Distributed Enforcement of Unlinkability Policies: Looking Beyond the Chinese Wall

Apu Kapadia*
Institute for Security Technology Studies (ISTS)
Dartmouth College

Prasad Naldurg
Microsoft Research

Roy H. Campbell
Department of Computer Science
University of Illinois at Urbana-Champaign

Abstract

This paper presents an access control model that preserves the unlinkability of audit-logs in a distributed environment. The model restricts entities from accessing and correlating two or more audit-records belonging to different service invocations created by the same user. While the traditional Chinese Wall (CW) model is sufficient to enforce this type of unlinkability, in distributed environments CW is inefficient because the simple security condition semantics requires knowledge of a user's access history. Our model allows specifications that are simple and efficient to enforce in a decentralized manner without the need for an access history. The proposed enforcement architecture allows users to negotiate unlinkability policies with the system. The system attaches automatically generated policy constraints to the audit-records. When these constraints are enforced appropriately, they implement unlinkability policies that are provably secure and precise for a fixed protection state. The model extends to a versioning scheme that adapts to evolving protection state, trading off precision to maintain the security of deployed policies.

1 Introduction

Our problem is motivated by privacy concerns within a distributed environment like an organization or corporate network. Local and external administrators from across various departments may monitor a user's accesses to various resources through audit-logs. In the case where "linking" these accesses is not explicitly sanctioned by a mandatory policy, such linkages can compromise the privacy of users. We present a model that allows users to express and refine discretionary unlinkability concerns and have them enforced by the system.

A variety of databases with independent access control mechanisms often store the audit-records of users' actions and service invocation requests across departmental boundaries. This makes enforcement of unlinkability a difficult task. In theory, centralized mechanisms based on the Chinese Wall (CW) model can solve this problem. For example, a CW policy could ensure that administrators can access at most one dataset within a "conflict of interest" class. However, the implementation must maintain a history of administrators' accesses. In a distributed setting, maintaining this access history across departments may render this approach infeasible or impractical. Centralized solutions present a bottleneck for distributed access to resources and act as a single point of failure for access control. Distributed enforcement of CW policies requires the propagation of history information that must be kept consistent across different databases, incurring high communication and computational overheads.

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. To address this problem, researchers have proposed a number of cryptographic mechanisms to construct anonymous credentials [3, 1, 2, 7] that make it computationally infeasible for a server to link the use of these credentials. However, even if a user presents

*Apu Kapadia was funded in part 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 Laboratories. This work was conducted primarily at the University of Illinois at Urbana-Champaign as part of Apu Kapadia's dissertation research.

an anonymous credential to access a service, the set of users allowed to possess those credentials in the first place may be small enough to limit or compromise anonymity. Furthermore, while many of these schemes rely on providing user anonymity, there are systems in which users simply cannot be anonymous. For example, an organization may be required to keep detailed audit-records about who accessed payroll information by law. In such systems, it becomes important to provide unlinkability through access control, allowing for linkability in only certain cases, e.g., legal subpoenas, or only by certain individuals in the organization. We note that cryptographic mechanisms are also vulnerable to timing attacks [9], and adequate access control mechanisms can prevent such attacks. In this paper we focus on providing unlinkability through confidentiality policies, with the observation that denying access to related audit-records prevents the possibility of linking the contents of the audit-records. We also assume that feedback to administrators does not leak information about the existence of an audit-record. For example, feedback of the form "record unavailable" can imply that either the record was not found, or access was denied. This prevents administrators from inferring that two audit-records are related from the feedback.

In this context, we introduce an access control model based on RBAC (role based access control) [1] for **policy-based unlinkability** that addresses the problem of restricting accesses by a single administrator to multiple audit-records belonging to the same user as defined by a "session." This model does not require an implementation to maintain the access history of users in the system. We provide an efficient enforcement framework based on this model that can analyze the system protection state for unlinkability threats, and change the authorizations based on the user's requirements (except when they are explicitly required by system policy) to counter these threats. For example, the system may inform Alice that network administrators can access information of her (possibly anonymized) access to an online medical store. Furthermore, there may be some network administrators who can also access Alice's local system log and read her cookies, which contain information about what prescription was filled. Using our framework, Alice can request that network administrators who also have access to her local system be prevented from correlating the information and compromising her privacy with regards to the ailment she is seeking treatment for. In effect, the system allows Alice to negotiate a set of constraints to prevent certain administrative users from linking her transactions, while allowing certain trusted administrators to do so. These constraints are attached to individual audit-records and local access control decisions can be made based on these constraints allowing for the distributed enforcement of users' unlinkability policies.

We prove that our architecture is both secure and precise with respect to enforcing unlinkability properties. We first prove these results under the strong tranquility assumption where the protection state of the system does not change over a session. That is, the administrators' access rights remain fixed. Subsequently, we show how we can relax these assumptions and present an approach 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 existing sessions, refining their unlinkability requirements iteratively.

We briefly summarize our contributions:

1. We introduce a new access control model for scalable and decentralized enforcement of unlinkability policies without the need for maintaining the access history of users (Section 2).
2. We provide an efficient framework based on our access control model and prove that it is secure and precise for fixed protection state (Section 3).
3. To cope with evolving protection state, we present an approach based on versioning. We prove that our approach maintains the security of deployed unlinkability policies by trading off precision for evolving protection state (Section 4).

We discuss our approach in Section 5 and end with related work and conclusions in Sections 6 and 7.

2 Access control model

In this section, we provide some background on the Chinese Wall (CW) model and present our model in relation to its simple security condition. We show how our specification of unlinkability is more restrictive in terms of the set of allowed behaviors, but nevertheless captures our intent adequately.

2.1 Chinese Wall Model

Policies in the Chinese Wall (CW) Model group objects into *Conflict of Interest* (COI) classes and individual users are not allowed to access information from

[1] We use RBAC as a means to group access control permissions for administrative convenience and do not use extended models of RBAC such as role hierarchies.

two or more objects in a COI class. In particular, the semantics of CW policies allow an individual to access any one object in a COI class, and prevents further accesses to other objects in that class. As a result, Chinese Wall policies are enforced using centralized history-based approaches [11] or require explicit coordination [8], which is expensive in practice.

To elaborate further, objects belonging to a company are grouped into a Company Dataset (CD). CDs can be grouped into COI classes. The overall goal of this model is that no subject can read objects from two or more CDs within the same COI class. Once a subject reads from a particular CD, future accesses to other CDs within the COI class are denied. Let $CD(O)$ be the CD of object O (similarly $COI(O)$) and $PR(S)$ be the set of objects that S has read. Each object belongs to exactly one COI class.

The CW access control model specifies the following conditions:

Definition 1. CW-Simple Security Condition: *A subject S can read an object O if and only if any of the following holds:*

1. *There is an object O' such that S has accessed O' and $CD(O') = CD(O)$.*
2. *For all objects O', $O' \in PR(S) \Rightarrow COI(O') \neq COI(O)$.*

This means that once a subject has accessed an object within a COI class, a "Chinese Wall" is created around $CD(O)$. Access to any object outside this wall is denied unless the object is in a COI class not accessed earlier. Once the user accesses an object in another COI class, the "wall" is extended around the CD of that object.

Definition 2. CW-*-Property: *A subject S may write to an object O if and only if both of the following conditions hold:*

1. *The CW-simple security condition permits S to read O.*
2. *For all objects O', S can read $O' \Rightarrow CD(O') = CD(O)$.*

This property prevents the leaking of information through writes. A subject can write to a CD only if all the objects he/she can read belong to the same CD. This means that a CD's data stays within the CD.

2.2 Our model

In this subsection, we show how we can modify the simple security condition in the CW model to capture the notion of unlinkability in our environmental context. 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*. Users negotiate unlinkability policies with a policy negotiation server (PNS) that generates policy constraints to enforce unlinkability using access control mechanisms.

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 dissemination of audit information for that transaction to other users in the organization. A collection of audit-flows (corresponding to access transactions) 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.

It is possible to enforce unlinkability policies using the Chinese Wall model. The user's session can be specified as a COI class, where each audit-flow is equivalent to a CD. The CW model would prevent administrators from accessing audit-records of two or more audit-flows within the specified session. However, the distributed enforcement of CW policies requires a mechanism to record and disseminate access history across the system, which directly impacts its scalability. We present an alternative access control model that *does not maintain the access history of users*, making distributed enforcement of the CW-simple security condition efficient.

We assume that administrators will not explicitly change audit-records (e.g., by writing over) and exercise only *read* accesses to audit-log databases. The problem we are addressing is end-user privacy, and it is safe to assume that administrators are not intentionally malicious. An under-specification of privacy concerns may make it possible for some administrators to violate user privacy, and our goal is to refine this with user input. We do not account for the CW-*-property any further for this reason.

As mentioned earlier, we assume that users and access permissions are organized into roles using RBAC. Our underlying RBAC system governs read accesses to audit-logs in the absence of unlinkability policies. We refer to these permissions as "static read access":

Definition 3. *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 ac-**

cess *to the audit-flow. These static permissions can be overridden by policy constraints.*

In our proposed model, users identify transactions and group them together to define their "session." Our unlinkability specification will guarantee that subjects with static read access to two or more audit-records of different flows are denied access to *any* object within the session. Note that the set of behaviors allowed in this model is more restrictive that CW (that would allow access to one object), we show that this semantics allows us to enforce policies in a decentralized setting, where access decisions can be made local to the object being accessed.

Let $Session(O) = I_1, \ldots, I_n$ be the session (set of audit-flows) that object O belongs to. Let $I(O)$ be the audit-flow for object O. Users also supply a set of roles called the "deny-set" that the unlinkability policies should apply to. Subjects not in the deny-set are allowed to link audit-flows within the session. We explain this in more detail in Section 3. Let $DenySet(O)$ be the deny-set associated with $Session(O)$.

We define the simple security condition for Unlinkability as follows:

Definition 4. Simple Security Condition (SSC1) for Unlinkability: *A subject S is granted read access to an object O if and only if the following hold:*

1. *S has static read access to O.*
2. *if $S \in DenySet(O)$ and there is no object O' such that $Session(O') = Session(O)$ and $I(O) \neq I(O')$, and S has static read access to O'.*

This means that subjects in the deny-set of O with static read access to two or more audit-flows within a session are denied access to any objects within the session. This semantics does not give subjects the choice of accessing exactly one flow within a session and is applied only to subjects identified as unlinkability threats by the user (as defined by the user's deny-set for that session). SSC1 can be enforced easily if all the audit-record types (audit-flows) in a session are specified in advance. However, for "open-ended" sessions where all the audit-flows are not known in advance (a more realistic assumption), we modify SSC1 to allow read access to at most one audit-flow within a session, but access to the flow of the user's choosing is not guaranteed. We call this SSC2:

Definition 5. Simple Security Condition 2 for Unlinkability (SSC2): *A subject S is granted read access to an object O if the following hold:*

1. *S has static read access to O.*
2. *if $S \in DenySet(O)$ and there is no object O' such that $Session(O') = Session(O)$ and $I(O) \neq I(O')$, and S has static read access to O'.*

Furthermore, if a subject S is granted read access to an object O then the following hold

1. *S has static read access to O*
2. *S will not be granted read access to any object O' such that $I(O) \neq I(O')$ and $Session(O') = Session(O)$*

In short, a user with static read access to two or more audit-flows in a session may be able to access zero or one audit-flow in the session, but access to a particular audit-flow cannot be guaranteed.

In the remainder of this paper we show that our access control model can be enforced efficiently in a distributed environment by attaching policies to data. Since evolving protection state can result in a violation of the simple security condition, we present a system that uses versioning to maintain the security of deployed policies.

3 Enforcement architecture

In this section, we present a high level overview of our system and describe the various architectural components needed to support our framework using an end-user example scenario as shown in Figure 1.

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 propagate 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.

(1) In the first step, a concerned user Alice sends her session information to the policy negotiation server (PNS). This is a set of identifiers (or unique types) corresponding to access transactions to unique servers. 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

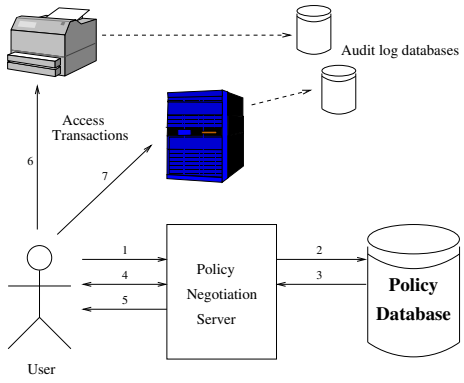


Figure 1. System Architecture

roles whose users can access her audit-log information from two or more audit-flows, e.g., *Security Officer* and *Student Administrator*

In Step (4) Alice identifies her discretionary unlinkability requirements in terms of roles (we call this Alice's "deny-set") whose users she wants to prevent from linking her audit information, e.g., *Student Administrator*. The PNS may not be able to enforce some of Alice's choices if there are mandatory access requirements, e.g., the request to add *Security Officer* to Alice's deny-set may be disallowed by mandatory system policy. After Alice and the PNS agree on Alice's deny-set, (5) the PNS sends Alice a certificate with policy constraints for her audit-records. This certificate is 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. Access to an audit-flow is allowed only if the user role is not precluded by the policy constraints. We assume that all interactions are cryptographically secured for authenticity, confidentiality, and integrity.

3.1 Construction

We now propose an approach to enforce unlinkability for Alice's session by analyzing the roles that are explicitly granted read access to each audit-flow. In our construction, a PNS examines 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. The main idea here is that if two audit-flows have common overlapping roles, then the flows are potentially linkable since a common overlapping role between audit-flows indicates that there may be users with that role who can access both audit-flows. If this is the case, then we call the common overlapping role a *conflicting role* for that session. For example, if I_1 is accessible by *Network Admin* and I_2 is accessible by *Local Admin* there may be users in the *Student* role that also belong to both *Local Admin* and *Network Admin*. In this case *Student* is a conflicting for I_1 and I_2. The PNS identifies the set of conflicting roles and presents this to the user, who picks a subset of these conflicting roles as the "deny-set." Alice may decide that *Student* administrators are potential threats to her privacy and pick *Student* as her deny set. Policy constraints are generated that will ensure that all read accesses to audit-flows satisfy SSC1. We also assume that for the purposes of accessing audit-logs, the system has access to all the roles that a user *can* activate, not only those that the user has activated currently. 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 the policy constraints and deny access appropriately. 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.

3.2 Audit-Flow Graph

Let the set of roles [2] 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. $URA(u)$ is the set of all 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 $V \subseteq \Delta \cup \Gamma \cup \Gamma'$, representing databases, roles, and overlapping roles (where Γ' is a copy of Γ). 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.

We now describe how to create an audit-flow graph, given a root vertex that represents Alice's transaction,

[2] Our system includes roles, databases, and users. 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.

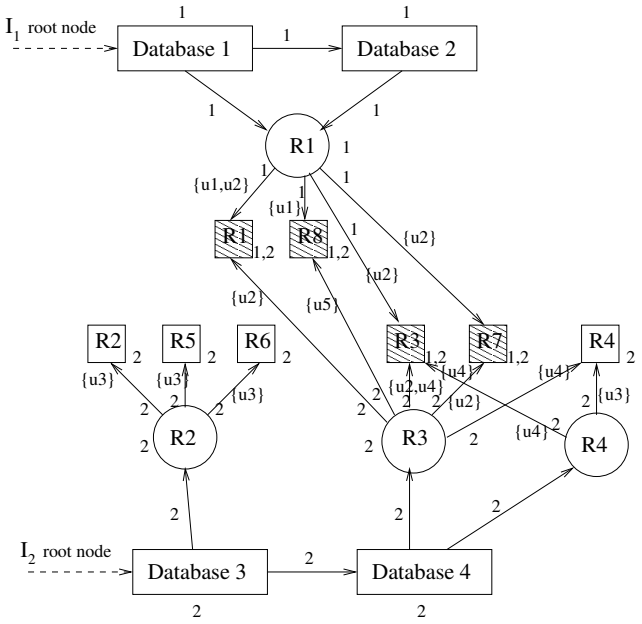


Figure 2. Session Graph

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, 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. These edges also contain information about the set of users common to both roles. 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) and describe its construction in Appendix A. The cost of updating the AURA graph is incurred when there is a change in protection state (user-role assignments and deletions) and otherwise lookups incur no additional cost during session graph creation.

3.3 Session Graph

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 $Color : I_i \rightarrow \mathbb{N}$, 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 $Colors(v_s) : V_S \rightarrow 2^{\mathbb{N}}$, 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 *common overlapping roles* or *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 since these users can access two or more audit-flows within the session. 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 that if these constraints are enforced appropriately the simple security condition for unlinkability is satisfied.

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. We call C_{Alice} Alice's *deny-set*.

A linkability conflict occurs for users with role $c \in C_{Alice}$ that can access databases 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

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 $\mathcal{P}_S$ for session 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 $O(|URA(u)| + |\Gamma|)$ oper-

ations (using hash-tables), 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 two 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. And secondly, in large systems we expect a role based formalism to be a more compact representation of linkability conflicts.

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

Definition 6. 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* with respect to SSC1 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 trivially hold 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. (SSC1 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 $I_1, \cdots, I_n$ since one of u's roles appears in C_{Alice}. □

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' with respect to SSC2. In effect, we show that SSC2 holds for open-ended sessions.

Theorem 4. (SSC2 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, static read access to exactly one audit-flow in S', will still 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 (by assumption)and is allowed access to I' by the new constraints.

□

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 role o is 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 policy constraints.

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 without requiring updates to deployed policies. Similar to maintaining consistent access histories for CW policies, updating policies throughout the system is considered to be infeasible.

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

Definition 7. Weak Tranquility for user u **with respect to policy constraint** $\mathcal{P}_S$ *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* $\mathcal{P}_S$ *for user* u.

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 and identify such users for each policy constraint in 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 current *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 $Version(u) \leq Version(I)$, which implies that weak tranquility holds for u with respect to the policy constraint for 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*,*

Version$(u) \leq$ *Version*(I_i) *for all* $i = 1, \ldots, n$, *then weak tranquility holds for user* u *with respect to the policy constraints* $\mathcal{P}_S$.

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 static read access and weak tranquility trivially holds for u with respect to $\mathcal{P}_S$. Weak tranquility for other users with respect to $\mathcal{P}_S$ is not affected by this change.

New Role r **Added:** No change to system version number. No permissions have changed in the system, and weak tranquility holds for all users with respect to $\mathcal{P}_S$.

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 $\mathcal{P}_S$ with respect to u, the system version number is incremented, and u is assigned the new version number. Hence existing policies $\mathcal{P}_S$ will deny access to u based on u's version number. Weak tranquility for other users with respect to $\mathcal{P}_S$ is not affected by this change.

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). Weak tranquility for other users with respect to $\mathcal{P}_S$ is not affected by this change.

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 for users in r with respect to existing policy constraints $\mathcal{P}_S$. 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. Weak tranquility for other users with respect to $\mathcal{P}_S$ is not affected by this change.

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. Weak tranquility for other users with respect to $\mathcal{P}_S$ is not affected by this change. 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.

□

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 for the policy constraints in I_1, which implies security with respect to user u. If $Version(u) > Version(I_i)$ for some i then the user is trivially denied access to I_i even if their access did not cause a linkability conflict, implying the security condition. □

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. Note that if there is an open-ended session for which some audit-flows have lower version numbers (and some higher) than u's version number, the constraints guarantee precision for the subset of audit-flows with higher version numbers. □

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 Discussion

Guaranteeing the unlinkability of a user's accesses is a hard problem in general because of other channels of observation outside the scope of anonymizing protocols or an access control system. Motivated adversaries can physically observe a user accessing a printer, room, etc. We have made a first attempt at characterizing the semantics of unlinkability in a distributed setting as a confidentiality property. We assume that users will take the necessary physical safeguards for their privacy and our model provides the user with the specific guarantee that two or more records within the user's session will not be accessible by certain individuals, preventing the unlinkability of the contents within the records.

Our enforcement architecture uses versioning to maintain the security of policies, but these policies lose their precision as protection state evolves. Our solution does not claim to achieve optimality. Better enforcement mechanisms (e.g., a more sophisticated versioning scheme) may yield better precision under evolving protection state. Ideally distributed reference monitors would identify exactly whether a user violates the security and precision of a policy and deny access accordingly. Our solution takes the conservative approach of identifying users for which the system cannot guarantee security and precision. Further study is required to measure the rate of degradation of precision in realistic systems.

6 Related Work

Research on the privacy of a user's accesses has focused on cryptographic mechanisms for anonymous authorization. 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. To provide unlinkability, 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[14, 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, 7] 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.

With respect to access control mechanisms other than CW, the Separation of Duty (SoD) problem is traditionally viewed as preventing a single user from performing different actions on the *same* object in the course of a workflow to protect the transactional integrity [12]. 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 Separation of Duty (SoD) constraints for RBAC, Simon and Zurko [12] 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 [11]) 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, violating our requirement of not maintaining access history. Gligor et al. [4] 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. [6] 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 with 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 policy constraints 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 [5, 13]. We believe that some of these techniques can be extended to look for unlinkability conflicts at the semantic level. As mentioned in Section 1, our framework examines the unlinkability problem at the level of authorizations to access audit-flows. Analysis of the semantics of whether two audit-flows that can be linked by a user can be leveraged to improve the precision of enforcement of unlinkability policies.

7 Conclusions

We explored the problem of unlinkability in the context of administrators accessing a user's audit-logs in a distributed environment. We argued that Chinese Wall policies are difficult to enforce in a distributed environment and proposed new semantics that allow for the efficient enforcement of unlinkability policies without having to maintain any access-history. We presented an efficient enforcement architecture for our access control model. We showed how audit-flows for different access transactions can be composed to generate a session graph that encodes the linkability conflicts compactly. Using this session graph, we showed how we can transform the unlinkability problem into a policy engineering problem, and presented an algorithm to generate authorization constraints for unlinkability. With appropriate tranquility assumptions on the underlying authorizations, we proved that our constraints can guarantee unlinkability. We formalized 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 proposed a solution based on versioning that maintains security by trading precision for evolving protection state.

Acknowledgment

We thank Marianne Winslett for her 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.-H. Evertse. A secure privacy preserving protocol for transmitting personal information between organizations. In *CRYPTO*, 1986.

[4] V. D. Gligor, S. I. Gavrila, and D. F. 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.

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

[6] 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.

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

[8] N. H. Minsky. A Decentralized Treatment of a Highly Distributed Chinese-Wall Policy. In *Proceedings IEEE 5th International Workshop on Policies for Distributed Systems and Networks (POLICY 2004)*, pages 181–184, June 2004.

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

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

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

[14] 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.

A Constructing the AURA Graph

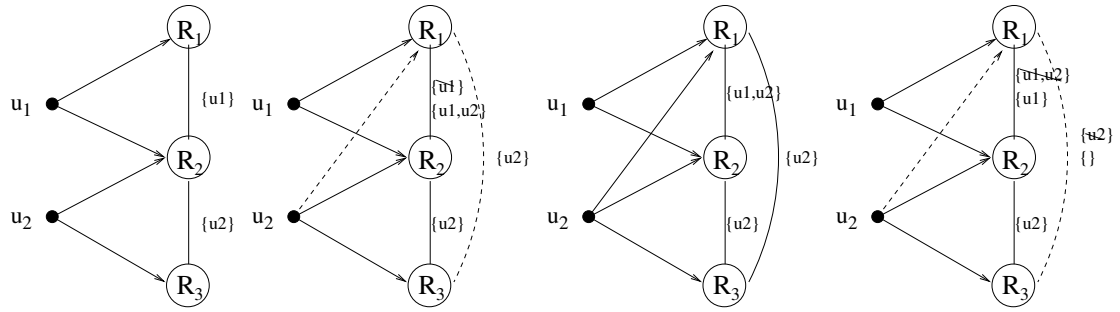
A directed edge (u, r) mean that user u is assigned to role r. An undirected edge (r_1, r_2) means that r_1 and r_2 are overlapping roles. Each undirected edge is associated with the set of overlapping users for roles r_1 and r_2, $U(r_1, r_2)$. In Section 3.3 we will use these user-sets to identify 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 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).



(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

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.