Unlinkability through Access Control: Respecting User-Privacy in Distributed Systems

Apu Kapadia[*], Prasad Naldurg, Roy H. Campbell
University of Illinois at Urbana-Champaign, Urbana, IL, USA
{akapadia, naldurg, rhc}@cs.uiuc.edu

Abstract

We propose a policy-based framework using RBAC (Role Based Access Control) to address the unlinkability problem in the context of correlating audit records generated from access to distributed services. We explore this problem in an environment where the enforcement of access control policies is decentralized and ensuring policy consistency as the protection state of the system evolves becomes important. We introduce the notion of an audit flow associated with a user's access transactions, which represents the flow of information through audit logs within an administrative domain. Users of our system can present a set of audit flows to a decision engine that uses global access rules to detect potential linkability conflicts. Users can use this information to specify discretionary unlinkability requirements, depending on whether these accesses can expose sensitive attributes. We present an algorithm that can generate policy constraints based on these discretionary requirements. We also show how these policy constraints can be attached to individual audit log records to enforce unlinkability in a distributed manner. We prove that our proposed algorithm generates constraints that are secure and precise under strong tranquility assumptions with respect to the system's protection state. When we relax these assumptions, we show how versioning can cope with evolving protection state, trading off precision to maintain the security of deployed policies.

[*]Apu Kapadia is funded by the U.S. Dept. of Energy's High-Performance Computer Science Fellowship through Los Alamos National Laboratory, Lawrence Livermore National Laboratory, and Sandia National Laboratory.

1. Introduction

We examine the problem of preventing administrators from accessing (or "linking") multiple audit records corresponding to transactions initiated by the same user. Our problem is motivated by user identity and location privacy concerns in our university environment, where both physical access[1] to our facilities as well as virtual access to computing resources across our department and the wider campus are controlled by software. While the problem of loss of privacy when a user's actions are observed *in person* by a third party persists, the integration of physical and virtual access control mechanisms exposes a new concern. Users are now worried about other users being able to track their movement as well as their service-access behavior *remotely* by correlating system audit logs. These system audit logs are stored in various databases with independent access control mechanisms, making the enforcement of unlinkability a difficult task. While centralized mechanisms can solve such problems in theory, such approaches are not practical in our setting. They present a bottleneck for distributed access to resources and also provide a single point of failure for access control.

[1]Users have to swipe their i-card to gain access to the building, labs, offices etc. Each access attempt, along with the user-id and time stamp as well as the access decision, is recorded in a database.

Traditionally, *unlinkability* is defined as the infeasibility of an adversary to correlate two transactions initiated by the same user who does not reveal his/her identity, even when the user presents the same set of attributes to gain access. To address this problem, researchers have proposed a number of cryptographic mechanisms to construct anonymous credentials [3, 1, 2, 8] that make it computationally infeasible for a server to link the use of these credentials. However, even if a user presents an anonymous credential to access a service, the set of users allowed to possess those credentials in the first place may be small enough compromise anonymity. Furthermore, while many of these schemes rely on providing user anonymity, there are systems in which users cannot be anonymous. For example, an organization may be required to keep detailed audit records about who accessed payroll information by law. However, access to such information should only be provided to authorized users. In such systems, it becomes important to provide unlinkability through access control, allowing for linkability in only certain cases, e.g., legal subpoenas. We also note that cryptographic mechanisms are also vulnerable to collusion attacks between verifiers and issuers that correlate timing information for access logs [9], and adequate access control mechanisms can prevent such attacks.

In this context, we introduce **policy-based unlinkability** as the problem of restricting access to multiple audit records belonging to the same user, corresponding to multiple access transactions, which can be correlated to expose sensitive information. For example, two or more log records that can link a user Alice's identity with her location or other privacy sensitive attributes must not be accessible by administrative users unless they are explicitly authorized by a mandatory system policy or allowed by Alice. Our goal therefore is to provide a framework that can analyze conflicts, and change the authorizations (except when they are explicitly required by system policy) to access audit logs, and prevent such exposures. For example, the system may inform Alice that users in role Network Administrator can access information about her network transactions. There may be some Network Administrators who are also Students and Research Programmers. Based on Alice's privacy requirements, she can then request that Network Administrators who are Students be prevented from linking her audit records. In effect, the system allows Alice to negotiate a set of constraints to prevent certain administrative users from linking her transactions.

In this aspect, our work is related to the Separation of Duty (SoD) problem [17] in the context of RBAC, where different tasks and their associated authorizations are distributed among multiple users to prevent fraud and errors. Our problem is similar to the SoD problem in the sense that we are interested in engineering an appropriate set of roles with specific audit access authorizations subject to constraints. However, while the SoD problem is to prevent a single user from performing two or more related actions on *single* object in the context of a work flow, our problem is in preventing a single user from accessing *two or more* related data objects, across different audit flows. This distinction impacts the way in which unlinkability constraints can be enforced. With respect to the SoD problem, dynamic SoD constraints in the context of a work flow can be enforced by annotating the history of who accessed the object to the object itself, and preventing access to the same user [13]. This is not a viable option for our unlinkability problem as each object can only record its local history and cannot exchange this information with other related objects without explicit coordination.

Furthermore, the task of building a system that can accommodate for individual users' unlinkability constraints for any ordering of access transactions, with guarantees that extend over long periods of time, is formidable. It is not feasible for the security engineers to simulate all the possible access transaction scenarios for different users, identify unlinkability violations and enforce authorizations accordingly, especially when new services, users and modes of interaction are added periodically. To tackle this problem, we propose a policy engineering framework where a system entity called the Policy Negotiation Server (PNS) works with the user to refine the authorizations to access audit traces based on discretionary unlinkability concerns. The PNS collects information about different audit logs and their associated authorized users and stores it in a policy database. A user Alice can present to the PNS a set of access transactions constituting a session, ask the PNS to analyze these sessions for conflicts, and subsequently update this set by adding new transactions. We explore this problem in the context of RBAC [14, 4] (Role-Based Access Control) system, where users are assigned to roles, which are associated with a set of permissions. The PNS identifies conflicts and generates authorization constraints in terms of access restrictions on user roles, which can be attached to Alice's audit flow records in our system.

Assuming that we trust the underlying enforcement mechanisms, given a set of user transactions, we show how we can generate policy constraints that are both secure and precise with respect to enforcing unlinkability properties. We first prove these results under the strong tranquility assumption where the user-role assignments and permission-role assignments do not change over a session. Subsequently, we show how we can relax these assumptions and present an algorithm that uses versioning to handle changes in the authorizations under a weak tranquility assumption, sacrificing precision for the ability to change protection state. Using versioning we can always identify the set of users for which the policies are secure and precise. In both cases we show how users can add new flows to their sessions, refining their unlinkability requirements iteratively.

The rest of the paper is organized as follows: In Section 2 we present an architectural overview of system components and describe the basic policy negotiation protocol. Section 3 describes our approach, highlighting the assumptions, formulating the problem, and presenting our algorithm to generate policy constraints in detail. This section also presents a proof of how our algorithm is secure and precise under the strong tranquility assumption. Section 4 relaxes this assumption, and presents results for systems with evolving protection state. Section 5 presents related work and we present our conclusions in Section 6.

2. Architecture

In this section, we provide a brief overview of our system architecture. We introduce some terminology and outline the steps involved in policy negotiation shown in Fig. 1.

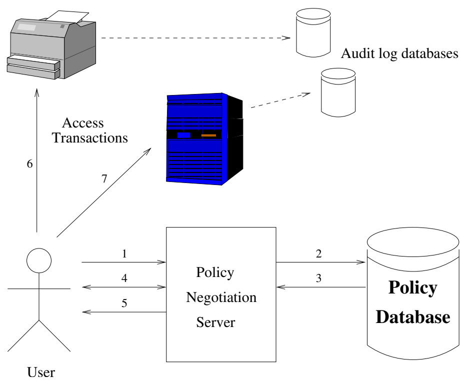


Figure 1. System Architecture

We assume a distributed system for sharing resources that allows us to specify and enforce system-wide access control policies. Users in the system access services by presenting credentials resulting in an *access transaction*. We focus on a single administrative domain, where users may engage in attribute-based authentication, possibly with cryptographically unlinkable certificates, although identity-based authentication may be used as well. Users negotiate unlinkability policies with a policy negotiation server (PNS) that generates policy constraints, which when enforced correctly prevents any exposure of sensitive information through audit log analysis by other administrative users.

Information related to an access transaction (e.g., audit logs for location tracking) is stored in one or more databases. We define an *audit flow* for a given transaction as the set of databases and their associated authorizations that define the possible flow of audit information to other users in the organization. A collection of transactions and their associated audit flows that a user desires to keep unlinkable is called a *session*. Sessions are associated with individual users and may be open-ended, i.e., they last for the lifetime of the system and users are allowed to update the list of transactions in their session. As mentioned earlier, users and access permissions are organized into roles using RBAC.

We now describe a high level overview of our system with the aid of Figure 1. (1) In the first step, a concerned user Alice sends her session information to the policy negotiation server (PNS). This is a set of unique identifiers corresponding to access transactions. In steps (2)-(3), the PNS looks up relevant information for each service including access policies and flow policies (replication of data between servers), builds the audit flows $I_1, \cdots, I_n$, and analyzes them for unlinkability conflicts. The PNS presents Alice with a set of roles whose users can access her audit log information from two or more audit flows.

In Step (4) Alice identifies her discretionary unlinkability requirements in terms of roles whose users she wants to prevent from linking her audit information. This approach places the burden of specifying useful privacy policies on Alice. In a real system, Alice can be guided to make an informed choice based on standard organizational policy. The granularity of access to records in a database can be finer, and Alice can specify requirements for specific fields in records. For simplicity of notation, we only model access to a database record as an all-or-nothing permission, and this can be extended to finer-grained permissions as necessary.

As mentioned earlier, the PNS may not be able to enforce some of Alice's choices if there are mandatory access requirements. After Alice and the PNS agree on the policies, in (5) the PNS sends the Alice a certificate with policy constraints for her audit records. This certificate is a digitally signed and can be tagged to Alice's audit data and sent to the databases as the information is generated. The PNS also stores Alice's discretionary policies and session information in the policy database. In Steps (6)-(7), for each access transaction, Alice presents these certificates, which are attached to audit records that make up the audit flows. These policy constraints enforce her unlinkability requirements for the session. We assume that all interactions are cryptographically secured for authenticity, confidentiality, and integrity.

Alice may update her session at any time and introduce new transactions, which may introduce new conflicts. The PNS generates new constraints in this situation and sends the updated credentials to Alice. We assume that Alice has no motivation to delete any transactions from her session. Therefore, we can guarantee that all old policies will still be honored and each successive iteration of our system will be a refinement of the original access restrictions. However, updating policy constraints every time there is a change in protection state can be prohibitively expensive. In Section 4, we show how we can maintain security by trading precision for the ability to change protection state, without impacting the privacy guarantees of previously issued policy constraints.

We deliberately choose to limit the functionality of our PNS as a decision engine that generates and distributes access constraints. Users (or agents working on behalf of the user) have to attach these constraints to their audit records. An alternative design would be to centralize policy decision making at the PNS and require that all databases contact the PNS every time an audit record is accessed, to evaluate whether it should be allowed or denied based on the current set of sessions and discretionary policies in the policy database. While this alternative design is more precise with respect to enforcement of the privacy policies, we believe it induces a performance overhead and may create a single point of failure.

Throughout this paper we will refer to three types of policies. *Flow policies* are explicit representations of data flows between databases. For example a policy such as (d_1, d_2) allows database d_1 to supply copies or transformations of data to d_2. The system can use these flow policies to construct graphical representations of audit flows throughout the system. *Access policies* are Permission-Role assignments (d, r), where role r may access database d. Lastly *policy constraints* are described in Section 3.5, and are attached to audit records. Access to an audit record is granted to users based on the access policy for that database, and the policy constraints of that audit flow, which can override the former.

An important point to note in our framework is that we address the unlinkability problem at the level of RBAC access permissions, and assume that the system is aware of the semantics of how the information stored in the audit logs can be linked to expose sensitive user information. However, we do not require that users are aware of the semantic relationships across audit logs. If a user is not aware of the relationships, our system can still enforce a default policy that is conservative and prevent anybody who has access to two or more flows from accessing audit records in both. Techniques such as information hiding, anonymizers and statistical mixing etc., address the unlinkability problem at the semantic level by modifying the contents of the databases. We believe that using access control to enforce unlinkability complements these other approaches and our solution explores the effectiveness of using access control permissions to perform audit flow analysis comprehensively, and prevent linkability in this context.

3. Approach

In this section we present our algorithm for constructing audit flow graphs and for generating policy constraints from these graphs based on the system's mandatory requirements and the user's discretionary unlinkability policies. Users in the system typically interact with various services over the lifetime of the system. The set of access transactions that a user wishes to keep unlinkable is referred to as a *session.*

Our first goal is to provide users with a set of mechanisms to negotiate and enforce unlinkability for all their transactions as specified in their session. We explain our system with regard to a particular user Alice who wishes to keep her transaction information unlinkable from other users in the system. For example, Alice may decide that her location information should not be linked with the use of physical services that she may access anonymously. Correlating logs with the location database would expose her identity. Hence, Alice may want to restrict the ability of other users in the system to access both audit flows, viz., location information and Alice's service audit trail. We also show how we can extend this protection when Alice iteratively adds transactions to her session, which we call "open-ended sessions."

One way to specify policy constraints to preserve unlinkability for Alice's session is to find all the users who can access information from two or more flows and restrict their access to sensitive audit records. We propose that data objects in an audit flow are tagged with these access policy constraints in terms of lists of users that are not authorized to view the data. We rely on the trustworthiness of the underlying access mechanisms to enforce these authorizations correctly. Since we assume that the system is working with the users to protect their privacy, access is granted only if these constraints are not violated.

One of the issues with this approach is that these access lists associated with audit flow records could become very large. Furthermore, changes in user permissions during the lifetime of the session will affect the validity of the policy constraints. To improve the compactness of policy representation and handle dynamically changing authorizations, we feel that an RBAC system can address these concerns effectively. In an RBAC system, users are removed and added to roles, and the permissions for roles will not be affected by these operations.

We now propose an approach to enforce unlinkability for Alice's transactions, defined by her session, by finding all possible roles that are explicitly granted read access to each audit flow. We then examine all the users in this set of roles and construct a set of *overlapping roles*, i.e., the set of roles that these users can activate in the system. Roles with a common user are overlapping roles. The main idea here is that if two audit flows have common overlapping roles, then the flows are potentially linkable. A common overlapping role indicates that there may be users with that role who can access both audit flows.

For example, say role *Administrator* can read audit flow I_1. Now suppose that there are two users u_1, u_2 in the system with the *Administrator* role that also possess the *Student* role, which is an overlapping role. Further, suppose that u_1 in the *Teaching Assistant* role can read audit flow I_2. Since *Student* appears in both audit flow graphs, it is a common overlapping role. Since there is a particular user u_1 in role *Student* who can access both flows, we call *Student* a *conflicting role*. An access policy constraint can be generated if Alice would like to prevent *Students* from linking her flows. Alice could specify "Do not allow any user whose role set contains all the roles $\langle \mathit{Student}, \mathit{Administrator}, \mathit{TeachingAssistant} \rangle$ to access audit flow records tagged I_1 and I_2." While this will prevent *Students* from accessing Alice's information, it will still allow other (non-*Student*) *Administrators* and *Teaching Assistants* to access I_1 and I_2 respectively. Furthermore, students who are not linkability threats (i.e., those who can access only one flow), will still be allowed to access Alice's audit records. A reference monitor enforcing access to the audit record database will check if a particular user has all three roles at the time of access and deny access appropriately. This is more concise than listing all the possible users that can link audit flows. We now formalize these concepts, and show how we can provide users with unlinkability with respect to audit flows. The key idea here is that Alice can specifically deny users of certain roles from linking her information.

3.1. Notation

Our system includes roles, databases, and users. In this paper we refer to these entities both in the context of general access control, and as vertices in graphs. For simplicity, we use the same notation for both contexts, instead of having separate "role vertices" for the corresponding roles, and so on.

3.2. Audit flow graph

Let the set of roles in the system be Γ, and the set of databases be Δ. Let $\mathcal{U}$ be the set of users in our system. Let URA and PRA be the user-role assignment and the permission-role assignment, defined according to standard RBAC terminology. $URA(u)$ returns all the roles that a user u can activate. Similarly, $PRA(r)$, returns all the permissions or accesses allowed to a role r.

An audit flow graph for an access transaction is a directed graph $I = (V, E)$ with the set of vertices $V \subseteq \Delta \cup \Gamma$, representing databases, roles, and overlapping roles. Overlapping roles are discussed shortly. A directed edge $(u, v) \in E$ indicates the flow of audit information from u to v. We identify the first database in the audit flow I_i of a given user as the *root* vertex δ_i for that flow.

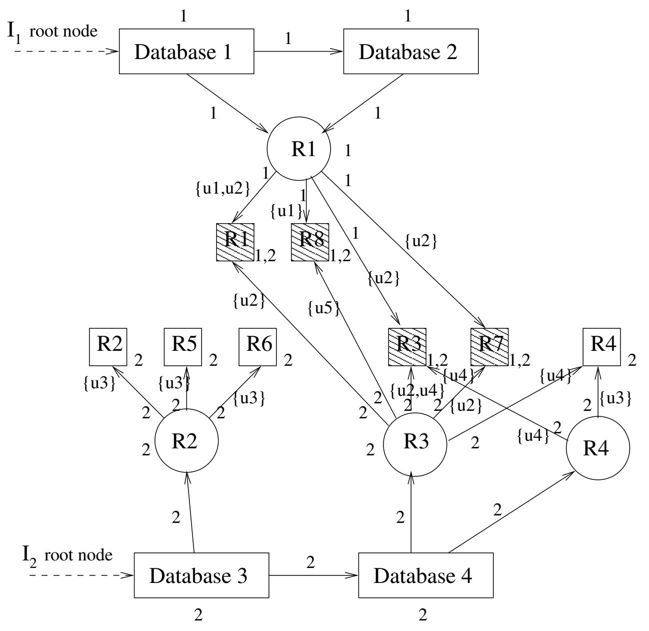


Figure 2. Session Graph

We now describe how to create an audit flow graph, given a root vertex that represents Alice's transaction, and show how to construct the combined session graph for multiple audit flows. Figure 2 represents an example session graph for two audit flows, which we will refer to for clarity. The audit flow graph I_i for transaction i for user u is constructed as follows:

1. Adding databases: The root vertex δ_i represents the start of the audit flow I_i. Starting from this vertex, we iteratively add vertices and edges corresponding to all databases that receive audit log information about the access transaction δ_i initiated by the user. This operation is repeated until all databases for the audit flow have been added to the audit flow graph. For databases d_1, d_2 we have $(d_1, d_2) \in E$ if and only if the audit flow information for that transaction flows from d_1 to d_2.

In Figure 2, databases are represented as rectangles. The root vertex for I_1 is *Database 1.* As information related to audit flow I_1 flows from *Database 1* to *Database 2*, we have a directed edge from *Database 1* to *Database 2.* Similarly, we have audit flow I_2 flowing from *Database 3* to *Database 4.*

2. Adding roles: For each database $d \in V$, determine the set of roles $R \subseteq \Gamma$ with read permission to database d. These roles are added to the audit flow graph vertices V, along with the edges (d, r) for each $r \in R$. We have $(d, r) \in E$ if and only if role r has permission to read database d.

In Figure 2, roles are represented as circles. The individual access policies of *Database 1* and *Database 2* allow read access to users with role $R1$. Hence we have directed edges from *Database 1* and *Database 2* to $R1$, and so on.

3. Adding overlapping roles: For each role $r \in V$, we generate the corresponding overlapping roles, and include directed edges to them. Let $O \subseteq \Gamma$ be the set of overlapping roles such that for every $o \in O$, some user u can activate role o in addition to r. We call r the *parent* of overlapping role o. We have $(r, o) \in E$ if and only if o is an overlapping role of r.

Consider the following URA for a system with five users u_1, u_2, u_3, u_4, u_5. $URA(u_1) = \{R_1, R_8\}$, $URA(u_2) = \{R_1, R_3, R_7\}$, $URA(u_3) = \{R_2, R_5, R_6\}$, $URA(u_4) = \{R_3, R_4\}$ and $URA(u_5) = \{R_3, R_8\}$ Figure 2 shows the overlapping roles overlapping roles (represented as squares). Role R_1 has overlapping roles $\{R_1, R_3, R_7, R_8\}$, R_2 has overlapping roles $\{R_2, R_5, R_6\}$, and so on. The user-sets on edges of overlapping roles show the users common to both roles.

We now examine the complexity of creating an audit flow graph for a given transaction. In Step 1, at most $|\Delta|$ new vertices can be added to the graph. For each vertex, at most $|\Delta| - 1$ new edges can be added. Therefore we are bounded by $O(|\Delta|^2)$ operations. In Step 2, for each database, at most $|\Gamma|$ role edges can be added to the graph. Therefore Step 2 is bounded by $O(|\Delta||\Gamma|)$ operations.

3.2.1 Constructing the AURA graph

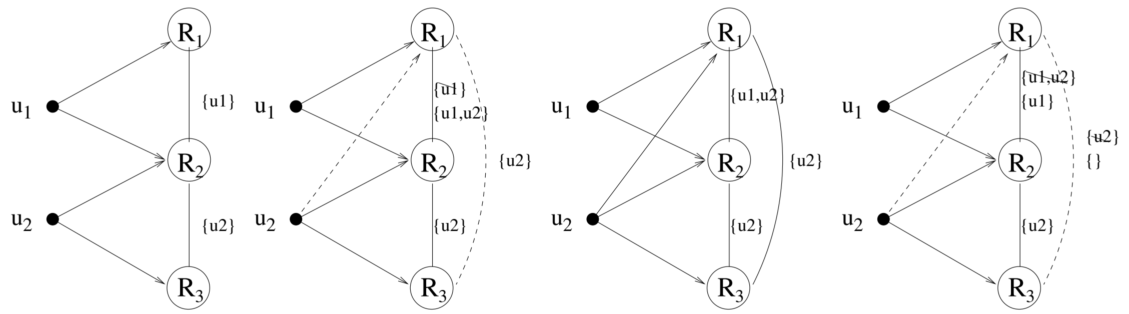
Step 3 involves generating overlapping roles. We show how we can amortize the cost of this step by augmenting a standard URA mapping to include overlapping role assignments. We call this the *AURA* graph (Augmented User Role Assignment graph). A directed edge (u, r) mean that user u is assigned to role r. An undirected edge (r_1, r_2) means that r_1 and r_2 are overlapping roles. Each undirected edge is associated with the set of overlapping users for roles r_1 and r_2, $U(r_1, r_2)$. In Section 3.3 we will use these user-sets to identify conflicting roles. A *conflicting role* is a role that contains one or more users who can access two or more audit flows within a session.

We show how we can use the AURA graph to maintain overlapping role information, and describe how to update an AURA graph when the protection state of the system changes:

Adding a Role: This operation does not create any extra overhead, since overlapping roles are not affected until a User-Role assignment changes.

Adding a User-Role assignment: If a User-Role assignment (u, r) is added, then for each of u's roles $r' \in URA(u)$, the undirected edges (r, r') are added unless these edges exist already, and u is added to the set $U(r, r')$. There are $|URA(u)|$ operations, which is bounded by $|\Gamma|$. The time complexity for set union for adding u to $U(r, r')$ is constant (using hash tables). For example, consider the AURA graph in Figure 3(a). We omit self-loops (r, r) with user-sets $U(r, r)$ equal to the set of all users in r. We add the assignment (u_2, R_1) as shown in Figure 3(b). We must now update edges (R_1, R_1), (R_1, R_2) and (R_1, R_3), resulting in three operations on the AURA graph. Since $URA(u) = \{R_1, R_2, R_3\}$, we have $|URA(u)| = 3$ as expected. The resulting AURA graph is shown in Figure 3(c).

Removing a User-Role assignment: If a User-Role assignment (u, r) is removed, then for each of u's roles $r' \in URA(u)$, u is removed from $U(r, r')$. If $U(r, r') = \emptyset$, the edge (r, r') is removed. There are $|URA(u)| + 1$ operations, which are bounded



(a) Current AURA Graph (b) Adding User-Role assignment (u_2, R_1) (c) New AURA Graph (d) Removing User-Role assignment (u_2, R_1)

Figure 3. AURA Graph example

by $|\Gamma|$. Again, removing u from $U(r, r')$ can be done in constant-time with the use of hash tables. In our previous example we added the assignment (u_2, R_1), resulting in the AURA graph shown in Figure 3(b). To remove this assignment, we must update the edges (R_1, R_1), (R_1, R_2) and (R_1, R_3), as shown in Figure 3(d). Here $URA(u) = \{R_2, R_3\}$ since the assignment (u_2, R_1) was removed, and we have $|URA(u)| + 1 = 3$ as expected. The resulting AURA graph is the same as the original AURA graph in Figure 3(a).

Removing a Role: Each User-Role assignment must be removed first. Let U be the set of users for the role being removed. Hence we have $|URA(u)|$ operations for each user $u \in U$. This is bounded by $|\mathcal{U}||URA(u)|$.

Therefore, this approach requires at most $|URA(u)|$ operations on the AURA graph for each addition/deletion of a User-Role assignment. In the worst case this is $O(|\Gamma|)$ operations for each addition/deletion of a User-Role assignment. Deleting a role in the system is more expensive and is bounded by $O(|\mathcal{U}||\Gamma|)$. Overlapping roles for any particular role can be efficiently extracted from the AURA graph by a simple lookup.

3.3. Session graph

While individual audit flow graphs capture the dissemination of log information to authorized users in an organization, users are interested in exploring how their sensitive attributes can be exposed by log correlation across these databases. Given a set of audit flows $\{I_1, \cdots, I_n\}$, corresponding to a set of transactions that user Alice may execute, we define session graph S by constructing a composite graph which includes each audit flow graph that was constructed as described in Section 3.2. The set of vertices and edges in the composite graph is the union of the sets of vertices and edges in the original audit flow graphs. However, we preserve the information about distinct flows in this composite graph by augmenting edges with colors as described next.

In order to represent overlapping nodes and edges between these graphs and identify linkability conflicts, we introduce the mapping $Color : I_i \to \mathbb{N}$, which identifies a unique natural number with each audit flow. For simplicity, we assume that edges $e_i \in E_i$ from I_i are assigned color i, i.e., $Color(I_i) = i$. An edge $e_s \in S$ may therefore have multiple colors, reflecting which flow it belongs to for each color. We define the colors for a vertex $v_s \in V_S$ as $Colors(v_s) : V_S \to 2^{\mathbb{N}}$, as the set of colors of its incident edges. Figure 2 shows the session graph with colors for each edge and vertex.

Let $C' \subset V_S$ be the set of all overlapping role vertices in the composite session graph S with two or more colors. We call this the set of *potentially conflicting roles.* These roles may contain users that have static read access to two or more flows. To illustrate, R_7 and R_8 are potentially conflicting roles in Figure 2, and are indicated with shaded squares. After these potentially conflicting roles are identified, they are further examined for linkability conflicts.

Consider the potentially conflicting role $c' \in C'$.

Recall that all the incident edges (r, c') are augmented with the set of common users $U(r, c')$ from the AURA graph, in addition to their colors. For a given potentially conflicting role, if the intersection of the user sets for edges of *different* colors is not empty (that is if there is a user u in two edge sets of different colors) then we identify c' as a *conflicting role.* Also, if any edge has two or more colors, and at least one user in its user-set, then c' is a conflicting role. Let the set of conflicting roles be $C \subseteq C'$.

In Figure 2, R_8 is not a conflicting role since there are no users in R_8 that are in parent roles R_1 and R_3 ,that can access flows of different colors, viz., I_1 and I_2. R_7 is a conflicting role because u_2 appears on the edges (R_1, R_7) and (R_3, R_7), i.e., user u_2 with role R_7, also has roles R_1 and R_3 and can access two flows of different colors I_1 and I_2. The conflicting roles in Figure 2 are R_1, R_3 and R_7.

Complexity of detecting conflicting roles: Let E be the set of incident edges on a potentially conflicting role c'. In the worst case, each edge $e \in E$ has a different color from the other edges. For each color i (or flow), compute the union $\mathcal{U}_i$ of the edge sets $U(r, c')$ for all parent roles r of c' and all edges with color i. $\mathcal{U}_i$ is the set of users in c' that can access flow I_i. Now we must check for pairwise intersections between the $\mathcal{U}_i$'s ($O(n^2)$ intersections) to identify real conflicts. Since there are at most $|E|$ union operations bounded by the number of roles $|\Gamma|$, and each such operation is linear in $|U(e)|$ bounded by $|\mathcal{U}|$ (set union using a hash table), the worst case complexity for this step is $O(n^2|\mathcal{U}| + |\Gamma||\mathcal{U}|)$.

We now show how a user of the system can specify discretionary policies representing unlinkability requirements and present an automated technique to generate constraints on the dissemination of audit flow information. We also show how if we can enforce these constraints appropriately, we can prevent linkability.

3.4. Specifying Discretionary Policies

As described in Section 3.3, the PNS returns to Alice a set of conflicting roles C in S. Alice picks a subset of these roles C_{Alice} as her discretionary unlinkability requirements.

A linkability conflict occurs for users with role $c \in C_{Alice}$ that can access a database belonging to two or more flows. When Alice creates a new audit record that flows to a database that can be accessed by a user in a conflicting role, the underlying access control system denies the right to access these records to all users in these roles who pose a linkability threat. The PNS subsequently generates policy constraints that Alice can attach to her audit records.

3.5. Generating and Enforcing Policy constraints

We call C_{Alice} Alice's *deny-set.* The members in Alice's deny-set should be prevented from linking Alice's flows. Note that not all users in the deny-set are linkability threats, and hence we need to make sure that only the users who can link Alice's flows must be denied access. We define the Alice's policy constraints for session S, $\mathcal{P}_S$, as the tuple $\langle C_{Alice}, \mathcal{R}_1, \ldots, \mathcal{R}_n \rangle$, where $\mathcal{R}_i$ is the set of roles with static read permission to information flow I_i, and are parents of some role in C_{Alice}. This is easily obtained from the session graph S.

Audit flow records in session S are tagged with $\mathcal{P}_S$. When a user u attempts to access an audit record, the database's reference monitor first checks to see if u has static read access for that database. If so, it then checks the attached $\mathcal{P}_S$ to see if any of u's roles are in C_{Alice}. If so, the reference monitor checks to see if u's role-set $URA(u)$ has a non-empty intersection with at least two different sets in $\{\mathcal{R}_1, \ldots, \mathcal{R}_n\}$. If so, the user has static read access to two or more flows in S, and the user is denied access by the reference monitor. In the worst case, for users with static read access to the database, the reference monitor needs to compute $n + 1$ intersections, where each intersection takes $O(|URA(u)| + |\Gamma|)$ operations, which is $O(|\Gamma|)$. Hence the time complexity for evaluating $\mathcal{P}_S$ is $O(n|\Gamma|)$ if u is in Alice's deny set. If not, the time complexity is $O(|\Gamma|)$, the cost of computing the intersection $URA(u) \cap C_{Alice}$. From Figure 2, assuming that $C_{Alice} = \{R_7\}$. We have $\mathcal{P}_S = \langle \{R_7\}, \{R_1\}, \{R_3\} \rangle$.

At this point, a valid question is why not generate policy constraints with user IDs. There are three reasons for this. Firstly, if a user u was identified to be a linkability threat, then adding u to the policy constraints will prevent u from accessing two or more flows. However, if u is removed from a particular role and is no longer a linkability threat, u will *still* be denied access. Our scheme adds more precision to the system by allowing users who are no longer linkability threats to access audit records. Secondly, we would like to give administrators feedback as to why their access was denied. Our policy is able to capture the reasons *why* access control decisions are made in addition to *what* access control decisions are made.. And lastly, in large systems we expect a role based formalism to be a more compact representation of linkability conflicts.

We now present two definitions, and prove that our system is secure, sound, and precise under certain assumptions.

Definition 1. *If the access permissions for a database record associated with a flow for user* u *includes the right to read, then we say that* u *has* **static read access** *to the audit flow. These static permissions can be overridden by policy constraints.*

Definition 2. Strong Tranquility *asserts that the access permissions associated with the users of the system (i.e., the URA and the PRA) do not change by system operation.*

Policy constraints are generated based on the current protection state of the system (i.e., the URA and the PRA.) Changes to the protection state can result in policy constraints that are "out of date." We first prove that our constraints are *secure*, *sound*, and *precise* under the strong tranquility assumption. We relax this assumption in Section 4 and show how we can trade precision for security when the protection state and the session information are allowed to change. The following theorems are easy to prove because of the strong tranquility assumption, which makes the properties hold by construction of session graph S and policy constraints $\mathcal{P}_S$.

Theorem 1. *user u has static read access to two or more audit flows in a session, then all of the user's roles $URA(u)$ appear as conflicting roles in the session graph.*

(Security) *Assuming strong tranquility, if a user* u *with a role in Alice's deny-set C_{Alice}, has static read access to two or more audit flows in Alice's session $I_1, \cdots, I_n$, the policy constraints will prevent* u *from accessing these flows. Furthermore, Alice was presented with all of* u*'s roles as conflicting roles.*

Proof. Since u has static read access to two or more flows in $I_1, \cdots, I_n$ and since we assume strong tranquility, by construction all of u's roles will appear as conflicting roles in the session graph S. By construction of the constraints, u will be denied access to $I_1, \cdots, I_n$.

□

Theorem 2. (Soundness) *Assuming strong tranquility, if a user* u *is denied access to a flow I_i by the policy constraints, then the user has static read access to two or more audit flows in the session S.*

Proof. Since u was denied access by the policy constraints, u's role set includes a conflicting role role $c \in C_{Alice}$, and intersects with two or more role sets in $\mathcal{R}_1, \ldots, \mathcal{R}_n$. Since we assume strong tranquility, this implies that u has access to two or more flows in S.

□

The following theorem is simply the contrapositive of Theorem 2. In the following sections we will only refer to security and precision, since precision follows from soundness.

Theorem 3. (Precision) *Assuming strong tranquility, if a user* u *has static read access to exactly one audit flow within a session, then* u *is not denied access by the policy constraints.*

3.6. Open-ended Sessions

Our algorithm in Section 3.5 maintains security and precision for a predefined session. Consider the case when user Alice does not know all her transactions *a priori*. Alice would like to dynamically generate constraints for new audit flows, without invalidating her constraints to older audit flows. We extend our algorithm to allow users to add audit flows to existing sessions and generate new constraints appropriately.

Consider the session graph S, and the new flow I_{n+1}. Construct the session graph S' by combining the audit-flow graph for I' with S as described previously in Section 3.3, and generate the new policy constraints for audit-flow I_{n+1} as described in Section 3.5. We now show how security and precision holds for session S'. We modify the definition of security to allow access to at most one flow, since this does not violate unlinkability, and implies the security property defined in Theorem 1.

Theorem 4. (Security)
Assuming strong tranquility, if a user u *with a role in Alice's deny-set C_{Alice}, has static read access to two or more audit flows in Alice's session $I_1, \cdots, I_{n+1}$, then the policy constraints will prevent* u *from accessing* **two or more** *of these flows.*

Proof. We prove this by induction on the number of audit flows. For the base case we consider policy constraints generated for one audit flow. The set of constraints is empty. Since there is only one flow, there are no linkability conflicts. Now consider session S with audit-flows $I_1, \ldots, I_n$, and assume the security property holds for policy constraints for flows in S. If we generate new policy constraints for I' as described in Section 3.6, then any user u that has static read access to two or more flows in S' is denied access to audit-flow I'. Users with static read access to two or more flows in S are allowed access to at most one flow in S (inductive hypothesis). Consider a user u that has static read access to exactly one flow in S, and to I'. Policy constraints for S will still allow u to access a single flow in S, and the new constraints for I' will prevent u from accessing I'. Hence u can access at most one flow in S' and security holds. □

Theorem 5. (Precision)
Assuming strong tranquility, if a user u *has static read access to exactly one audit flow within a session, then* u *is not denied access by the policy constraints.*

Proof. For the base case, again consider one audit flow. Since there are no policy constraints, u will not be denied access by the policy constraints. Assume that for a session S with audit-flows $I_1, \ldots, I_n$, precision holds for the policy constraints. If we generate new policy constraints for I' as described in Section 3.6, then any user u who has static read access to exactly one audit-flow in S', will still be allowed access to I'. Consider the case when u tries to access a flow in S. If u has static read access to a flow in S, then precision holds by the inductive hypothesis. If u has static read access to I', then u does not have static read access to any flow in S and is (trivially) denied access to a flow in S.

□

3.7. Mandatory Audit Flows

The PNS may consider access by certain conflicting roles to be mandatory. In our example mentioned earlier, the PNS may mandate that student administrators cannot be denied access (in this case, *Administrator* is the parent role of the overlapping role *Student*). Specifically, the PNS can specify edges (r, o) that are mandatory, where r is a role vertex, and o is an overlapping role of r. Hence, any user with roles r and o are exempted from the policy constraints. If there are exempted users that can access two or more audit flows, the user is informed of this.

Our goal is to make the privacy implications of sensitive information explicit to the user. Users will have complete information of who can access the user's information, and will proceed only if they agree to the PNS's mandatory policy.

In the next section, we relax the strong tranquility assumption and present a discussion of what policies we can enforce when the permissions are allowed to change and investigate the trade-off between security and precision.

4. Security under Weak Tranquility

Our strong tranquility assumption in Section 3.5 is restrictive since the users, roles, and permissions, which define the protection state in any organization will change over time. Once the protection state changes, it may not be possible to enforce some of the unlinkability requirements. New conflicts may emerge that may invalidate existing guarantees.

In this section, we extend our results to model the effect of changing the protection state. Our proposed solution uses versioning to localize the impact of these updates. Since our policy enforcement mechanisms are decentralized, i.e., records belonging to a particular flow in a database are tagged with access restrictions, it is important to guarantee the security of these access restrictions under evolving protection state.

We define the notion of weak tranquility which captures the effect of changing permissions on the satisfaction of unlinkability properties.

Definition 3. Weak Tranquility *states that the access permissions (i.e., the URA and the PRA) associated with a user* u *of the system do not change in such a way that it violates the security and precision of the enforcement of discretionary unlinkability policies for that user.*

Our goal is to guarantee that changes to the protection state can preserve the weak tranquility property for as many users as possible during the lifetime of the system.

When a policy is agreed upon by the user and the PNS, the policy constraints certificate is stamped with what we call the *system version number* maintained by the PNS. When users are added to the system, they are also stamped with the current system version number. The user's version number will be updated when certain changes are made to the protection state. A user u can access an audit record belonging to flow I only if $\mathit{Version}(u) \leq \mathit{Version}(I)$. We assume that reference monitors have access to the current version number for a user (e.g., policy database or a revocation-based certificate approach). We prove Lemma 1 based on the following update rules for a user's version number.

Lemma 1. *Consider audit flows $I_1, \ldots, I_n$ in a session S. After any change to URA or PRA, if for a user u, $\mathit{Version}(u) \leq \mathit{Version}(I_i)$ for all $i = 1, \ldots, n$, then weak tranquility holds for user u with respect to audit flows $I_1, \ldots, I_n$.*

Proof. We prove this for each possible update to the protection state, and hence the lemma holds by induction on the number of updates to the protection state. For the base case, there are no updates to the protection state, and the lemma trivially holds by strong tranquility, which implies weak tranquility.

New User u Created: No change to system version number. Assign current system version number to user u. u has not been granted any new permissions and weak tranquility holds for u.

New Role r Added: No change to system version number. No permissions have changed in the system, and weak tranquility holds for all users.

User-Role (u, r) Assignment Added: When a User-Role assignment (u, r) is added, it is possible that u now has static read access to two or more flows in session S, but will not be denied access to two or more flows by the policy constraints. To maintain the security property of the policy constraints with respect to u, the system version number is incremented, and u is assigned the new version number. Since the permissions of all other users remain unchanged, security and precision of the constraints hold for all other users, whose version numbers remain unchanged.

User-Role Assignment (u, r) Deleted: No change in version number. We only need to examine the case when u had static read access to two or more flows in S before the user-role assignment was deleted. If u continues to have static read access to two or more flows in S, then u must activate roles other than r, which must appear in the original policy constraints. Hence u will be prevented access by the policy constraints if u has a role in the deny list of the constraints (security property). If u does not have any roles on the deny list (see discussion for *privilege escalation* for the case when $r \in C_{Alice}$), then u is allowed access. If it is the case that u no longer has static read access to two or more audit flows, then r was necessary for access to two or more flows. Hence $r \in URA(u)$ is a necessary condition for being denied access by the policy constraints. Since now $r \notin URA(u)$, the policy constraints will allow u to access flows in S (precision). Since the permissions of all other users remain unchanged, security and precision of the constraints hold for all other users, whose version numbers remain unchanged.

User u Deleted: Version number does not change. Equivalent to iteratively removing all User-Role assignments for u. Delete all the User-Role assignments.

Role r Deleted: Equivalent to iteratively removing all User-Role assignments for r followed by removing all $PRA(r)$. Note that after this operation, the system version number remains unchanged.

Permission-Role (d, r) Assignment Added: This means that a role r has been granted static read access to some database d. Since this role may not have been included in the session graph, it is possible that some users in r can now access two or more audit flows, and will not be denied access by the policy constraints, violating the security of the policy constraints, and weak tranquility does not hold. If there are any users assigned to role r, the system version number is incremented, and all users in r are assigned the new version number. Hence, if $Version(u) \leq Version(I_i)$ for all $i = 1, \ldots, n$, then u is not a member of r, and weak tranquility holds for u.

Permission-Role (d, r) Assignment Deleted: This means that the static read access to database d has been removed for a role r. It is possible that users in r are no longer a threat to linkability, but will still be denied access by policy constraints, violating the precision of the policy constraints. Hence weak tranquility does not hold for users in r. If there are any users assigned role r, the system version number is incremented, and all users in r are assigned the new version number. Hence, if $Version(u) \leq Version(I_i)$ for all $i = 1, \ldots, n$, then u is not a member of r, and weak tranquility holds for u. Note that the security of policy constraints is not affected by adding the assignment (d, r). However for every policy we would like to maintain the set of users for which weak tranquility holds, which is why we update the version numbers for affected users.

Privilege escalation: Consider the situation when a user has access to only one flow in a session. After accessing this information, the user is removed from a particular role, and then added to a new role, giving the user access to another flow in the session, violating the unlinkability requirement. However, the version number of the user is incremented when a new user-role assignment is added, which will prevent this kind of privilege escalation. Similarly, incrementing the version number on the addition of a new permission-role assignment prevents privilege escalation due to changing permission-role changes. More generally, privilege escalation is prevented by the fact that a user's version number is incremented whenever the user's static permission set increases. It is important to note that if a role r is removed from a user's role-set, it is possible that r is on the deny list of some policy constraint, and that the user will now be able to link flows in that session, which was disallowed before this removal. With cooperation from the security officer, a user can remove, and subsequently add, r to his/her role-set resulting in one form of privilege escalation. We assume that the security officer is trusted, and that privilege escalation from the removal of a conflicting role is semantically correct and secure. An alternative approach would be to define this type of privilege escalation as not secure, and increment the version number when a user-role assignment is removed.

$\square$

Under versioning, the following theorems follow from Lemma 1.

Theorem 6. (Secure) *If a user u with a role in Alice's deny-set C_{Alice}, has static read access to two or more audit flows in Alice's session $I_1, \cdots, I_{n+1}$, then the policy constraints will prevent* u *from accessing* **two or more** *of these flows.*

Proof. If $Version(u) \leq Version(I_i)$ for all $i =$

$1, \ldots, n$ then the weak tranquility assumption holds by Lemma 1, which implies security with respect to user u. If $Version(u) > Version(I_i)$ then the user is trivially denied access, even if their access did not cause a linkability conflict. □

Theorem 7. (Precise up to Versioning) *If a user u has static read access to exactly one audit flow within a session $S = \{I_1, \ldots, I_n\}$, then u is not denied access by the policy constraints if $Version(u) \leq Version(I_i)$ for all $i = 1, \ldots, n$.*

Proof. If $Version(u) \leq Version(I_i)$ for all $i = 1, \ldots, n$ then the weak tranquility assumption holds by Lemma 1, and hence the constraints are precise up to versioning. For users with higher version numbers, precision does not hold, since they will be denied access even if they cannot link flows within a session.

□

After the policy constraints have been generated, previously deployed policy constraints gradually lose precision by being overly restrictive to users affected by evolving system permissions. However, this is restricted only to users who gain new permissions, and users of roles for which database permissions change. We argue that the latter case is rare and can be performed at predefined system epochs. To cope with degrading precision, the PNS can choose to honor the policy constraints for a certain time-period called *unlinkability window.* This window can either be a static parameter in the system, or can be negotiated with the user. As mentioned earlier, changes in flow policies are considered to be non-trivial changes. These changes can take place in epochs that honor the unlinkability window. When this is not possible, all data along the new flow is tagged as sensitive and is only allowed access by designated administrators. Users can be informed in general that changes in flow policy are possible, and that certain designated administrators will have access to audit flows in the session.

5. Related Work

In this section, we present related research in the context of ensuring unlinkability across different access transactions within a session. Research on unlinkability in the past has mostly focused on cryptographic mechanisms for anonymous authorization. We also explore the relationship between our problem and other anonymity solutions. Finally, we examine how our problem is related to role-engineering for enforcing Separation of Duty (SoD) constraints.

We first examine different cryptographic techniques that allow a user to disclose only those attributes that are strictly necessary for a given service access transaction. One of the first proposals in this direction is the work by Brands [1], where he proposes a certificate system that gives a user control over what is known about the attributes of his or her certificate (or authorizations), and can prove their possession using zero-knowledge protocols. However, with this scheme a user who presents the same certificate twice can be linked across his or her sessions with the same server, even though the attributes are still hidden.

Other researchers have explored the construction of credential systems that satisfy the *multi-show* property whereby the owner of a certificate can construct two or more credentials with the same attributes that are unlinkable[19, 10]. The construction of *anonymous credentials* presented by Chaum in [3] relies on interaction with a trusted third party for unlinkability. Camenisch, Lysyanskaya et al. [2, 8] extend this unlinkability based on computational zero-knowledge proofs, and the credential system proposed in [10] defines what the authors call Chameleon certificates that provide a user complete control over the amount of information revealed as well as computational zero-knowledge proofs for unlinkability of credentials, provided these credentials can be encoded as linear Boolean formulas.

As discussed earlier, preserving unlinkability across access transactions with respect to the same server is not sufficient to ensure global unlinkability. One of the issues with anonymous credentials is that though the identity of a user is not revealed by engaging in multiple access transactions, the list of attributes revealed at the end of the access negotiation can be logged by the server, along with timing information. With respect to a door-lock authentication server, even if a user was able to hide all privacy sensitive attributes from the door-lock server, a successful access transaction is sufficient to expose the user's location. Furthermore, a system may not be able to support anonymous access transactions if required by law.

We also observe that privacy preserving communication techniques such as Crowds and Hordes [12, 16, 15, 11] that protect the anonymity of a sender or receiver are orthogonal to our work. These solutions are geared towards protecting the IP address of the end points of a communication session. Knowledge of endpoint sender addresses may provide enough information for a server to link two transactions even if pseudonyms were employed to obtain access. Using a solution like Crowds to hide a DHCP address from a server would not really solve our unlinkability problem as we are concerned with users being able to link DHCP log information with server log information. In fact, if an administrator is allowed access to all Crowds router logs, timing analysis can expose the endpoint identity of the sender.

As explained in Section 1, our unlinkability problem differs from the SoD problem in two important ways. The SoD problem is defined as preventing a single user from performing different actions on the same object in the course of a workflow to protect the transactional integrity [17]. In our problem, we want to prevent an unauthorized user from accessing different audit records associated with different information flows initiated by a single user.

In their discussion on different types of SoD constraints for RBAC, Simon and Zurko [17] distinguish between three types of SoD constraints : static, dynamic, and operational. Given a set of static SoD constraints, policy conformance reduces to checking if the roles involved have disjoint memberships so that no single person has access to all operations in a workflow.

With respect to enforcing dynamic SoD constraints Sandhu's work on Transaction Control Expressions (TCE [13]) shows how dynamic SoD constraints can be enforced adequately using history if the information about each transaction is annotated with the object itself. Simon and Zurko argue that such history is essential to enforce general SoD constraints. Gligor et al. [5] formalize the relationship between SoD and RBAC and show how RBAC is not sufficient to enforce all types of SoD properties, especially dynamic SoD constraints. More recently, Li et al. [7] show how directly enforcing static SoD policies is intractable, let alone dynamic SoD policies, and show how statically mutually exclusive roles can be engineered to enforce these constraints on a best-effort basis. In the context of our unlinkability problem, annotating audit records in different databases with history information does not provide us a mechanism to enforce unlinkability as these data objects are independent and local history cannot be used to enforce global constraints. Instead, our proposed solution annotates different audit records with different authorizations to enforce unlinkability.

In terms of detecting semantic conflicts that can be exploited by a user to correlate different types of audit records and expose the privacy of a user, a number of data mining techniques that explicitly represent knowledge can prove to be useful. Researchers have examined how to use data mining techniques to correlate logs in the context of intrusion detection, to detect attacks [6, 18]. We believe that some of these techniques can be extended to look for unlinkability conflicts at the semantic level. As mentioned in Section 2, our framework examines the unlinkability problem at the level of authorizations to access audit flows. As we mention earlier, analysis of the semantics of whether two flows that can be accessed by the same user can be leveraged to improve the precision of enforcement of unlinkability policies.

6. Conclusions

We explore the problem of user unlinkability in the context of correlating audit data. Our work examines how administrative users with authorizations to view audit records across different servers in an organization can link different access transactions to sensitive attributes of other users such as identity and location. We show how this problem persists even if users employ anonymous credentials to gain access to a service. To the best of our knowledge, our work in this paper is the first to discuss a policy-based approach for enforcing unlinkability.

We formalize the unlinkability problem by defining the the notion of an audit flow associated with a user's access transaction. Audit flows for different access transactions can be composed to generate a session graph that encodes the linkability conflicts compactly and captures the scope of the problem adequately. Using this session graph, we show we can transform the unlinkability problem into a policy engineering problem, and present an algorithm to generate authorization constraints that can enforce unlinkability by restricting access in the context of RBAC.

With appropriate tranquility assumptions on the underlying authorizations, we prove that our constraints can guarantee unlinkability. We formalize the notion of security and precision with respect to enforcing unlinkability constraints. To maintain the security of deployed policy constraints under evolving protection state, we propose a solution based on versioning that maintains security by trading precision for evolving protection state. Using our approach, the set of users for which policy constraints are secure and precise can always be identified.

7 Acknowledgments

We thank Sariel Har-Peled, Mike Corn, and Shea Nangle for their helpful comments.

References

[1] S. Brands. *Rethinking Public Key Infrastructures and Digital Certificates; Building in Privacy.* MIT Press, 2000.

[2] J. Camenisch and A. Lysyanskaya. An efficient non-transferable anonymous multishow credential system with optional anonymity revocation. In *EUROCRYPT*, 2001.

[3] D. Chaum and J. Evertse. A secure privacy preserving protocol for transmitting personal information between organizations. In *CRYPTO*, 1986.

[4] D. F. Ferraiolo and D. R. Kuhn. Role-based access controls. In *In Proceedings of the 15th NIST-NSA National Computer Security Conference, Baltimore, MD, Oct*, 1992.

[5] V. D. Gligor, S. I. Gavrila, and D. Ferraiolo. On the formal definition of seperation-of-duty policies and their composition. In *In Proceedings of the IEEE Symposium on Research in Security and Privacy. (Oakland, CA.), 172–183*, 1998.

[6] W. Lee and S. Stolfo. Data mining approaches for intrusion detection. In *Proceedings of the 7th USENIX Security Symposium*, San Antonio, TX, 1998.

[7] N. Li, Z. Bizri, and M. V. Tripunitara. On mutually-exclusive roles and separation of duty. In *Proceedings of the ACM Conference on Computer and Communications Security (CCS), October*, 2004.

[8] A. Lysyanskaya, R. Rivest, A. Sahai, and S. Wolf. Pseudonym systems. In *Selected Areas of Cryptography, Volume 1758 LNCS*, 1999.

[9] A. Pashalidis and C. J. Mitchell. Limits to anonymity when using credentials. In *Proceedings of the 12th International Workshop on Security Protocols, Springer-Verlag LNCS, Berlin*, Cambridge, UK, April 2004.

[10] P. Persiano and I. Visconti. An anonymous credential system and a privacy-aware pki. In *R. Safavi-Naini and J. Seberry, editors, Information Security and Privacy, 8th Australasian Conference, ACISP 2003, volume 2727 of Lecture Notes in Computer Science. Springer Verlag*, 2003.

[11] M. G. Reed, P. F. Syverson, and D. M. Goldschlag. Anonymous connections and onion routing. *IEEE Journal on Selected Areas in Communication (JSAC), Special Issue on Copyright and Privacy Protection*, 1998.

[12] M. K. Reiter and A. D. Rubin. Anonymous web transactions with crowds. *Communications of the ACM*, 42(2):32–48, 1999.

[13] R. Sandhu. Transaction control expressions for separation of duties. In *Proceedings of the 4th Aerospace Computer Security Applications Conference*, 1998.

[14] R. S. Sandhu, E. J. Coyne, H. L. Feinstein, and C. E. Youman. Role-based access control models. *IEEE Computer*, 29(2):38–47, 1996.

[15] R. Sherwood, B. Bhattacharjee, and A. Srinivasan. P5: A protocol for scalable anonymous communication. In *Proceedings of the 2002 IEEE*

Symposium on Security and Privacy, page 58. IEEE Computer Society, 2002.

[16] C. Shields and B. N. Levine. A protocol for anonymous communication over the internet. In *Proceedings of the 7th ACM conference on Computer and communications security*, pages 33–42. ACM Press, 2000.

[17] R. T. Simon and M. E. Zurko. Separation of duty in role-based environments. In *IEEE Computer Security Foundations Workshop*, pages 183–194, 1997.

[18] J. L. Undercoffer and A. Joshi. *Data Mining, Semantics and Intrusion Detection: What to dig for and Where to find it.* MIT Press, December 2003.

[19] E. R. Verheul. Self-blindable credential certificates from the weil pairing. In *Proceedings of the 7th International Conference on the Theory and Application of Cryptology and Information Security*, pages 533–551. Springer-Verlag, 2001.