

[7] S. C. Dey, D. Zinn, and B. Ludäscher, "Propub: Towards a declarative approach for publishing customized, policy-aware provenance," in *Proceedings of the 23rd International Conference on Scientific and Statistical Database Management*, ser. SSDBM'11. Berlin, Heidelberg: Springer-Verlag, 2011, pp. 225–243. [Online]. Available: http://dl.acm.org/citation.cfm?id=2032397.2032414

[8] A. Boyko, J. Kunze, J. Littman, L. Madden, and B. Vargas, "The bagit file packaging format (v0. 97)," *Washington DC*, 2011.

[9] W3C PROV Specification. [Online]. Available: http://www.w3.org/TR/prov-dm/

[10] Rabbit MQ. [Online]. Available: http://www.rabbitmq.com/

[11] P. Missier, B. Ludascher, S. Bowers, S. Dey, A. Sarkar, B. Shrestha, I. Altintas, M. Anand, and C. Goble, "Linking multiple workflow provenance traces for interoperable collaborative science," in *Workflows in Support of Large-Scale Science (WORKS), 2010 5th Workshop on*, Nov 2010, pp. 1–8.

[12] DataONE Scientific Workflows and Provenance Working Group. [Online]. Available: https://www.dataone.org/working_groups/scientific-workflows-and-provenance-working-group/

[13] B. Ludäscher, I. Altintas, C. Berkley, D. Higgins, E. Jaeger, M. Jones, E. A. Lee, J. Tao, and Y. Zhao, "Scientific workflow management and the kepler system: Research articles," *Concurr. Comput. : Pract. Exper.*, vol. 18, no. 10, pp. 1039–1065, Aug. 2006. [Online]. Available: http://dx.doi.org/10.1002/cpe.v18:10

[14] SASL Authentication for RabbitMQ. [Online]. Available: https://www.rabbitmq.com/authentication.html

[15] Open Archives Initiative Object Reuse and Exchange. [Online]. Available: http://www.openarchives.org/ore/