The Design and Demonstration of an Actor-Based, Application-Aware Access Control Evaluation Framework

William C. Garrison III
Department of Computer Science
University of Pittsburgh
bill@cs.pitt.edu

Adam J. Lee
Department of Computer Science
University of Pittsburgh
adamlee@cs.pitt.edu

Timothy L. Hinrichs
Department of Computer Science
University of Illinois at Chicago
hinrichs@uic.edu

***Abstract*—To date, most work regarding the formal analysis of access control schemes has focused on quantifying and comparing the expressive power of a set of schemes. Although expressive power is important, it is a property that exists in an *absolute* sense, detached from the application-specific context within which an access control scheme will ultimately be deployed. In this paper, by contrast, we formalize the access control *suitability analysis problem*, which seeks to evaluate the degree to which a set of candidate access control schemes can meet the needs of an application-specific workload. This process involves both reductions to assess whether a scheme is *capable* of implementing a workload, as well as cost analysis using ordered measures to quantify the *overheads* of using each candidate scheme to service the workload. We develop a mathematical framework for analyzing instances of the suitability analysis problem, and evaluate this framework both formally (by quantifying its efficiency and accuracy properties) and practically (by exploring a group-based messaging workload from the literature). An ancillary contribution of our work is the identification of *auxiliary machines*, which are a useful class of modifications that can be made to enhance the expressive power of an access control scheme without negatively impacting the safety properties of the scheme.**

I. Introduction

Access control is one of the most fundamental aspects of computer security, and has been the subject of much formal study. However, existing work on the formal analysis of access control schemes has focused largely on comparing the *relative expressive power* of two or more access control schemes (e.g., [1]–[8]). Although expressive power is an interesting and meaningful basis for comparing access control schemes, it exists only as a comparison made in absolute terms. That is, the knowledge that a scheme $\mathcal{S}$ is more expressive than another scheme $\mathcal{S}'$ provides no assurance that $\mathcal{S}$ is the best access control scheme for use within a particular real-world application context. It could be the case, for instance, that $\mathcal{S}'$ is *expressive enough* for a particular application and also has lower administrative overheads than $\mathcal{S}$ would in the same situation. As was noted in a recent NIST report, access control is not an area with "one size fits all" solutions and, as such, systems should be evaluated and compared relative to application-specific metrics [9]. This report notes a variety of possible access control quality metrics, but provides little guidance for actually applying these metrics and carrying out *practical* analyses of access control schemes.

Considering the wide availability of many diverse access control schemes and the relative difficulty of designing and building new secure systems from the ground up, an interesting topic for exploration is that of *suitability analysis*. Informally, this problem can be stated as follows: *Given a description of a system's access control needs and a collection of access control schemes, which scheme best meets the needs of the system?* Instances of this question can arise in many different scenarios, encompassing both the deployment of new applications and the reexamination of existing applications as assumptions and requirements evolve. Modern software applications are complex entities that may control access to both digital (e.g., files) and physical (e.g., doors) resources. Given that organizations are typically afforded little guidance in choosing appropriate security solutions, suitability analysis could help software developers sort through the myriad available security frameworks and the multiple access control schemes embedded in each.

In this paper, we identify and formalize the access control suitability problem, and develop a mathematical framework and techniques to facilitate suitability analysis. We first formalize the notion of an access control workload to abstract the application's access control needs and the expected uses of these functionalities. Analysis then consists of two orthogonal tasks: (i) demonstrating that each candidate access control scheme is capable of *safely* implementing the workload, and (ii) quantifying the costs associated with the use of each candidate scheme. Within this context, we develop techniques for safely extending the functionality of candidate schemes that require additional expressive power, develop guidelines for formally specifying a wide range of access control cost metrics, and present a simulation framework for carrying out Monte Carlo-based cost analysis within our mathematical model. In doing so, we make the following contributions:

- We formalize the *access control suitability analysis problem*, and articulate a set of requirements that should be satisfied by suitability analysis frameworks.
- We present the first formal definition of an *access control workload*. This enables system administrators to clearly and concisely specify the functionalities that must be provided by access control schemes that are to be used within a given context, as well as identify the ways in

which these schemes are envisioned to be exercised.

- We develop a *two-phase analysis framework* for assessing the suitability of an access control scheme with respect to a particular workload. We first establish whether the candidate system is expressive enough to safely implement the functionality of the workload via reduction. We then utilize a constrained, actor-based workload invocation structure to drive a cost analysis simulation that explores the expected costs of deployment.
- To address issues of fragility that arise when constructing reductions between a workload and a candidate scheme, we introduce the notion of *access control auxiliary machines* (AMs). From a practical perspective, auxiliary machines represent "tweaks" that can be made to an existing scheme to increase the range of questions that it can answer. From a theoretical perspective, AMs describe a class of enhancements to a scheme's expressive power that *do not* alter its safety properties (i.e., those that *strictly* expand the set of policies that can be represented). We prove the safety guarantees of AMs, and demonstrate their use during suitability analysis.
- We present a detailed case study demonstrating how our framework can be used to gain insight into a realistic scenario. Namely, we investigate a workload derived from a group messaging scenario [10]–[12]. We confirm the intuition that such a scenario can be implemented in commonly-used general-purpose access control schemes (though extensions are required to do so safely). We also found that such implementations differ widely in their costs, confirming the belief that addressing group-based sharing using general-purpose access control (even with a scheme that is expressive enough) can lead to inferior results. This emphasizes the importance of suitability analysis when making access control decisions.

The remainder of this paper will be structured as follows. In Section II, we describe related work. In Section III, we present a formal problem statement, solution requirements, and an overview of our proposed analysis framework. In Sections IV and V, we describe the two phases of our framework in detail (expressiveness evaluation and cost analysis, respectively). In Section VI, we discuss techniques for extending an under-expressive scheme so that it may implement a workload that it otherwise could not. We describe our case study and present its results in Section VII. In Section VIII, we evaluate the degree to which our analysis framework meets the requirements in articulated in Section III, and discuss a number of interesting open problems related to the suitability analysis problem. We conclude in Section IX.

II. Related Work

The formal study of access control schemes began with the seminal paper by Harrison, Ruzzo, and Ullman that investigated the rights leakage problem [1]. This paper formalized a general access control model and proved that determining whether a particular access right could ever be granted to a specific individual—the so-called "safety problem"—was undecidable. Shortly thereafter, Lipton and Snyder showed that in a more restricted access control system, safety was not only decidable, but decidable in linear time [2]. These two results introduced the notion that the most capable system is not always the right choice—that restricting our system can yield higher efficiency and greater ease in solving relevant security problems. This led to many results investigating the relative expressive power of various access control schemes, often leveraging some notion of (bi)simulation (e.g., [3], [4], [6]–[8]).

Further work by Ammann et al. [3], Chander et al. [4], and Li et al. [13] developed simulation-based frameworks for comparing the expressive power of various access control schemes. These simulation frameworks proved to be too relaxed, allowing almost any reasonable scheme to be shown equivalent to all others. To address this, Tripunitara and Li [5] developed a more restrictive notion of expressive power. Their framework supersedes the more informal notions of simulation developed in prior works by requiring the use of specific types of mappings between systems that guarantee relevant security properties are preserved under simulation; this provides a greater level of precision when ranking access control schemes in terms of their expressiveness. Unfortunately, none of these frameworks supports the comparison of access control schemes with regards to their ability to perform *well* within a particular environment.

The need for application-aware evaluation of access control systems was reinforced by a recent NIST report, which states that "when it comes to access control mechanisms, one size does not fit all" [9]. The report bemoans the lack of established quality metrics for access control systems, going so far as to list numerous possibilities, but stopping short of explaining how one might choose between them or evaluate systems with respect to one's specific requirements. In this paper, we develop a formal framework for exploring exactly this problem.

Wang et al. [14] described methods to safely extend role-based access control schemes with delegation primitives. However, role-based access control is only one particular scheme, and delegation is only one particular access control feature. Thus, this work provides no guideline for extending other access control schemes, or using extensions to allow different classes of abilities. In our work, we discuss the general problem of extending access control schemes, and present a particular class of safe and useful extensions, called auxiliary machines.

As a result of the lack of tools for evaluating suitability in access control, there is little work in the field for generating synthetic traces that are representative of an access control application. Thus, for inspiration in designing the access control workload's invocation component (see Section V-A), we turn to work in other domains. In the field of disk benchmarking, Ganger [15] observed that interleaved workloads provided the most accurate approximation of recorded traces. Thus, mechanisms for representing access control workloads must be capable of simulating the interleaved actions of multiple actors. This view is reinforced by the design of IBM's SWORD workload generator for stream processing systems [16]. This work also points out that synthetic workloads need to replicate both volumetric and contextual properties of an execution environment in order to provide an accurate indication of a system's performance within that environment. Thus, we conjecture that access control workloads as well may need to be capable of expressing not only volumetric statistics such as number of documents created, but also contextual statistics such as the type of content in created documents.

Recent work in workflow systems has analyzed the complexity of the workflow satisfiability problem (WSP), which determines whether a workflow can be completed by the participants in the system [17], [18]. This problem turns out to be important for our approach, since our analysis framework includes a simulation procedure that utilizes workflow systems for describing behavior. Without an efficient method of solving WSP, our simulation would suffer either intractability or incorrect behavior.

III. A New Approach

Historically, evaluating the expressive power of access control schemes has allowed researchers to separate schemes into equivalence classes and answer important policy analysis questions. However, absolute assessments tell us very little about the performance and suitability of a particular access control scheme for a given application. In this section, we identify the access control suitability analysis problem, develop a set of requirements that solutions to this problem must satisfy, and overview our solution approach.

A. Problem Definition

Given a formalization of an application's access control requirements, we postulate that assessing the *suitability* of an access control scheme for that application will involve two classes of suitability measures: *expressiveness* and *cost*. As such, suitability analysis is necessarily a two-phased process.

In the first phase, one must ensure that candidate schemes for use within an application are expressive enough to safely meet the needs of the application; that is, whether the candidate schemes can admit *at least* the policies required by the application. In this expressiveness phase, the analyst formalizes the candidate access control schemes, the operations required by the application, and the set of properties that a safe implementation must satisfy. Examples of potential implementation requirements range from simply enforcing the same accesses to ensuring a strict bisimulation over state transitions. Upon completion of this phase, the analyst should be able to narrow down the list of schemes to those that are expressive enough to operate within the application while satisfying all required properties.

The notion of costs, on the other hand, requires examining ordered measures of suitability such as administrative overheads, workflow throughput, or degree of reliance on system extensions (e.g., to increase expressiveness) that result from the choice of a particular candidate access control scheme. In the cost analysis phase, the analyst formalizes the cost measures of interest, the expected usage of the access control system, and the expected costs of individual actions within each scheme. This information can be used to conduct a cost analysis that determines a partial order over the candidate schemes that expresses their relative suitability to the application with respect to the cost measures of interest.

More formally, we address the following problem:

Problem (Suitability Analysis) Given an access control workload W, a set of candidate access control schemes $\mathcal{S} = \{\mathcal{S}_1, \ldots, \mathcal{S}_n\}$, a notion of safe implementation $\mathcal{I}$, and a set of ordered cost measures $\mathcal{C} = \{C_1, \ldots, C_m\}$, determine:

(i) the subset $\mathcal{S}' \subseteq \mathcal{S}$ of schemes that admit implementations of W preserving $\mathcal{I}$

(ii) the schemes within $\mathcal{S}'$ whose cost assessments are optimal within the lattice $C_1 \times \cdots \times C_m$ $\diamondsuit$

B. Solution Requirements

We now explore requirements for suitability analysis frameworks. First, we consider requirements in how an access control workload (W) is represented. These requirements ensure that a suitability analysis framework is capable of modeling the tasks carried out within an organization, and the interactions required to support and process these tasks. Considering both facets of a workload is critical, as neither one alone can fully parameterize the behavior of an organization.[1]

- *Domain exploration:* Large organizations are complex systems with subtle interactions. The emergent behaviors of such systems may not be captured during the static process of workload specification. It must be possible to efficiently explore many initial conditions (e.g., types of actors, operations supported, organization size, and operation distributions) to examine the effects of various levels of concurrency and resource limitation.
- *Cooperative interaction:* Tasks within large organizations typically require the interaction of many individuals. To model these interactions, a suitability analysis framework should support the use of operational workflows, as well as constraints on their execution (e.g., to model separation or binding of duty).

Next, we ensure that the analyst is able to tune the suitability analysis framework to meet the specific needs of her application. believe that For maximum flexibility, it must be possible to choose the metrics used to assess the suitability of an access control scheme for a given workload. This should include both the binary metrics used in expressiveness evaluation ($\mathcal{I}$) and the ordered metrics used in cost evaluation ($\mathcal{C}$).

- *Tunable safety:* Given a particular workload and scheme, there may be many different ways for the scheme to implement the workload. Without enforcing structure on the mapping encoding this implementation, even the most under-expressive schemes can appear to implement a workload [5]. However, as mentioned in Section III-A,

[1]For instance, the well-documented shortcomings of the U.S. military's access control scheme result not from some core inability to process data, but instead from overheads associated with scaling these processes to support high volumes of data and dynamic sharing patterns [19], [20].

the particular properties that any given implementation is required to satisfy will depend on the application in which the access control system will be utilized.

- *Tunable cost:* There is no single notion of cost that is sensible for use in every analysis instance. As evidenced by a recent NIST report [9], the costs that are relevant in evaluating access control schemes are very application-dependent. Any suitability analysis framework should be flexible enough to represent many types of costs, including computational, communication, and administrative costs. It must also be possible to examine multiple notions of cost simultaneously during an analysis.

Finally, we consider requirements that ensure that the suitability analysis framework remains practical to use—in terms of runtime efficiency and accuracy—even for large-scale application workloads.

- *Tractability:* Steps of the analysis process that can be automated should be done so using tractable (e.g., polynomial time or fixed-parameter tractable) algorithms that remain feasible to use even for very large systems.
- *Accuracy:* In many cases, full exploration of all possible system traces for the purposes of cost analysis (e.g., via model checking) will be impractical. As such, it must be possible to approximate the expected error of costs obtained by exploring only a subset of these traces.

We have allowed these requirements to drive the development of our suitability analysis framework, and will thus refer to them when justifying various design decisions throughout the following sections. We discuss our framework's success in achieving each of these requirements in Section VIII-A.

C. Framework Overview

Figure 1 presents a overview of the technical approach that we propose for analyzing instances of the access control suitability analysis problem. The first phase of this process is largely manual, and begins by capturing the requirements of the application in what we call an *access control workload*. The workload includes a state machine that formalizes the application's required protection state and supported commands and queries (Section IV-B). In addition, the workload contains a specification of the expected utilization patterns of this functionality, encoding individual behaviors using actor-based probabilistic models, and collaborative tasks via constrained workflows (Section V-A). Candidate access control schemes are then specified as state machines, using the same formalism as the operational aspects of the access control workload (Section IV-A).

The representational similarity between the workload's operational description and the candidate schemes grants us the ability to construct *implementations* of the workload. This is done by mapping states, commands, and queries in the workload to states, (sequences of) commands, and queries in the candidate schemes (e.g., $\sigma_{\mathcal{S}}$ and $\sigma_{\mathcal{T}}$ in Fig. 1). Security properties that must be upheld by a workload implementation can be expressed as constraints on these mappings, and proofs are manually constructed by the analyst indicating that these properties are upheld (Section IV-C). In this way, the process of constructing implementations is a conceptual extension of prior work in expressive power analysis (e.g., [3], [4], [6]–[8], [13]). It may be necessary for a candidate scheme to be augmented to support such a safe implementation (Section VI). The result of the first phase of analysis determines goal (i) of the Suitability Analysis problem: the subset of schemes that admit implementations of the workload while preserving the requisite security properties.

After the initial specification of cost measures to quantify the costs of interest to the analyst (Section V-B) and cost functions to assign cost distributions to actions taken within each candidate access control scheme (Section V-C), Phase 2 of the suitability analysis process is largely automated. Specifically, our approach makes use of Monte Carlo simulation to carry out cost analysis: input parameters (e.g., number of users, frequency of execution for various types of processes, etc.) are sampled from appropriate distributions, the actor-based probabilistic model of workload utilization is walked to generate concurrent traces of workload activities, these activities are mapped to (sequences of) actions in each candidate scheme being analyzed, actions are carried out, and the resulting costs are aggregated. This process is repeated until either (i) adequate coverage of the input space is obtained or (ii) adequate confidence intervals can be placed on the costs for specific points within this input space (Section V-D). The result of this phase of analysis determines goal (ii) of the Suitability Analysis Problem: the set of schemes whose cost assessments are optimal within the lattice formed by the collection of all costs measures.

IV. Phase 1: Expressiveness Evaluation

In this section, we discuss the first phase of suitability analysis, expressiveness evaluation. In this phase, the analyst formalizes the workload and the candidate access control schemes, then constructs expressiveness mappings to ensure that each scheme has the expressiveness necessary to properly implement the workload.

A. Formalizing Access Control Schemes

At the heart of an access control system is the access control *model*, the collection of data structures used to store the information needed to make access control decisions. An access control model is formalized as a set of *states*, the possible configurations of these data structures. An access control *scheme*, then, defines the set of commands and queries that can be used to interact with the model's states. Lastly, an access control *system* is an instantiation of a scheme, defining the subset of the scheme's commands that are immediately available, as well as an initial state. Previous work has shown distinctions in expressiveness between schemes with identical models but different commands [4], [5] or queries [5], [8]. However, there seems to be little benefit in including a system's initial state in an analysis, since generalizing over *all* states allows us to make stronger claims about its properties. Thus, in this paper, our analysis considers access control *schemes*.

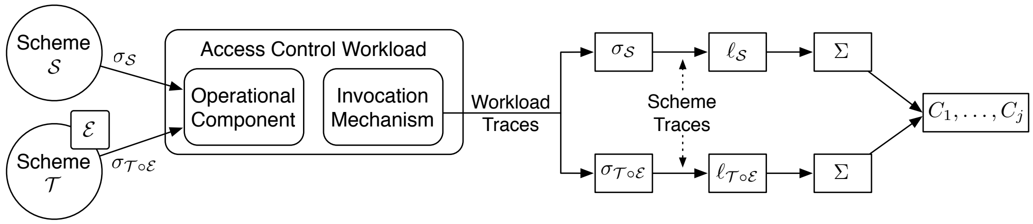

Fig. 1: Overview of an application-aware analysis framework for access control

Our particular formalism is adapted from prior work [5], and represents an access control scheme as a state transition system operating over a set of protection states Γ. States in Γ contain all information necessary for the operation of the access control scheme (e.g., sets of principals, objects, roles, etc.) *Queries* provided by the scheme enable inspection of this state, while *commands* enable transitions between states. We now more formally define these concepts.

Definition 1 (Query) Given a set of access control states, Γ, an *access control query* over Γ is a question that can be asked of an access control system, defined as $q = \langle n, P, \vdash \rangle$, where:

- n names the query
- $P = \langle P_1, \ldots, P_j \rangle$ is the set of parameter spaces from which the query's j parameters are drawn (e.g., the set of subjects, objects, roles), where $p_1 \in P_1$ represents the entity executing the query.[2] We denote $P_1 \times \cdots \times P_j$ as P^*.
- $\vdash : \Gamma \times P^* \to \{\text{TRUE}, \text{FALSE}\}$ is the entailment relation that maps each state and parameterization to a truth value, asserting that truth value of the query in the given state with the given parameters ◊

Given access control query $q = \langle n_q, P_q, \vdash_q \rangle$ over Γ, state $\gamma \in \Gamma$, and parameterization $p \in P_q^*$, we say that $\gamma \vdash q(p_1, \ldots, p_j)$ to indicate that $\vdash_q(\gamma, p_1, \ldots, p_j) = \text{TRUE}$. To better explain the intuition behind queries, we present the following example.

Example 1 Consider the access control state γ, in which Alice has no access to user Bob's document, `foo`. Bob may choose to verify whether Alice has access to `foo` by asking a query where $n =$ `can_read`, which takes parameters from spaces $\langle U, U, D \rangle$, the sets of users, users, and documents, respectively, and whose entailment $\vdash$ maps γ and the parameterization `(Bob, Alice, foo)` to FALSE, indicating that Alice does not have the access in question. ◊

While queries are used to inspect access control states, commands are used to modify these states.

Definition 2 (Command) Given a set of access control states, Γ, an *access control command* over Γ is the mechanism for state transformations, defined as $c = \langle n, P, e \rangle$, where:

- n names the command
- $P = \langle P_1, \ldots, P_j \rangle$ is the set of parameter spaces from which the command's j parameters are drawn (e.g., the set of subjects, objects, roles), where $p_1 \in P_1$ represents the entity executing the command. We denote $P_1 \times \cdots \times P_j$ as P^*.
- $e : \Gamma \times P^* \to \Gamma$, the effect mapping, which maps each state and parameterization to the state that results from the execution of the command with the given parameters in the given state. ◊

We now give an example command to clarify Definition 2.

Example 2 Consider the state γ from Example 1. Bob may choose to grant Alice read access to `foo` by executing a command where $n =$ `grant_read`, which takes parameters from spaces $\langle U, U, D \rangle$, the sets of users, users, and documents, respectively, and whose effect mapping e maps γ and the parameterization `(Bob, Alice, foo)` to γ', an identical state with the exception of Alice being granted read access to `foo`.◊

Given a set of access control commands Ψ over Γ, and two states $\gamma, \gamma' \in \Gamma$, we say that $\gamma \mapsto_\Psi \gamma'$ if there exists a command $\psi = \langle n, P, e \rangle \in \Psi$ and a parameterization $p \in P^*$ such that $e(\gamma, p) = \gamma'$. We use $\gamma \overset{*}{\mapsto}_\Psi \gamma'$ to denote the transitive closure of $\mapsto_\Psi$: i.e., there exists a sequence of commands $\langle \psi_1 = \langle n_1, P_1, e_1 \rangle, \ldots, \psi_k = \langle n_k, P_k, e_k \rangle \rangle$ and a sequence of parameterizations $\langle p_1 \in P_1^*, \ldots, p_k \in P_k^* \rangle$ of these commands such that $e_k(\ldots e_1(\gamma, p_1), \ldots, p_k) = \gamma'$. We can now precisely formalize an access control scheme.

Definition 3 (Scheme) An *access control scheme* is a state transition system $\mathcal{S} = \langle \Gamma, \Psi, Q \rangle$, where Γ is the set of access control states, Ψ is the set of commands over Γ, and Q is the set of queries over Γ. ◊

We now give an example to demonstrate the structure.

Example 3 DAC is the discretionary access control scheme, defined by $\mathcal{D} = \left\langle \Gamma^{\mathcal{D}}, \Psi^{\mathcal{D}}, Q^{\mathcal{D}} \right\rangle$. Its states, $\Gamma^{\mathcal{D}}$, are defined by the sets $\langle S, O, R, M \rangle$, where:

- S is the set of subjects
- O is the set of objects
- R is the set of rights
- $M : S \times O \to 2^R$ is the access matrix

DAC's commands, $\Psi^{\mathcal{D}}$, include the following.

- `CreateObject(`S`, `O`)`, which adds an object

[2]The first parameter of a query or command represents the executing entity. For queries, this allows the access control scheme to respond differently to different queriers (e.g., a user may not be allowed to find out the existence of another user's documents). For commands, this allows the scheme to determine whether the requested execution is allowed.

- `DestroyObject(`S`, `O`)`, which deletes an object
- `CreateSubject(`S`, `S`)`, which adds an subject
- `DestroySubject(`S`, `S`)`, which deletes an subject
- `Grant(`S`, `S`, `O`, `R`)`, which grants a right over an object to a subject
- `Revoke(`S`, `S`, `O`, `R`)`, which revokes a right over an object from a subject

Finally, DAC's queries, $Q^{\mathcal{D}}$, include the following.

- `Access(`S`, `S`, `O`, `R`)`, which asks whether a user has a right over an object
- `SubjectExist(`S`, `S`)`, which asks whether a subject exists ◊

B. Formalizing Workloads

An access control workload describes an abstraction of the access control needs of an environment. A workload specifies both an *operational* component describing the relevant operations that must be supported, as well as an *invocation* component that describes how those operations are expected to be used. The operational component can be viewed as the collection of high-level commands and queries that the application would like to execute, and hence can be formalized as an (abstract) access control state machine using Definition 3. We note that, while formalized in the same way, workloads and schemes differ in their intention. While a scheme represents a functioning piece of software, a workload is built by the analyst to represent the higher-level *desired* functionality of a system, without necessarily being appropriate for direct implementation. We discuss possible ways to more formally express this difference in intention in Section VIII.

The invocation component describes the ways in which the system is typically used; i.e., the order in which the high-level commands and queries are executed. At a minimum, the invocation component should be able to dictate the probabilities with which various commands are executed and queries are asked during paths of execution. Our framework allows the invocation component to remain flexible. We discuss the invocation mechanism and present a particular instantiation of this concept in Section V-A.

Definition 4 (Workload) An *access control workload* is defined by $\langle \mathcal{S}, I^{\mathcal{S}} \rangle$, where:

- $\mathcal{S} = \langle \Gamma, \Psi, Q \rangle$ is an abstract access control scheme and acts as the operational component
- $I^{\mathcal{S}}$ is an invocation mechanism over $\mathcal{S}$, e.g., an instance of Definition 11 (cf. Section V-A) ◊

Note that it is not always obvious how to transform an abstract description of a desired access control policy into a machine-level specification for use as the scheme component of a workload. We discuss this problem in Section VIII. We now give an example of a workload operational component.

Example 4 Consider an environment that grants users discretionary control over their own resources, but allows administrators to have full access to any object. This workload, $W_{\mathcal{A}} = \langle \mathcal{A}, I^{\mathcal{A}} \rangle$, utilizes as its operational component the administrative DAC scheme (ADAC), $\mathcal{A}$. The ADAC scheme is similar to the DAC scheme from Example 3, but must also maintain the set of administrators, who have full access to each object in the system. It is defined as $\mathcal{A} = \langle \Gamma^{\mathcal{A}}, \Psi^{\mathcal{A}}, Q^{\mathcal{A}} \rangle$. Its states, $\Gamma^{\mathcal{A}}$, are defined by the sets $\langle S, A, O, R, M \rangle$, where:

- S is the set of subjects
- $A \subseteq S$ is the set of administrators
- O is the set of objects
- R is the set of rights
- $M : S \times O \to 2^R$ is the access matrix

ADAC's commands, $\Psi^{\mathcal{A}}$, include the following.

- `CreateObject(`S`, `O`)`, which adds an object
- `DestroyObject(`S`, `O`)`, which deletes an object
- `CreateSubject(`S`, `S`)`, which adds an subject
- `DestroySubject(`S`, `S`)`, which deletes an subject
- `Grant(`S`, `S`, `O`, `R`)`, which grants a right over an object to a subject
- `Revoke(`S`, `S`, `O`, `R`)`, which revokes a right over an object from a subject
- `GrantAdmin(`S`, `S`)`, which grants administrative status
- `RevokeAdmin(`S`, `S`)`, which revokes administrative status

Finally, ADAC's queries, $Q^{\mathcal{A}}$, include the following.

- `Access(`S`, `S`, `O`, `R`)`, which asks whether a user has a right over an object
- `SubjectAdmin(`S`, `S`)`, which asks whether a subject is an administrator
- `SubjectExist(`S`, `S`)`, which asks whether a subject exists message ◊

C. Implementing a Workload in a Scheme

Once the analyst selects an appropriate set of candidate access control schemes, she must verify each scheme's ability to safely execute the operations required by the workload. To do so, the analyst demonstrates the existence of mappings from the workload's operational component to each of the candidate access control schemes. These mappings provide a translation from the workload's state representation and actions to those of each candidate scheme. Moreover, these mappings are used to guarantee that the safety properties of the workload are preserved in each candidate scheme.

Definition 5 (Implementation) Given an access control workload $W = \langle \mathcal{W}, I^{\mathcal{W}} \rangle$ in which $\mathcal{W} = \langle \Gamma^{\mathcal{W}}, \Psi^{\mathcal{W}}, Q^{\mathcal{W}} \rangle$, and an access control scheme $\mathcal{S} = \langle \Gamma^{\mathcal{S}}, \Psi^{\mathcal{S}}, Q^{\mathcal{S}} \rangle$, an *implementation* of $\mathcal{W}$ in $\mathcal{S}$ is a set of mappings $\sigma = \langle \sigma_\Gamma, \sigma_\Psi, \sigma_Q \rangle$, where:

- $\sigma_\Gamma : \Gamma^{\mathcal{W}} \to \Gamma^{\mathcal{S}}$ is the state mapping
- $\sigma_\Psi : \Psi^{\mathcal{W}} \to \left(\Psi^{\mathcal{S}}\right)^+$ is the command mapping (each $\psi \in \Psi^{\mathcal{W}}$ is mapped to a sequence $\langle \psi_1, \ldots, \psi_k \rangle$, where each ψ_i is a command in $\Psi^{\mathcal{S}}$)
- $\sigma_Q : Q^{\mathcal{W}} \to Q^{\mathcal{S}}$ is the query mapping ◊

While Definition 5 describes the structure of an implementation, the properties that such an implementation must satisfy are defined by the application in question. One particularly natural set of properties that an implementation might be required to preserve is the set of *compositional security analysis instances* [5]. The compositional security analysis instance is a generalization of simple safety analysis [1] to arbitrary quantified boolean formulas over queries.

Definition 6 (Compositional Security Analysis) Given an access control scheme $\mathcal{S} = \langle \Gamma, \Psi, Q \rangle$, a *compositional security analysis instance* has the form $\langle \gamma, \varphi, \Pi \rangle$, where $\gamma \in \Gamma$ is a state, φ is a propositional formula over Q, and $\Pi \in \{\exists, \forall\}$ is a quantifier. If $\Pi = \exists$, the instance asks whether there exists $\gamma' \in \Gamma$ such that $\gamma \overset{*}{\mapsto}_\Psi \gamma'$ and $\gamma' \vdash \varphi$ (whether φ is *possible*). If $\Pi = \forall$, the instance asks whether for every $\gamma' \in \Gamma$ such that $\gamma \overset{*}{\mapsto}_\Psi \gamma'$, $\gamma' \vdash \varphi$ (whether φ is *necessary*). ◊

The compositional security analysis instance is a natural language for expressing many types of practical policies (e.g., "Bob cannot edit payroll data while his wife, Alice, is also an employee." [5]). An implementation that preserves all compositional security analysis instances is said to be *strongly security-preserving*. Unfortunately, directly proving that a mapping is strongly security-preserving can be quite expensive, as it requires the analysis of all possible compositional security analysis instances. For this reason, Tripunitara and Li presented the *state-matching reduction* [5], a type of mapping that is defined by a set of structural properties that are necessary and sufficient for being strongly security-preserving. Using the state-matching reduction is advantageous, as it is easier to prove that a mapping satisfies these structural requirements than it is to directly prove that it preserves all compositional security analysis instances. We now present the state-matching implementation, a type of implementation based on (and maintaining the security properties of) Tripunitara and Li's state-matching reduction.

Definition 7 (State-Matching Implementation) Given an access control workload $W = \langle \mathcal{W}, I^{\mathcal{W}} \rangle$ in which $\mathcal{W} = \langle \Gamma^{\mathcal{W}}, \Psi^{\mathcal{W}}, Q^{\mathcal{W}} \rangle$, an access control scheme, $\mathcal{S} = \langle \Gamma^{\mathcal{S}}, \Psi^{\mathcal{S}}, Q^{\mathcal{S}} \rangle$, and an implementation $\sigma = \langle \sigma_\Gamma, \sigma_\Psi, \sigma_Q \rangle$ of $\mathcal{W}$ in $\mathcal{S}$, we say that two states $\gamma^{\mathcal{W}}$ and $\sigma_\Gamma(\gamma^{\mathcal{W}}) = \gamma^{\mathcal{S}}$ are *equivalent* with respect to the implementation σ (and denote this equivalence as $\gamma^{\mathcal{W}} \sim_\sigma \gamma^{\mathcal{S}}$) when for every $q^{\mathcal{W}} = \langle n, P, \vdash \rangle \in Q^{\mathcal{W}}$ (with $q^{\mathcal{S}} = \sigma_Q(q^{\mathcal{W}})$) and every $p^{\mathcal{W}} \in P^*$ (with $p^{\mathcal{S}} = \sigma_\Gamma(p^{\mathcal{W}})$), $\gamma^{\mathcal{W}} \vdash q^{\mathcal{W}}(p^{\mathcal{W}})$ if and only if $\gamma^{\mathcal{S}} \vdash q^{\mathcal{S}}(p^{\mathcal{S}})$.

An implementation σ of $\mathcal{W}$ in $\mathcal{S}$ is said to be a *state-matching implementation* if for every $\gamma^{\mathcal{W}} \in \Gamma^{\mathcal{W}}$, with $\gamma^{\mathcal{S}} = \sigma_\Gamma(\gamma^{\mathcal{W}})$, the following two properties hold:

1) For every state $\gamma_1^{\mathcal{W}} \in \Gamma^{\mathcal{W}}$ such that $\gamma^{\mathcal{W}} \overset{*}{\mapsto}_{\Psi^{\mathcal{W}}} \gamma_1^{\mathcal{W}}$, there exists a state $\gamma_1^{\mathcal{S}} \in \Gamma^{\mathcal{S}}$ such that $\gamma^{\mathcal{S}} \overset{*}{\mapsto}_{\Psi^{\mathcal{S}}} \gamma_1^{\mathcal{S}}$ and $\gamma_1^{\mathcal{W}} \sim_\sigma \gamma_1^{\mathcal{S}}$.
2) For every state $\gamma_1^{\mathcal{S}} \in \Gamma^{\mathcal{S}}$ such that $\gamma^{\mathcal{S}} \overset{*}{\mapsto}_{\Psi^{\mathcal{S}}} \gamma_1^{\mathcal{S}}$, there exists a state $\gamma_1^{\mathcal{W}} \in \Gamma^{\mathcal{W}}$ such that $\gamma^{\mathcal{W}} \overset{*}{\mapsto}_{\Psi^{\mathcal{W}}} \gamma_1^{\mathcal{W}}$ and $\gamma_1^{\mathcal{W}} \sim_\sigma \gamma_1^{\mathcal{S}}$. ◊

The following proposition demonstrates the power of this notion of implementation. The proof of Proposition 1 can be found in Appendix A-A.

Proposition 1 *Given an access control workload $W = \langle \mathcal{W}, I^{\mathcal{W}} \rangle$ in which $\mathcal{W} = \langle \Gamma^{\mathcal{W}}, \Psi^{\mathcal{W}}, Q^{\mathcal{W}} \rangle$, an access control scheme, $\mathcal{S} = \langle \Gamma^{\mathcal{S}}, \Psi^{\mathcal{S}}, Q^{\mathcal{S}} \rangle$, and an implementation $\sigma = \langle \sigma_\Gamma, \sigma_\Psi, \sigma_Q \rangle$ of $\mathcal{W}$ in $\mathcal{S}$, σ is a state-matching implementation if and only if it is strongly security-preserving; that is, every compositional security analysis instance in $\mathcal{W}$ is true if and only if the image of the instance under σ is true in $\mathcal{S}$.*

PROOF (SKETCH) In proving Proposition 1, we utilize a previous result from Tripunitara and Li [5], which states that a mapping is a state-matching *reduction* if and only if it is strongly security-preserving. We show that, if an implementation σ is a state-matching implementation of $\mathcal{W}$ using $\mathcal{S}$, there exist $\mathcal{W}'$ and $\mathcal{S}'$, schemes under Tripunitara and Li's definition that are equivalent to $\mathcal{W}$ and $\mathcal{S}$, respectively. We prove that this equivalence is strongly security-preserving, and then than the implementation corresponds to a state-matching reduction, σ', from $\mathcal{W}'$ to $\mathcal{S}'$. This proves that σ' is strongly security-preserving, and finally that σ is. (This ends the "if" direction.)

For the "only if" direction, we consider an implementation that is strongly security-preserving. Again, we show that this implementation corresponds to a Tripunitara-Li mapping, this time deducing a state-matching reduction from the strongly security-preserving mapping. We show that this state-matching reduction is equivalent to our implementation, and thus that the implementation is a state-matching implementation. □

As a result of these security guarantees, we will require that, in order for an implementation of workload W in scheme $\mathcal{S}$ to be considered a *safe implementation*, that implementation be state-matching. In addition, in this work we restrict implementations to preserve the semantics of the respective model with respect to accesses. We accomplish this by requiring the query mapping to map the access queries in the workload to the access queries in the scheme, thus forcing the implementation of the workload to use the same procedure for deciding accesses as the scheme uses. These restrictions on implementations are examples of how an application can dictate how a scheme can be used for the purpose of "simulating" a workload. Previous work on various expressiveness properties (e.g., [21]) can provide guidelines for choosing the type of implementation that best suits expressiveness evaluation in the context of the relevant application. Exploring the full range of possible implementation properties and their corresponding implementation structure is a subject of future work (see Section VIII-B).

Consider ADAC from Example 4 and DAC from Example 3. Despite the similarities between them, DAC does not seem to admit a state-matching implementation of ADAC using DAC, since DAC is unable to maintain information about the set of administrators. For scenarios such as this, we explore the ability to *extend* schemes in Section VI.

V. Phase 2: Cost Analysis

As discussed in Section III, our approach to suitability analysis is two-phased. In the previous section, we discussed the first phase, expressiveness evaluation, which allows the analyst to ensure that all candidate schemes are expressive enough to safely meet the needs of the application. In this section, we present the details of the second phase, cost analysis, which explores more quantitative suitability measures.

A. Actor-based Invocation Mechanism

Recall from Section IV-B that an access control workload, $W = \langle \mathcal{W}, I^{\mathcal{W}} \rangle$, consists of an operational component, $\mathcal{W}$, and an invocation mechanism over $\mathcal{W}$. The invocation mechanism describes the expected usage of the access control system within the application being described. Most simply, this mechanism could be a recorded trace of operations that will be "played back" while its operating costs are recorded. However, this violates several of the requirements from Section III-B. For example, *Domain Exploration* requires that we are able to alter input parameters. While this type of static invocation mechanism does not preclude the varying of the initial access control state, it does not allow the trace to react to these changes (e.g., more users typically means more frequent execution of commands and queries).

To overcome these types of issues, we define an invocation mechanism utilizing the concepts of *actors* carrying out *actions* within the system. Actors are human users, daemons, and other entities that act on the access control system. We determine the set of actors by extracting the active entities from an access control state. We express the various ways in which actors cooperate to complete a task using *constrained workflows*. Within this structure, *workflows* express the dependency between related actions, and *constraints* express the restrictions placed on which user can execute each action in a task. Finally, *actor machines* express the behavior models for the actors within the constraints imposed upon them by the constrained workflow. Together, these structures enable the modeling and simulation of complex and concurrent behaviors of the entities that are active within a given workload.

We now formalize the notion of an *action*, which is the basic component of work executed by an actor in the system. An action is a partially parameterized command or query. The free parameters are assigned statically by the executing actor's behavior machine or dynamically during execution.

Definition 8 (Action) Let $\mathcal{S} = \langle \Gamma, \Psi, Q \rangle$ be an access control scheme and $\mathfrak{V}$ a set of variable symbols. An *access control action* from scheme $\mathcal{S}$ is defined as $\alpha = \langle n, a, C \rangle$, where:

- n names the action
- $a \in \Psi \cup Q \cup \{\varnothing\}$ is the command or query (whose set of parameter spaces is $P = \langle P_1, \ldots, P_j \rangle$) that the action executes. A value of $\varnothing$ indicates that the action does not execute a command or query in the access control system.
- $C \in (P_1 \cup \mathfrak{V}) \times \ldots \times (P_j \cup \mathfrak{V}) \cup \{\varnothing\}$ is the partial parameterization. For each parameter in P, C specifies a parameter value or a variable from $\mathfrak{V}$. For actions that do not execute a command or query, C, like a, is $\varnothing$. ◊

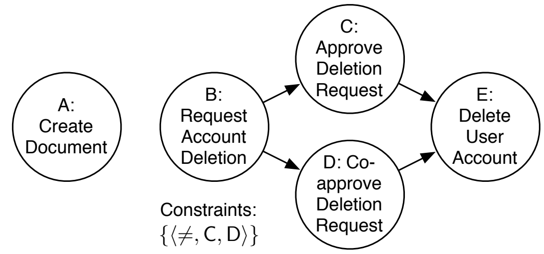

Fig. 2: An example of a constrained workflow

Although actions that do not execute commands or queries within a scheme seem counter-intuitive, they become important in the context of workflows that link together multi-user tasks within a workload. To describe various dependencies between actions (executed by a single actor or a set of actors in coordination), we present the notion of a *constrained access control workflow*, which organizes the execution of actions. Formally, this structure specifies the partial order describing action dependence as well as a set of constraints that restrict the set of users that can execute various actions.

Definition 9 (Constrained Workflow) Let $\mathcal{S} = \langle \Gamma, \Psi, Q \rangle$ be an access control scheme and $\mathfrak{A}$ a set of access control actors within $\mathcal{S}$. We say that $W = \langle A, \prec, C \rangle$ is an *constrained access control workflow* over the scheme $\mathcal{S}$, where:

- A is the set of actions from scheme $\mathcal{S}$
- $\prec \subset A \times A$ is the partial order describing the dependency relation between actions. If $\alpha_1 \prec \alpha_2$, then α_2 depends on α_1. That is, α_2 cannot be executed unless a corresponding execution of α_1 has occurred.
- C is the set of constraints, where each constraint is of the form $\langle \rho, \alpha_1, \alpha_2 \rangle$. Here, ρ is a binary operator of the form $\mathfrak{A} \times \mathfrak{A} \to \{\text{TRUE}, \text{FALSE}\}$. A constraint restricts execution of actions α_1 and α_2 to actors who satisfy the binary operator ρ. For example, $\langle \neq, \alpha_1, \alpha_2 \rangle$ says that α_1 and α_2 must be executed by different actors. ◊

Within a workflow $\langle A, \prec, C \rangle$, subsets of A that are pairwise disjoint with respect to $\prec$ are referred to as *tasks*, and each action within a task is referred to as a *step* in that task.

Example 5 Figure 2 displays a constrained workflow that includes two tasks, corresponding to document creation and account deletion. The former is a degenerate task containing a single action. Execution of this action is thus effectively unconstrained by the workflow. However, the task of deleting an account requires the approval of two different administrators. The workflow allows administrators to approve deletion of accounts only after the deletion request, and the deletion can only happen after it has been approved twice. Furthermore, the example constraint requires that the two approval actions be executed by two different administrators. ◊

We draw a distinction between the use of workflows here and their use in, e.g., R²BAC [17]. While there exist access control schemes with the native ability to enforce workflow semantics, our goal is to represent workflow properties at the access control workload level, and utilize implementations of these workloads to ensure tasks execute according to these higher-level constraints. This allows us to utilize even simple access control schemes while still constraining actors to work within such organizational policies as separation of duty.

To describe the patterns with which actors execute actions, we employ *actor machines*, which are state machines that describe each actor's behavior. Each state in the machine is labeled with an action name and a refining parameterization (which assigns values to parameters that were left as variables in the action specification). Transitions in this state machine are labeled with *rates* akin to those used in continuous-time Markov processes (e.g., [22]). We then generate representative traces of actor behavior by probabilistically walking this machine, following transitions with probabilities proportional to their rates.

Definition 10 (Actor Machine) Let $\mathcal{S} = \langle \Gamma, \Psi, Q \rangle$ be an access control scheme, $W = \langle A, \prec, C \rangle$ a constrained workflow over $\mathcal{S}$, and $\mathfrak{V}$ a set of variable symbols. An *actor machine* for $\mathcal{S}$ and G is the state machine $\langle S, \Phi, R \rangle$, where:

- S is the set of states
- $\Phi : S \to A \times (P_1 \cup \mathfrak{V}) \times \ldots \times (P_j \cup \mathfrak{V})$ labels each state with an action and a *refinement* of the action's parameterization (i.e., parameters assigned by the action remain the same, while parameters not assigned by the action may be assigned to values)
- $R : S \times S \to \mathbb{R}$ is the set of rates of transitioning from state to state ◊

The semantics of the execution of an actor machine are as follows. R describes the rates of transitioning from one state to another. In order to achieve the Markov property, the time spent waiting to exit a state is exponentially distributed, with rate parameter proportional to the sum of the rates of all exiting transitions. When executing, an actor carries out a state's action upon *entering* the state. We distinguish between *entering* a state and *remaining in* a state. Transitioning from a state back to itself will result in a re-execution of the state's action. Remaining in a state while waiting for the next transition to trigger will *not* result in a re-execution.

Example actor machines are demonstrated in Fig. 3. In this example, we classify users into two categories of actors: administrators and non-administrators. The former add users and approve and execute user deletions, while the latter generate documents and occasionally request to be deleted. Due to the labeled rates on this machine, each administrator creates users at the expected rate of one per month, and roughly 10% of non-administrative users request deletion each month. High rates on transitions leading to, e.g., approving deletions indicate the rate at which these actions will be executed *when enabled*. Transitions labeled with ∞ occur immediately after completing the preceding action.

Our actor-based invocation mechanism that will complete

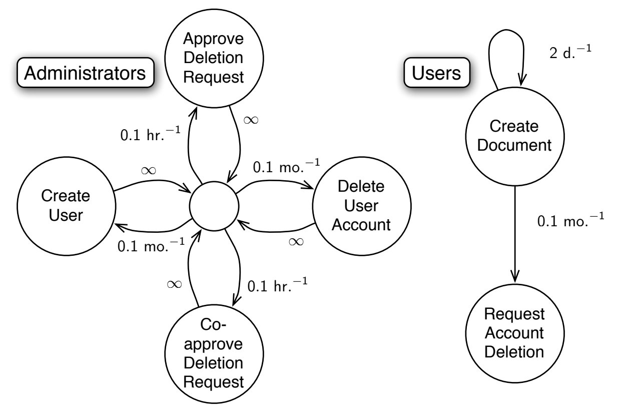

Fig. 3: Example actor machines

Definition 4, then, consists of a constrained workflow, a set of actor machines, and a method for extracting the current actors and their assigned machines from an access control state.

Definition 11 (Actor-Based Invocation) Let $\mathcal{S} = \langle \Gamma, \Psi, Q \rangle$ be an access control scheme. We say that $I^{\mathcal{S}} = \langle W, \mathfrak{A}, A, G_A, g \rangle$ is an *constrained, actor-based access control invocation mechanism* over the scheme $\mathcal{S}$, where:

- W is a constrained workflow over $\mathcal{S}$
- $\mathfrak{A}$ is the set of all actors
- $A : \Gamma \to \wp(\mathfrak{A})$ is the actor relation, mapping each access control state to the set of actors active in that state
- G_A is the set of actor machines
- $g : \mathfrak{A} \to G_A$ is the actor machine assignment, mapping each actor to its actor machine ◊

B. *Cost Measures*

An important part of cost analysis is choosing relevant cost measures. These measures should be representative of the "problem" (i.e., what types of cost the analyst cares about), while also enabling the definition of a cost function for each candidate scheme (see Section V-C). For example, while "operational cost per day" may be representative of access control evaluation goals in industry, it is hard to assign costs in this measure to each fully parameterized access control action. A measure such as "average administrative personnel-hours spent per access control operation," on the other hand, is more easily quantified and enables the same types of analyses.

In this paper, we make no commitment to any particular cost measures but rather develop an analysis framework that operates on any measure satisfying a number of simple properties. A cost measure must include a set of elements representing the costs, an associative and commutative operator that combines two costs to produce another cost (e.g., addition), and a partial order for comparing costs. Finally, we enforce that there are no "negative" costs.

Definition 12 (Cost Measure) A *cost measure* is defined by the ordered abelian monoid $\mathbf{G} = \langle G, \bullet, \preceq \rangle$, where G is the set of costs, $\bullet$ is the closed, associative, commutative accrual operator over G with identity 0_G, and $\preceq$ is a partial order over G such that $\forall a, b \in G : a \preceq a \bullet b \wedge b \preceq a \bullet b$. ◊

Definition 12 can be used to encode a variety of interesting access control measures, including several of those noted in a recent NIST report on the assessment of access control schemes [9]. For example, costs like "steps required for assigning and dis-assigning user capabilities" and "number of relationships required to create an access control policy" can be represented using the cost measure $\langle \mathbb{N}, +, \leq \rangle$. Our notion of measure is general enough to represent many other types of costs as well. Measures for human work such as "personnel-hours per operation" and "proportion of administrative work to data-entry work" can be represented using the cost measures $\langle \mathbb{Z}^+, +, \leq \rangle$ and $\langle \mathbb{Z}^+ \times \mathbb{Z}^+, +, \leq \rangle$, respectively. Maximum memory usage can be represented using $\langle \mathbb{N}, \max, \leq \rangle$.

A common desire is for an analyst to evaluate an access control scheme using several different cost measures in parallel. Thus, we define a *vector* of cost measures.

Definition 13 (Vector of Measures) Given cost measures $\mathbf{N}_1 = \langle N_1, \bullet_1, \preceq_1 \rangle$, $\mathbf{N}_2 = \langle N_2, \bullet_2, \preceq_2 \rangle$, ..., $\mathbf{N}_i = \langle N_i, \bullet_i, \preceq_i \rangle$, let $\mathbf{M} = \langle M, \bullet_*, \preceq_* \rangle$ be the *vector* of cost measures $\mathbf{N}_1, \mathbf{N}_2, \ldots, \mathbf{N}_i$, where:

- $M = N_1 \times N_2 \times \cdots \times N_i$.
- Given $a_1, b_1 \in \mathbf{N}_1$, $a_2, b_2 \in \mathbf{N}_2$, ..., $a_i, b_i \in \mathbf{N}_i$, $\langle a_1, a_2, \ldots, a_i \rangle \bullet_* \langle b_1, b_2, \ldots, b_i \rangle = \langle a_1 \bullet_1 b_1, a_2 \bullet_2 b_2, \ldots, a_i \bullet_i b_i \rangle$.
- Given $a_1, b_1 \in \mathbf{N}_1$, $a_2, b_2 \in \mathbf{N}_2$, ..., $a_i, b_i \in \mathbf{N}_i$, $\langle a_1, a_2, \ldots, a_i \rangle \preceq_* \langle b_1, b_2, \ldots, b_i \rangle$ if and only if $a_1 \preceq_1 b_1 \wedge a_2 \preceq_2 b_2 \wedge \ldots \wedge a_i \preceq_i b_i$. ◊

Definition 13 gives a simple way of combining several measures. As the following proposition states, a vector of cost measures is also a cost measure, enabling the analyst to use a combination of measures within our analysis framework. We prove Proposition 2 in Appendix A-B.

Proposition 2 *Given cost measures* $\mathbf{N}_1, \mathbf{N}_2, \ldots, \mathbf{N}_i$ *and their associated cost vector,* $\mathbf{M}$*,* $\mathbf{M}$ *is a cost measure.*

PROOF (SKETCH) Given the definition of measure, we know that all of $\mathbf{N}_i$ satisfy closure, associativity, identity, and non-negativity. By algebra we show that, given these properties and Definition 13, we can derive closure, associativity, identity, and non-negativity for $\mathbf{M} = N_1 \times N_2 \times \cdots \times N_i$. □

Once a measure is chosen, the analyst must next model how each candidate access control scheme accrues costs using that measure. This requires assigning costs associated with each fully parameterized access control action (command or query execution). Such an assignment is a *cost function*.

C. Cost Functions

In order to calculate the total cost of a particular implementation, costs of executing the various actions within the implementing schemes must be determined. Sometimes, the cost of any execution of a particular command or query is constant (e.g., creating a document requires a constant amount of I/O). In other cases, the parameters of the command or query affect the cost (e.g., adding a user to the system is more expensive for classes of users with greater capabilities). In addition, some costs depend on the current state (e.g., granting access to all documents with a certain property may require inspecting each document, a procedure that grows in cost with the number of documents in the system). Thus, in general, the cost function is required to map each (command, parameterization, state) or (query, parameterization, state) to an element of the relevant cost measure.

Definition 14 (Cost Function) Let $\mathcal{S} = \langle \Gamma, \Psi, Q \rangle$ be an access control scheme, A a set of actions from scheme $\mathcal{S}$, and $\mathbf{G} = \langle G, \bullet, \preceq \rangle$ a cost measure. A *cost function* for $\mathcal{S}$ in $\mathbf{G}$ is a function $\ell^{\mathcal{S}}_{\mathbf{G}} : A \times \Gamma \to G$, which which maps each access control action and state to the member of the cost measure that best represents the costs associated with executing the given action in the given state. ◊

Although most cost functions are infinite (since the number of states and parameterizations are usually infinite), we can often generalize (as mentioned above) for actions whose costs do not depend on the state and/or parameterization. In addition, when state or parameters do affect the cost, the correlation is generally formulaic (e.g., proportional to the size of certain state elements) and is thus simple to describe in a compact way. Finally, in cases where the relation between the state or parameterization and the resulting cost is more complex, we can often take advantage of the simulation-based nature of the cost analysis process and the law of large numbers by abstracting out parameters or state (or both) and reproducing their effect via a probability distribution.

In addition to the cost functions that are of specific interest to the analyst, our simulation process (Section V-D) also requires the specification of each scheme's *time function*. The time function is formalized as a cost function, describing the duration of time required to complete an access control action. The cost measure of this specialized cost function is $\langle \mathbb{R} \times \text{time}, +, \leq \rangle$.

D. Cost Analysis via Monte Carlo Simulation

In Section IV-C, we discussed the construction of implementations, which (in addition to their role in expressiveness evaluation) provide a recipe for using each candidate scheme to execute the access control actions needed by the application of interest. In Section V-A, we discussed an actor-based invocation mechanism, which serves as the second component of the access control workload and provides us with a mechanism for generating traces of access control actions that are characteristic of usage within the desired application. Finally, in Section V-C, we discussed cost functions, including the time function, which allow us to quantify the costs of individual access control actions as well as track the passage of time during the execution of generated traces. Given these inputs, we can utilize an automated cost analysis procedure that generates traces of workload actions, translates these into traces of scheme actions, then calculates the costs of these scheme actions.

Algorithm 1 Cost analysis simulation algorithm

Input: $\mathfrak{S}$, set of candidate schemes
Input: Σ, set of implementations ($\forall \mathcal{S} \in \mathfrak{S} : \sigma_{\mathcal{S}} \in \Sigma$)
Input: $\mathbf{C}$, set of cost measures ($\tau = \langle \mathbb{R} \times \text{time}, +, \leq \rangle \in \mathbf{C}$)
Input: L, set of cost functions ($\forall \mathcal{S} \in \mathfrak{S}, C \in \mathbf{C} : \ell_C^{\mathcal{S}} \in L$)
Input: $I = \langle W, \mathfrak{A}, A, G_A, g \rangle$, invocation mechanism
Input: $\gamma_0 \in \Gamma_{\mathcal{W}}$, start state
Input: T_f, goal time
Input: t, time step

procedure ACCostEvalSim($\mathfrak{S}, \Sigma, \mathbf{C}, L, I, \gamma_0, T_f, t$)
  $\mathbf{S} \leftarrow \{\}$ ▷ Initialize set of running AC systems
  $T \leftarrow 0$ ▷ Initialize master clock
  for all $\mathcal{S} = \langle \Gamma, \Psi, Q \rangle \in \mathfrak{S}$ **do** ▷ Initialize state
    $\mathbf{S} \leftarrow \mathbf{S} \cup \{\mathcal{S}\}$
    $\mathbf{A}_{\mathcal{S}} \leftarrow \{\}$ ▷ Set of running actor machines
    $\gamma_{\mathcal{S}} \leftarrow \sigma_{\mathcal{S}}(\gamma_0)$ ▷ Current state of scheme $\mathcal{S}$
    for all $C \in \mathbf{C}$ **do**
      $c_C^{\mathcal{S}} \leftarrow 0_C$ ▷ Total cost of scheme $\mathcal{S}$ in C
    for all $\alpha \in A(\gamma_{\mathcal{S}})$ **do**
      $\mathbf{A}_{\mathcal{S}} \leftarrow \mathbf{A}_{\mathcal{S}} \cup \{g(\alpha)\}$
      $T_\alpha \leftarrow 0$ ▷ Per-actor clock
  while $T \leq T_f$ **do** ▷ Main loop
    $T \leftarrow T + t$ ▷ Increment clock
    for all $\mathcal{S} \in \mathbf{S}$ **do** ▷ Each AC system
      $K = \{\}$ ▷ Clear action list
      for all $\alpha \in \mathbf{A}_{\mathcal{S}}$ **do** ▷ Choose next actions
        if $T_a < T$ **then** ▷ Check actor busy state
          $\langle k, P_k \rangle \leftarrow$ nextAction($g(\alpha)$)
          if $k \neq \varnothing \wedge$ WSat(k, α, P_k) $\neq \varnothing$ **then**
            $T_a \leftarrow T + \ell_\tau^{\mathcal{S}}(k)$ ▷ Busy state
            $K \leftarrow K \cup \{\langle k, \alpha, P_k \rangle\}$ ▷ Save action
      for all $\langle k, \alpha, P_k \rangle \in K$ **do** ▷ Compile costs
        for all $C \in \mathbf{C}$ **do**
          $c_C^{\mathcal{S}} \leftarrow c_C^{\mathcal{S}} \bullet_C \ell_C^{\mathcal{S}}(\sigma_{\mathcal{S}}(\langle k, \alpha, P_k \rangle))$
        if k is a command **then**
          $\gamma_{\mathcal{S}} \leftarrow \sigma_{\mathcal{S}}(e_k(\gamma_{\mathcal{S}}, P_k))$ ▷ Update state
  for all $\mathcal{S} \in \mathfrak{S}$ **do**
    Log $\left\langle \mathcal{S}, c_{C_1}^{\mathcal{S}}, \ldots, c_{C_m}^{\mathcal{S}} \right\rangle$

Algorithm 1 describes such a simulation procedure. First, each candidate scheme is instantiated as a system. An actor machine is then launched for each actor in the state of each system. During the main loop, the clock is incremented and each actor machine is inspected for the correct action to execute next, as per the execution semantics of the actor machine described in Section V-A. If an action is to be executed by the actor during this time step, a reference monitor for the workflow satisfiability problem (procedure WSat) is consulted to ensure that—with respect to the workflow and constraints—the actor can execute the action without rendering the workflow instance *unsatisfiable*. For independent actions (i.e., those in $\{a : \nexists a', a' \prec a\}$), the workflow instance in question is a new, blank instance added to the pool of partially executed instances. For dependent actions (i.e., those in $\{a : \exists a', a' \prec a\}$), the instance is chosen from the existing instances which belong to the same task as the current action in question.

After all action executions for a time step are collected (and verified by the reference monitor), they are simulated within the access control state and their costs are accrued into a running total for each scheme/cost measure combination. (We note that costs may also be accrued per user, per workflow, etc., by trivially extending Algorithm 1.) The final step in the loop adjusts the set of actors according to changes in the state. Once a specified amount of time has passed in the simulated system (denoted the *goal time*), the main loop breaks and the total costs are output.

To address the requirement of *Tractability*, we present the following theorem regarding the runtime of Algorithm 1. The proof of this theorem utilizes previous work by Wang and Li [17] on the complexity of deciding workflow satisfiability.

Theorem 3 *Assuming that workflow constraints are restricted to the binary operators $\{=, \neq\}$ (i.e., constraints expressing binding of duty and separation of duty)*[3]*, the simulation procedure described in Algorithm 1 is pseudo-polynomial in the number of simulated steps and* FPT *with parameter α, the number of actions in the largest task (i.e., the size of the largest disjoint subgraph of the workflow graph).*

Proof (sketch) By far, the step of Algorithm 1 that dominates its complexity is the call to WSat, as the workflow satisfiability problem (WSP) is NP-complete. The call to WSat is nested within loops which will cause it to be called $S \cdot T \cdot A$ times. By [17], WSP is solvable in $\mathcal{O}(C \cdot A^{\alpha})$, yielding a total complexity of $\mathcal{O}\big(S \cdot C \cdot T \cdot A^{\alpha+1}\big)$, which is in FPT with fixed parameter α (maximum number of actions in a task). □

Algorithm 2 Monte Carlo application of Algorithm 1

Input: $\mathfrak{S}$, set of candidate schemes
Input: Σ, set of implementations ($\forall \mathcal{S} \in \mathfrak{S} : \sigma_{\mathcal{S}} \in \Sigma$)
Input: $\mathbf{C}$, set of cost measures ($\tau = \langle \mathbb{R} \times \text{time}, +, \leq \rangle \in \mathbf{C}$)
Input: L, set of cost functions ($\forall \mathcal{S} \in \mathfrak{S}, C \in \mathbf{C} : \ell_C^{\mathcal{S}} \in L$)
Input: $I = \langle W, \mathfrak{A}, A, G_A, g \rangle$, invocation mechanism
Input: $\Pr(\gamma)$, probability distribution over start states
Input: χ, number of Monte Carlo runs
Input: T_f, goal time
Input: t, time step

procedure ACCostEvalMC($\mathfrak{S}, \Sigma, \mathbf{C}, L, I, \Pr(\gamma), \chi, T_f, t$)
  for all $[1, \chi]$ **do** ▷ Monte Carlo loop
    $\gamma_0 \leftarrow$ random sample from $\Pr(\gamma)$
    ACCostEvalSim$\big(\mathfrak{S}, \Sigma, \mathbf{C}, \mathfrak{L}, I, \gamma_0, T_f, t\big)$

Algorithm 1 executes a single run of the system. We next discuss two approaches to utilizing this algorithm: using the Monte Carlo technique to generate large numbers of data points for trend analysis using scatter plots, and using fixed-sample-size point estimates for calculating cost assessments with a particular confidence interval for a small set of important input configurations. Algorithm 2 demonstrates the former. This algorithm repeatedly calls Algorithm 1 using randomly sampled start states in an attempt to exploit the potentially large variance between executions. An advantage of this approach is the detection of trends across a variety of start states. Furthermore, the repeated execution contributes to the complexity of the full analysis by only a multiplicative factor. As such, Monte Carlo analysis—like single run analysis—is in FPT.

[3]A recent result by Crampton et al. [18] allows the use of a wider range of constraints (including those over organizational hierarchies) while preserving the complexity result. For brevity and simplicity, we consider only $\{=, \neq\}$ as constraint operators in this work.

Algorithm 3 Confidence-bounding application of Algorithm 1

Input: $\mathfrak{S}$, set of candidate schemes
Input: Σ, set of implementations ($\forall \mathcal{S} \in \mathfrak{S} : \sigma_{\mathcal{S}} \in \Sigma$)
Input: $\mathbf{C}$, set of cost measures ($\tau = \langle \mathbb{R} \times \text{time}, +, \leq \rangle \in \mathbf{C}$)
Input: L, set of cost functions ($\forall \mathcal{S} \in \mathfrak{S}, C \in \mathbf{C} : \ell_C^{\mathcal{S}} \in L$)
Input: $I = \langle W, \mathfrak{A}, A, G_A, g \rangle$, invocation mechanism
Input: γ_0, start state
Input: T_f, goal time
Input: t, time step
Input: $u \in (0, 1)$, desired confidence level
Input: $v \in (0, 1)$, desired tolerance

procedure $\text{ACCostEvalCI}(\mathfrak{S}, \Sigma, \mathbf{C}, L, I, \gamma_0, T_f, t, u, v)$
  $n \leftarrow \emptyset$
  while $t_{|n|-1,1-u/2}\sqrt{\frac{S^2(n)}{|n|}} > v \cdot \bar{X}(n)$ **do**
    $n \leftarrow n \cup \text{ACCostEvalSim}(\mathfrak{S}, \Sigma, \mathbf{C}, L, I, \gamma_0, T_f, t)$

In the interest of the *Accuracy* requirement, we consider a second approach, which allows the analyst to achieve an intended confidence in the cost value generated for a particular start state. With this approach, we decide the number of simulation runs to conduct based on a desired confidence and the assumption of a normal distribution of costs across runs. We use the fixed-sample-size procedure for point estimate of a mean, which says that the confidence interval for a mean is:

$$\bar{X}(n) \pm t_{|n|-1,1-\frac{\alpha}{2}}\sqrt{\frac{S^2(n)}{|n|}}$$

where $\bar{X}(n)$ is the sample mean, $\frac{S^2(n)}{|n|}$ is the sample variance, and $t_{\nu,\gamma}$ is the critical point for the t-distribution with ν degrees of freedom. The resulting range is an approximate $100(1-\alpha)$-percent confidence interval for the expected average cost of the scheme. During simulation, we repeatedly calculate the confidence interval for incrementing n, terminating when a satisfactory confidence is reached. For example, assuming we desire a 90-percent confidence interval of no more than 0.1 of the mean, we run the simulation repeatedly until:

$$t_{|n|-1,0.95}\sqrt{\frac{S^2(n)}{|n|}} \leq 0.1\bar{X}(n)$$

Algorithm 3 demonstrates the use of this approach to execute Algorithm 1 until a desired confidence is reached, rather than executing for a fixed number of runs.

We note that our cost analysis procedure evaluates particular implementations of the workload within candidate schemes, and thus cannot make formal claims about schemes in general. However, in practice, an analyst will be concerned primarily with the costs associated with the specific implementation she has designed; the existence of more efficient, though unknown, implementations is not particularly helpful in choosing an access control scheme. Finding optimal implementations is an orthogonal problem that we discuss in Section VIII.

VI. Access Control Extensions

In the event that a scheme does not admit a safe implementation of the workload, the analyst may attempt to enable the construction of such an implementation by augmenting the scheme with additional functionality.[4] Intuitively, extending an access control scheme expands its protected state, commands, and/or queries. One must use care, however, when constructing such extensions. Although virtually any changes to an access control scheme will yield another valid scheme, not all changes will yield a scheme that preserves the security properties of the original. As an extreme example, almost any scheme will be "broken" if we add a `grant-all` command that grants all permissions to all subjects (similar to McLean's System Z [23]).

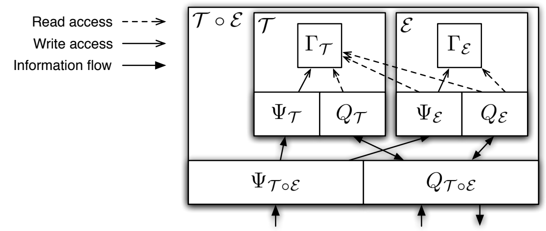

Fig. 4: A graphical representation of an access control scheme $\mathcal{T}$ augmented with an auxiliary machine $\mathcal{E}$

To maintain the intuition behind the concept of an extension, we require that the changes made to the scheme at most enable *additional* implementations (i.e., do not preclude the use of any implementations possible in the original). Specifically, in order to *safely* extend a scheme, one must prove that the extended scheme does not violate any of the security properties of the original. One can prove safety by viewing the original scheme as a workload operational description, and demonstrating a state-matching implementation of the original scheme within the extended scheme. This proves that the extended scheme can be used transparently in place of the original, and is therefore a safe extension. The violation of even simple safety resulting from extending a scheme with the above `grant-all` command can be detected by attempting (and failing) to construct such an implementation of the original scheme within this extended version while preserving simple safety.

In this paper, we explore a particular class of extensions that we call *auxiliary machines* (AMs).

Definition 15 (Auxiliary Machine) An *access control auxiliary machine* for augmenting an access control scheme over the set of access control states Γ_0 is a state-transition system $\langle \Gamma, \Psi, Q \rangle$, where:

- Γ is the set of auxiliary states.
- Ψ is the set of commands over $\Gamma_0 \times \Gamma$ where we enforce that $\forall \langle n, P, e \rangle \in \Psi, p \in P^*, \gamma_0 \in \Gamma_0, \gamma \in \Gamma, \exists \gamma' \in \Gamma : e(\langle \gamma_0, \gamma \rangle, p) = \langle \gamma_0, \gamma' \rangle$ (i.e., commands can reference the original scheme's state, but cannot alter it).
- Q is the set of queries over $\Gamma_0 \times \Gamma$ (i.e., that can reference the original scheme's state). $\Diamond$

Augmenting an access control scheme with an auxiliary machine is achieved by computing the cross product of the

[4]Note that it is not always possible to extend a scheme in a way that enables a particular implementation.

states of the two machines and the union of the commands and queries, as follows, and is represented graphically in Fig. 4.

Definition 16 (Augmented Scheme) Let $\mathcal{S} = \langle \Gamma^{\mathcal{S}}, \Psi^{\mathcal{S}}, Q^{\mathcal{S}} \rangle$ be an access control scheme, $\mathcal{U} = \langle \Gamma^{\mathcal{U}}, \Psi^{\mathcal{U}}, Q^{\mathcal{U}} \rangle$ be an access control auxiliary machine. The *augmented access control scheme* formed by augmenting scheme $\mathcal{S}$ with AM $\mathcal{U}$, is the scheme $\mathcal{S} \circ \mathcal{U} = \langle \Gamma^{\mathcal{S}\circ\mathcal{U}}, \Psi^{\mathcal{S}\circ\mathcal{U}}, Q^{\mathcal{S}\circ\mathcal{U}} \rangle$ where:

- $\Gamma^{\mathcal{S}\circ\mathcal{U}} = \Gamma^{\mathcal{S}} \times \Gamma^{\mathcal{U}}$
- $\Psi^{\mathcal{S}\circ\mathcal{U}} = \Psi^{\mathcal{S}} \cup \Psi^{\mathcal{U}}$
- $Q^{\mathcal{S}\circ\mathcal{U}} = Q^{\mathcal{S}} \cup Q^{\mathcal{U}}$ ◊

Definitions 15 and 16 give us the following theorem, which proves that the class of extensions that can be represented as auxiliary machines encode safe extensions to *any* access control scheme with respect to the state-matching implementation.

Theorem 4 *Given access control scheme $\mathcal{S} = \langle \Gamma^{\mathcal{S}}, \Psi^{\mathcal{S}}, Q^{\mathcal{S}} \rangle$ and access control auxiliary machine $\mathcal{U} = \langle \Gamma^{\mathcal{U}}, \Psi^{\mathcal{U}}, Q^{\mathcal{U}} \rangle$, there exists a state-matching implementation of $\mathcal{S}$ in $\mathcal{S} \circ \mathcal{U}$.*

PROOF (SKETCH) Intuitively, a scheme extended with an auxiliary machine can behave exactly as it would without the AM—it must answer the original queries in the same way as the original scheme, and it is forbidden from modifying elements of the original scheme's state in ways the original could not. Thus, to satisfy property (1) of the state-matching implementation, we map each state and action in the original to the same state or action in the augmented scheme, and the AM state is not utilized.

When considering only the original queries, the unmodified scheme can also easily mimic the augmented scheme, since these queries are guaranteed only to reference state that both schemes change in the same way (via the original commands). This satisfies property (2) of the state-matching implementation. □

While these security properties of auxiliary machines and augmented schemes enable the analyst to use the constructs without fear of contaminating the original schemes, they do not imply that the use of AMs (or extensions in general) is without penalty. Since AMs would be implemented as additional trusted code that communicates in a secure way with the original access control software, one may be concerned if a high proportion of the total state is stored within the AM, or if a large amount of communication needs to occur between the original state and the AM state. These types of concerns can be addressed by choosing appropriate cost measures for cost analysis.

Having presented a notion of scheme extensions and proven that it is safe, we now revisit the implementation of ADAC (Example 4) using DAC (Example 3).

Example 6 Recall that the workload $W_{\mathcal{A}}$ (Example 4) differs from the DAC scheme $\mathcal{D}$ (Example 3) mainly in that $W_{\mathcal{A}}$ has administrators with full rights to the system. In particular, the query `SubjectAdmin` is problematic, as the DAC scheme $\mathcal{D}$ has no way of maintaining the list of administrative users. One natural attempt at fixing this problem is to create a special object within $\mathcal{D}$, rights over which indicate administrator status. Another possibility is to create a special right that administrators have over all objects. Such approaches fail to allow a safe implementation, because they invalidate security analysis instances. In particular, these approaches alter the value of query `Access` for certain parameterizations.

Instead, we construct an auxiliary machine that stores the additional information and answers the additional query regarding administrative status of subjects. We extend DAC with auxiliary machine $\mathcal{M} = \langle \Gamma^{\mathcal{M}}, \Psi^{\mathcal{M}}, Q^{\mathcal{M}} \rangle$. The AM's states, $\Gamma^{\mathcal{M}}$, are defined by the sets $\langle A, N \rangle$, where:

- $A \subseteq S$ is the set of administrators
- $N : A \times O \to 2^R$ is the "hidden" access matrix that keeps track of the access rights each administrative subject would revert to upon losing administrator status

The extension's commands, $\Psi^{\mathcal{M}}$, include the following.

- `GrantAdmin(`S, S`)`, which grants administrative privilege to a subject
- `RevokeAdmin(`S, S`)`, which revokes administrative privilege from a subject
- `SoftGrant(`S, S, O, R`)`, which grants a right over an object to a subject in the hidden access matrix
- `SoftRevoke(`S, S, O, R`)`, which revokes a right over an object from a subject in the hidden access matrix

Finally, $Q^{\mathcal{M}}$ includes the following.

- `SubjectAdmin(`S, S`)`, which asks whether a subject is an administrator
- `HiddenAccess(`S, S, O, R`)`, which asks whether a user has a right over an object in the hidden access matrix ◊

Example 7 The AM described in Example 6 can augment the DAC scheme with the ability to keep track of which subjects are administrators, as well as which rights each would have if they lost such status. The implementation of $W_{\mathcal{A}}$ using DAC $\circ\, \mathcal{M}$ then has several non-trivial tasks. When a subject is added to A, the system must copy all current access for that subject from M to N and then grant that subject all accesses in M. This procedure is reversed when removing a user from A, and any rights granted to or revoked from a user in A are recorded in N and do not affect M. ◊

VII. Case Study

In this section, we discuss an example scenario which we will use to demonstrate a full analysis using our framework. This case study explores a workload based on a group messaging scenario with conflicts of interest.

A. Workload description

Group-centric Secure Information Sharing (g-SIS) [10] has been proposed as a new approach to access control that differs from the dissemination-centric approach that has inspired the development of schemes such as RBAC and DAC. Dissemination-centric models focus on bestowing policies upon objects as they are produced, sometimes refining these policies at later times. These policies are then referenced as consumers access these objects. The g-SIS approach, on the other hand, addresses collaboration- and subscription-based systems. In the g-SIS models, groups can be brought together to share information as they work toward a common goal. Accesses are decided not by attaching policies to objects, but in a time-variant way by inspecting the users' historical membership in groups. For example, an online periodical may offer a base subscription in which users have access to issues published during their subscription, and only while they remain subscribed. They might also offer (for an additional fee) current subscribers access to back issues, or former subscribers the ability to access issues published during their subscription.

The current state-of-the-art in implementations based on g-SIS is a formal specification in linear temporal logic for formal analysis [12]. The creators of g-SIS speculate that, with respect to expressiveness, these models may be equivalent to more traditional, dissemination-centric sharing models. However, they believe that the g-SIS approach will better enable the type of information sharing common in collaborative settings. Schemes inspired by g-SIS, then, would aim to provide an application-specific solution to access control. These schemes would aim to satisfy a category of applications that current models fail to capture, despite (possibly) possessing the expressive power necessary to express the applications' policies. Our framework is designed to investigate and quantify exactly this type of scenario, and as such this problem is a natural application of our framework. Thus, to verify and quantify the claims about g-SIS, we have modeled a group messaging workload after a particular use case within g-SIS, and evaluated within this workload the expressiveness and costs of common dissemination-centric schemes as well as a particular instantiation of the g-SIS approach within trust management.

B. *Our g-SIS Workload*

In our group messaging scenario, the main objects of interest are *messages* posted to *groups*. Current members of a group have access to the messages posted to it. When joining a group, a user can choose to request a *strict join* (in which access to previously posted messages is not granted) or a *liberal join* (in which access to all previous messages is granted). A similar decision is made when leaving a group. In the spirit of discussions such as those that take place in program committee meetings, we model workflows that accommodate users who must temporarily take leave from a group due to conflicts of interest, appointing temporary group administrators (if necessary) during this time.

The group messaging workload, $\mathcal{W}^{\mathcal{G}} = \langle \mathcal{G}, I^{\mathcal{G}} \rangle$, utilizes as its operational component the abstract group messaging scheme (GMS), $\mathcal{G}$. GMS is defined as $\mathcal{G} = \langle \Gamma^{\mathcal{G}}, \Psi^{\mathcal{G}}, Q^{\mathcal{G}} \rangle$. Its states, $\Gamma^{\mathcal{G}}$, are defined by the sets $\langle U, G, M, T, T_c, O, A, R, TX \rangle$, where:

- U is the set of users
- G is the set of groups
- M is the set of messages
- T is the ordered set of timestamps, including special timestamp ∞
- T_c is the current timestamp

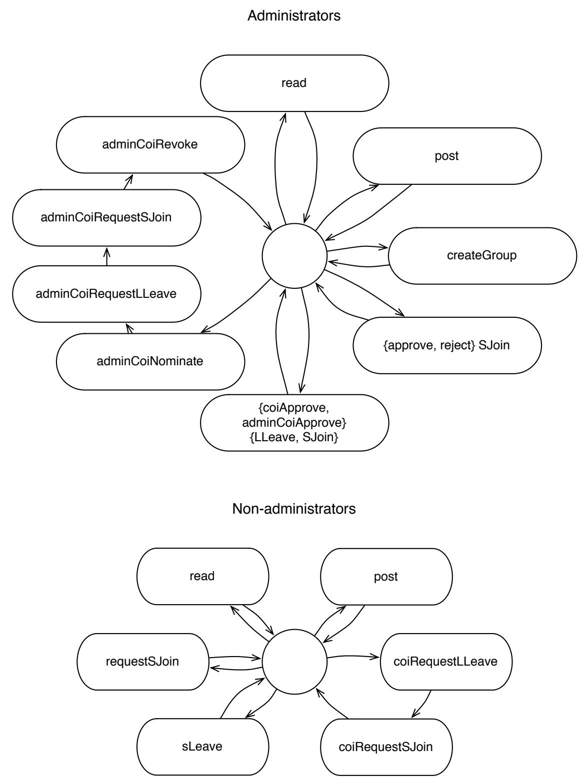

Fig. 5: Actor machines for the group messaging workload

- $O \subseteq U \times G$ is the group ownership relation
- $A \subseteq U \times G$ is the group administration relation
- $R \subseteq U \times G \times T \times T$ is the group membership record
- $TX \subseteq G \times M \times T$ is the messaging transcript

GMS's commands, $\Psi^{\mathcal{G}}$, include the following.

- `CreateGroup(`U`, `G`)`, which creates a group
- `GrantAdmin(`U`, `U`, `G`)`, which grants a user administrative permission for a group
- `RevokeAdmin(`U`, `U`, `G`)`, which revokes from a user administrative permission for a group
- `SAddMember(`U`, `U`, `G`)`, which strict-adds a user to a group (i.e., adds the user without granting permission to view existing messages)
- `LAddMember(`U`, `U`, `G`)`, which liberal-adds a user to a group (i.e., adds the user and grants permission to view existing messages)
- `SRemoveMember(`U`, `U`, `G`)`, which strict-removes a user from a group (i.e., removes the user and revokes permission to view currently existing messages)
- `LRemoveMember(`U`, `U`, `G`)`, which liberal-removes a user from a group (i.e., removes the user without revoking permission to view currently existing messages)
- `Post(`U`, `G`, `M`)`, which posts a message to a group

Finally, GMS's queries, $Q^{\mathcal{G}}$, include the following.

- `Access(`U`, `M`)`, which asks whether a user can view a message

We fully define GMS in Appendix B-A. The invocation mechanism for the group messaging workload, $I^{\mathcal{G}}$, is described

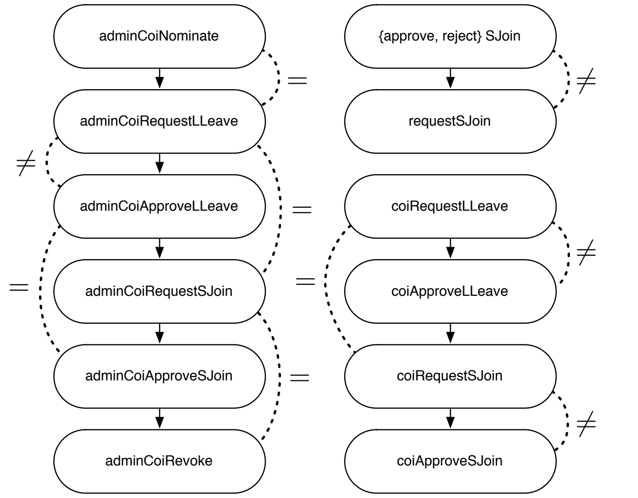

Fig. 6: Constrained workflow for the group messaging workload

by the actor graphs depicted in Fig. 5[5] and the constrained workflow depicted in Fig. 6.

C. Expressiveness Evaluation

In the first phase of suitability analysis, we examine the expressive power of our candidate schemes to determine which are capable of safely implementing the workload. In Appendix B, we formally describe these implementations and prove that they are state-matching implementations. In this section, we omit these details in favor of an intuitive discussion. In particular, we explored the use of the following access control schemes to implement the GMS workload:

- *SD3-GM* is a specially-parameterized instantiation of the trust management language SD3 [24]. Given the flexibility offered by a logical policy language, SD3-GM easily implements the group messaging workload.
- *DAC* is a discretionary access control scheme based on the Graham-Denning scheme [25]. DAC does not admit an obvious state-matching implementation. Thus, we extended DAC with an auxiliary machine to manage the group-based metadata (e.g., the group membership relation). DAC's access matrix is updated after changes are made to the AM data, allowing the `Access` query to be answered as in the original DAC scheme.
- *RBAC* is a role-based access control scheme based on NIST RBAC [26]. While SD3-GM is a near perfect fit for the workload, and DAC is reduced to having its native internal state used only as a projection of an auxiliary machine, RBAC's role relation can be used to maintain more relevant state natively. We still utilize an AM for RBAC, mainly to maintain the message-group relation which cannot be maintained in any obvious way within the RBAC state.
- *GTRBAC* (Generalized Temporal RBAC) is an extended version of RBAC that adds temporal features such as the time-constrained activation of roles [27]. However, it does not include the ability to make access decisions based on the time at which a user joined a role or the time an object was created. Thus, the features of GTRBAC beyond those of RBAC do not contribute to a more efficient implementation of GMS, and we thus dropped GTRBAC from consideration.

[5]We omit the rates in Fig. 5, as these are varied during our cost analysis.

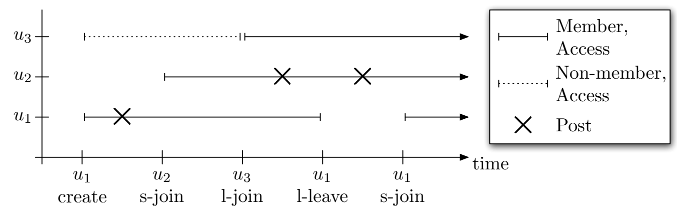

Fig. 7: An example scenario of accesses in a single group in the group-messaging scenario

At first blush, it may seem counter-intuitive that DAC and RBAC require extensions to correctly support the GMS workload. However, as demonstrated by Fig. 7, the group messaging scenario can be unexpectedly difficult to represent in dissemination-centric models. Although a group may seem to conceptually resemble a role in role-based access control, roles grant the same accesses to all members, while Fig. 7 shows that even a simple series of events within a single group containing a small number of users can lead to multiple disjoint sets of accesses in GMS. In this particular example, all three users have a different "view" of the objects in the group, despite all being members. This single-group scenario is impossible to represent in a role-based scheme with fewer than three roles, indicating that any implementation of GMS in a role-based scheme is very likely to exceed a role per user, reducing the administrative value of utilizing roles at all [28], [29].

The following theorem asserts that each of our three remaining candidate schemes satisfies our requirements for a safe implementation of the group messaging workload. This theorem is proved (individually for each scheme) in Appendix B.

Theorem 5 *There exists a state-matching implementation of GMS in SD3-GM, and in each of our extended versions of RBAC and DAC.*

D. Cost Analysis

To perform cost analysis for the group messaging case study described above, we consider cost measures representing communication with the auxiliary machine (where applicable) and maximum state size. We then defined cost functions over these cost measures for RBAC, DAC, and SD3-GM. We used these cost functions as inputs to an implementation we built of (extended versions of) Algorithms 1 and 2. The implementation of our simulator consists of about 2000 lines of Java code. We take the Monte Carlo approach in order to gain insight into the trends in the implementations' costs across the variety of start states, altering the number of users, number of administrators, global rate of conflict-of-interest scenarios, and global rate of

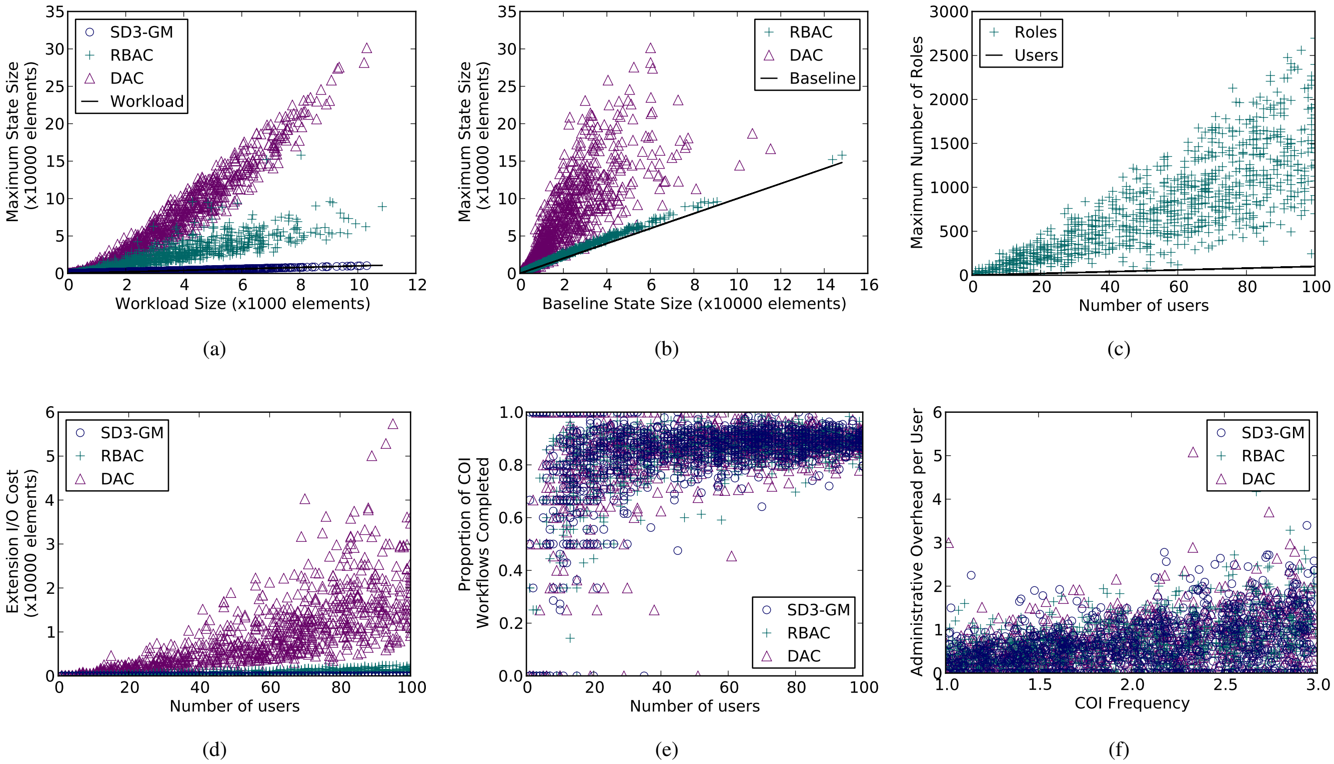

Fig. 8: Group messaging case study results

message posting. We simulated the messaging environment for 8-hour periods of interleaved action traces as described by the group messaging workload's actor machines (Fig. 5) and constrained workflow (Fig. 6). We repeated this simulation for 1,000 Monte Carlo runs.

Figure 8 shows the results of the our evaluation of the implementations of GMS. In Fig. 8a, we compare maximum state size to the state size occupied by the equivalent GMS state, demonstrating the additional storage needed to utilize each candidate access control scheme. While SD3-GM utilized a small constant amount of additional storage, both RBAC and DAC required many times the storage of GMS.

In Fig. 8b, we look at maximum state size in a different way—compared to the "baseline state," which describes the amount of storage needed to use the scheme naively to reproduce the same accesses as the workload. For DAC, this is the access matrix, including the appropriate accesses. For RBAC, this includes the user-role and role-permission relations, assigning each user to her own role with the permissions the user has access to. Although the baseline state does not maintain enough information to enable a state-matching (i.e., safe) implementation, it allows a comparison to the storage of using the scheme naively. As Fig. 8b shows, storage in both RBAC and DAC exceeds this baseline, with DAC being particularly excessive.

In Fig. 8c, we compare the number of users in the system to the number of roles needed to represent the GMS state in RBAC. The administrative value of the RBAC model diminishes when the number of roles exceeds the number of users [28], [29], and thus we assume that this scenario is evidence of the RBAC system being used outside of the use cases it was designed for. Thus, Fig. 8c is particularly strong evidence of the ill-suitedness of RBAC to the group messaging workload, since systems with less than 100 users can exceed 2,000 roles, and on average there were over 14 times as many roles as users.

Finally, as another proxy for implementation complexity, Fig. 8d shows the amount of communication with an auxiliary machine that occurred during a run, compared with the number of users in the system. DAC's much larger extension cost was the result of this scheme having no appropriate state elements that could store most of the information needed in the group messaging workload, while RBAC performed better due to its ability to store group membership, ownership, and administration relations within its role relation.

In addition, we present in Fig. 8 several findings that, although they do not support the selection of one scheme over another, nonetheless provided insight into the group messaging workload. In Fig. 8e, we show the relationship between the number of users in the system and the proportion of attempted conflict-of-interest workflows that are successfully completed within the simulation run. We found that the main bottleneck for completing COIs was the number of users. Thus, runs with fewer users had both fewer COIs complete and a longer

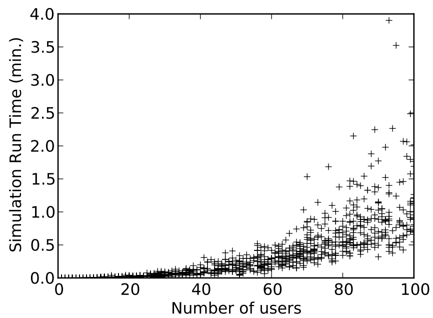

Fig. 9: Run time for simulating 8-hour periods in the group messaging workload

duration of time between the initialization and completion of those that did. In Fig. 8f, we present the results of our exploration of a related facet of this relationship, in which we found a clear positive trend between the frequency with which COIs are initiated and administrative work completed by non-administrative users (i.e., those that had been nominated to fulfill administrative duties temporarily).

E. Summary of Findings

The creators of g-SIS believed that dissemination-centric sharing models could represent group-centric sharing [10]. However, we found that, without extensions, commonly-used examples DAC and RBAC were not able to *safely* implement a workload inspired by a particular scenario for which g-SIS is well-suited. Although SD3 can also represent dissemination-centric sharing, we have used a particular parameterization of SD3 (which we call SD3-GM) to represent an instantiation of the g-SIS model and implement group messaging workload in an efficient way. Thus, it seems that at least some access control schemes are capable of performing well in both dissemination-centric and group-centric scenarios. By evaluating the relative suitability of dissemination-centric schemes (extended versions of DAC and RBAC) to the group messaging workload, we have confirmed the suspicions in [10] that, although these schemes can represent the workload, they cannot address it as naturally, and suffer from inefficiencies. This highlights the importance of conducting suitability analysis, especially for novel applications, and confirms that expressiveness alone is not enough to make decisions about access control schemes.

VIII. Discussion

In this section, we first revisit each of the requirement for suitability analysis frameworks outlined in Section III-B, and then discuss a number of areas of future work related to the suitability analysis problem.

A. Requirements Revisited

In Section III-B, we outlined six requirements to guide the development of our suitability analysis framework. We now discuss the degree to which each requirement was met.

- The *Domain Exploration* requirement is addressed equally by the workload formalism developed in Section IV-B and the Monte Carlo simulation procedure described in Section V-D: the former leaves the state defining the workload and the mechanisms that can alter it completely in the hands of the analyst, while the latter facilitates cost analysis over many such instances of the workload.
- *Cooperative Interaction* is met by combining the workflow and actor graph formalisms developed in Section V-A with the WSP solver leveraged by Algorithm 1 in Section V-D. Specifically, constrained workflows articulate the ways in which cooperation must be carried out, while the use of actor graphs and the WSP solver ensures that all traces generated during cost analysis are compliant with these workflows.
- With respect to safety, we focused in this paper on a particular notion of safe implementation—i.e., the state-matching implementation (cf. Section IV-C)—and its use in extending access control schemes and implementing workloads. However, the use of this particular notion of safe implementation is not required by our framework: proofs of safety are carried out manually, and thus any other notion of safe implementation could be used. As such, our framework provides *Tunable Safety*.
- In contrast to safety analysis, cost analysis is a largely automated procedure that is constrained by our framework. However, as was demonstrated in Sections V-B and VII, the notion of cost measure developed in this paper is capable of representing a wide range of system- and human-centric costs. Further, Proposition 2 shows that any vector of measures is itself a cost measure, so many costs can be considered in parallel. As such, our framework meets the *Tunable Costs* requirement.
- Supporting multi-user workflows is seemingly at odds with the *Tractability* requirement, as the workflow satisfiability problem has been shown to be NP-complete [17]. However, the proof of Theorem 3 makes use of recent results [17], [18] to show that our Monte Carlo analysis process (via Algorithms 1 and 2) is fixed-parameter tractable if the length of workflows within the system is treated as a small constant, as is typically the case in practice. In addition, Fig. 9 shows the time required for 8-hour simulation runs using our Java-based simulator on a 3.06 GHz Core 2 Duo with respect to the number of users in the system (the most significant variable in the run time). This figure shows that even with many users, simulating 8-hour runs takes less than four minutes on commodity hardware. In addition, since we utilize a Monte Carlo approach, the multiple simulation runs are inherently parallelizable.
- In terms of the *Accuracy* requirement, Section V-D discusses how to calculate confidence intervals for point estimates of cost. Further, Algorithms 1 and 3 demonstrate how the cost analysis process can be guided by a desired confidence interval for specific configurations of interest within the workload's parameter space.

The analysis framework developed in this paper meets each of the desiderata outlined in Section III-B, and provides a flexible, efficient, and precise mechanism for analyzing instances of the access control suitability analysis problem.

B. Open Problems and Future Work

We now discuss the future of application-aware suitability analysis, including refinements to our existing framework and ways in which this approach can be extended to the formalization of more general security workloads.

Implementation Non-Existence: A proof that a particular implementation does not exist is typically harder to produce than a constructive existence proof. Thus, in our work so far, when discussing a lack of an implementation, we often resort to informal arguments for justification. Ideally, it would be possible to more easily prove the non-existence of of an implementation, since such proofs give higher confidence in the necessity of extending access control schemes.

Implementation Optimality: The constructive nature of an implementation of a workload in a scheme leads quite naturally to the cost analysis of this scheme, as workload actions can be translated into scheme actions by the implementation. Given an access control scheme $\mathcal{S}$ and a workload W, we therefore carry out the cost analysis of a *particular* implementation of W in $\mathcal{S}$, rather than the *best* implementation of W in $\mathcal{S}$. It would be useful to develop techniques for proving the optimality of an implementation. This would enable analysts to make strong claims about the (sub-)optimality of an access control scheme for a given workload without needing to justify or defend the implementations used during their analysis.

Alternate Notions of Implementation: Recall that we consider a type of safe implementation based on the state-matching reduction, the strongest type of mapping studied in previous work [5]. However, other notions of implementation certainly exist in the literature (e.g., see [3], [4], [6]–[8]), and are likely applicable within certain classes of workloads. Understanding the benefits and limitations of using relaxed notions of implementation is an important area of future work. It is also important to explore relationships (e.g., implication, equivalence) between known access control implementations, as well as between implementations and mappings from other domains. For example, the state-matching reduction shares certain structural properties with weak simulations in model checking. Alternate formalizations of the access control problem could enable the application of analysis techniques from other domains toward access control.

Quantifying Human Costs: Although the cost measures and cost functions formalized in this paper are capable of representing a wide-range of interesting costs, capturing human-centric costs—such as, e.g., cognitive overheads for various tasks, or error rates in policy formulation—is a difficult task. Our focus in this paper lies in the utilization of these types of costs measures, rather than in their capture. However, we are inspired by recent work within the usable security community on measuring exactly these types of phenomena (e.g., [30]–[34]). These types of studies provide a roadmap for suitability analysts that wish to incorporate human costs into their analyses, and signal a shift in security analysis: quantitative analysis of these systems cannot be done in a strictly pencil-and-paper fashion, but must also include studies of the humans who manipulate and administer these systems.

Beyond Access Control: This paper focuses on one particular instance of the suitability analysis problem that is specific to access control schemes. However, we believe that suitability analysis can be cast in a more general manner and applied to broader security workloads, as solutions to many security problems need to balance formal requirements to be upheld by a system with the real-world impacts and costs of these solutions. As an example, consider the public key infrastructure (PKI) upon which many web authentication and authorization frameworks are built. Recently, there have been high profile compromises of CAs in the web PKI domain (e.g., [35], [36]). These failures have made clear the fragility of the trust model and revocation mechanisms in the web space, and have inspired the community to examine methods both for reinforcing the system's mechanisms to prevent fraudulent certificate issuances and improving the robustness of the revocation infrastructure (e.g., Perspectives [37], Sovereign Keys [38], CA Transparency [39], etc.). However, there is considerable debate in the community regarding what the appropriate metrics for judging replacement systems should be, and how the different proposals compare under realistic conditions. A more general formulation of the suitability analysis problem could enable better understanding of the trade-offs between the formal guarantees and the real-world costs incurred by such candidate infrastructures.

IX. Conclusion

Historically, most work regarding the formal analysis of access control schemes has focused on evaluating expressive power in absolute terms. By contrast, our goal in this paper was to formalize the *suitability analysis problem* and to develop a methodology for application-specific evaluation of access control schemes. To this end, we have developed a formal framework for specifying access control workloads, reasoning about the abilities of candidate access control schemes to safely service these workloads, safely augmenting schemes that are incapable of implementing a given workload, and carrying out cost-based analysis of the suitability of each candidate scheme for servicing the workload. Formal proofs demonstrate the soundness of our approach, and a detailed case study drawn from the literature illustrates the applicability of our framework for conducting real world suitability analyses. The framework that we have developed is a first step toward understanding the application-specific strengths of access control systems. However, the basic techniques used in this framework appear to be applicable to broader security problems, in which several systems may be capable of meeting a set of security goals, but the costs of using each candidate system vary.

Acknowledgements This work was supported in part by the National Science Foundation under awards CNS-0964295, CNS-1228697, and CNS-1228947.

References

[1] M. A. Harrison, W. L. Ruzzo, and J. D. Ullman, "Protection in operating systems," *Communications of the ACM*, vol. 19, no. 8, pp. 461–471, Aug 1976.

[2] R. J. Lipton and L. Snyder, "A linear time algorithm for deciding subject security," *J. ACM*, vol. 24, no. 3, pp. 455–464, 1977.

[3] P. Ammann, R. J. Lipton, and R. S. Sandhu, "The expressive power of multi-parent creation in monotonic access control models," *Journal of Computer Security*, vol. 4, no. 2/3, pp. 149–166, 1996.

[4] A. Chander, J. C. Mitchell, and D. Dean, "A state-transition model of trust management and access control," in *14th IEEE Computer Security Foundations Workshop (CSFW'01)*, 2001, pp. 27–43.

[5] M. V. Tripunitara and N. Li, "A theory for comparing the expressive power of access control models," *Journal of Computer Security*, vol. 15, no. 2, pp. 231–272, 2007.

[6] S. Osborne, R. Sandhu, and Q. Munawer, "Configuring role-based access control to enforce mandatory and discretionary access control policies," *ACM Transactions on Information and System Security*, vol. 3, no. 2, pp. 85–106, May 2000.

[7] R. Sandhu, "Expressive power of the schematic protection model," *Journal of Computer Security*, vol. 1, no. 1, pp. 59–98, 1992.

[8] R. S. Sandhu and S. Ganta, "On testing for absence of rights in access control models," in *IEEE Computer Security Foundations Workshop (CSFW)*, 1993, pp. 109–118.

[9] V. C. Hu, D. F. Ferraiolo, and D. R. Kuhn, *Assessment of Access Control Systems*. National Institute of Standards and Technology, 2006.

[10] R. Krishnan, R. Sandhu, J. Niu, and W. H. Winsborough, "Foundations for group-centric secure information sharing models," in *ACM Symposium on Access Control Models and Technologies (SACMAT)*, 2009, pp. 115–124.

[11] R. Krishnan, R. Sandhu, and W. H. Winsborough, "A Conceptual Framework for Group-Centric Secure Information Sharing Categories and Subject Descriptors," in *ACM Symposium on Information, Computer, and Communications Security (ASIACCS)*, 2009.

[12] R. Krishnan, J. Niu, R. S. Sandhu, and W. H. Winsborough, "Group-centric secure information-sharing models for isolated groups," *ACM Trans. Inf. Syst. Secur.*, vol. 14, no. 3, p. 23, 2011.

[13] N. Li, J. C. Mitchell, and W. H. Winsborough, "Beyond proof-of-compliance: security analysis in trust management," *Journal of the ACM*, vol. 52, no. 3, pp. 474–514, May 2005.

[14] Q. Wang, N. Li, and H. Chen, "On the security of delegation in access control systems," in *European Symposium on Research in Computer Security (ESORICS)*, Oct 2008, pp. 317–332.

[15] G. R. Ganger, "Generating representative synthetic workloads: An unsolved problem," in *International CMG Conference*, Dec 1995, pp. 1263–1269.

[16] K. S. Anderson, J. P. Bigus, E. Bouillet, P. Dube, N. Halim, Z. Liu, and D. E. Pendarakis, "Sword: scalable and flexible workload generator for distributed data processing systems," in *Winter Simulation Conference (WSC)*, Dec 2006, pp. 2109–2116.

[17] Q. Wang and N. Li, "Satisfiability and Resiliency in Workflow Systems," in *European Symposium on Research in Computer Security (ESORICS)*, 2007.

[18] J. Crampton, G. Gutin, and A. Yeo, "On the Parameterized Complexity of the Workflow Satisfiability Problem," in *ACM Conference on Computer and Communications Security (CCS)*, 2012, pp. 857–868.

[19] U.S. Air Force Scientific Advisory Board, "Networking to enable coalition operations," MITRE Corporation, Tech. Rep., 2004.

[20] "Horizontal integration: Broader access models for realizing information dominance," MITRE Corporation JASON Program Office, Tech. Rep. JSR-04-13, 2004.

[21] T. L. Hinrichs, W. C. Garrison III, A. J. Lee, and J. C. Mitchell, "Application-sensitive access control evaluation: Logical foundations (extended version)," University of Pittsburgh Department of Computer Science, Tech. Rep. TR-12-185, Nov 2011. [Online]. Available: http://www.cs.pitt.edu/~bill/pubs/T12a.pdf

[22] T. M. Liggett, *Continuous Time Markov Processes: An Introduction*, ser. Graduate Studies in Mathematics Series. American Mathematical Society, 2010.

[23] J. McLean, "Reasoning about security models," in *IEEE Symposium on Security and Privacy*, Apr. 1987, pp. 123–133.

[24] T. Jim, "SD3: A trust management system with certified evaluation," in *IEEE Symposium on Security and Privacy*, 2001, pp. 106–115.

[25] G. S. Graham and P. J. Denning, "Protection: principles and practice," in *Proceedings of the May 16-18, 1972, spring joint computer conference*, ser. AFIPS '72 (Spring). New York, NY, USA: ACM, 1972, pp. 417–429.

[26] D. F. Ferraiolo, R. S. Sandhu, S. I. Gavrila, D. R. Kuhn, and R. Chandramouli, "Proposed nist standard for role-based access control," *ACM Trans. Inf. Syst. Secur.*, vol. 4, no. 3, pp. 224–274, 2001.

[27] J. Joshi, E. Bertino, U. Latif, and A. Ghafoor, "A generalized temporal role-based access control model," *IEEE Transactions on Knowledge and Data Engineering*, vol. 17, no. 1, pp. 4–23, 2005.

[28] S. Lu, Y. Hong, Q. Liu, L. Wang, and R. Dssouli, "Access control in e-health portal systems," in *4th International Conference on Innovations in Information Technology*, Nov 2007, pp. 88–92.

[29] D. Zhang, K. Ramamohanarao, S. Versteeg, and R. Zhang, "RoleVAT: Visual assessment of practical need for role based access control," in *Twenty-Fifth Annual Computer Security Applications Conference (ACSAC 2009)*, Dec 2009, pp. 13–22.

[30] R. W. Reeder, L. Bauer, L. F. Cranor, M. K. Reiter, and K. Vaniea, "More than skin deep: measuring effects of the underlying model on access-control system usability," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI)*, 2011, pp. 2065–2074.

[31] L. Bauer, L. F. Cranor, R. W. Reeder, M. K. Reiter, and K. Vaniea, "A user study of policy creation in a flexible access-control system," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI)*, 2008, pp. 543–552.

[32] R. W. Reeder, L. Bauer, L. F. Cranor, M. K. Reiter, K. Bacon, K. How, and H. Strong, "Expandable grids for visualizing and authoring computer security policies," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI)*, 2008, pp. 1473–1482.

[33] J. Clark, P. C. van Oorschot, and C. Adams, "Usability of anonymous web browsing: an examination of tor interfaces and deployability," in *Proceedings of the 3rd Symposium on Usable Privacy and Security (SOUPS)*, 2007, pp. 41–51.

[34] A. Mazzia, K. LeFevre, and E. Adar, "The pviz comprehension tool for social network privacy settings," in *Proceedings of the Eighth Symposium on Usable Privacy and Security (SOUPS)*, 2012, pp. 13:1–13:12.

[35] BBC News, "Iranians hit in email hack attack," September 2011. [Online]. Available: http://www.bbc.co.uk/news/technology-14802673

[36] "Comodo Report of Incident - Comodo detected and thwarted an intrusion on 26-MAR-2011," March 2011. [Online]. Available: https://www.comodo.com/Comodo-Fraud-Incident-2011-03-23.html

[37] D. Wendlandt, D. G. Andersen, and A. Perrig, "Perspectives: improving ssh-style host authentication with multi-path probing." USENIX 2008 Annual Technical Conference, June 2008.

[38] Electronic Frontier Foundation, "The sovereign keys project." [Online]. Available: https://www.eff.org/sovereign-keys

[39] B. Laurie and A. Langley, "Certificate authority transparency and auditability," November 2011. [Online]. Available: http://www.links.org/files/CertificateAuthorityTransparencyandAuditability.pdf

[40] J. Hromkovic, *Algorithmics for Hard Problems: Introduction to Combinatorial Optimization, Randomization, Approximation, and Heuristics*. Berlin, Heidelberg: Springer-Verlag, 2010.

[41] N. Li, J.-W. Byun, and E. Bertino, "A critique of the ansi standard on role-based access control," *IEEE Security & Privacy*, vol. 5, no. 6, pp. 41–49, 2007.

Appendix A
Proofs

A. Proof of Proposition 1

First, we restate our definition of access control scheme from Section IV-A. For the purposes of this proof, we refer to this notion of scheme as the GLH scheme.

Definition 3 A *GLH scheme* is a state transition system $\mathcal{S} = \langle \Gamma, \Psi, Q \rangle$, where Γ is the set of access control states, Ψ is the set of commands over Γ, and Q is the set of queries over Γ.$\diamondsuit$

Next, we state the definition of scheme used by Tripunitara and Li [5], which we refer to in this proof as the TL scheme.

Definition 17 A *TL scheme* is a state-transition system $\mathcal{S} = \langle \Gamma, Q, \vdash, \Psi \rangle$, in which Γ is a set of states, Q is a set of queries, $\vdash : \Gamma \times Q \to \{\text{TRUE}, \text{FALSE}\}$ is called the entailment relation, and Ψ is a set of state-transition rules. $\Diamond$

Recall that GLH schemes formalize transitions and state inspection using commands and queries that accept parameters. For transitions, a TL scheme specifies a set of state transition rules, each a binary relation on the set of states. A running system, then, must specify which transition rule is active. For queries, a TL scheme specifies a set of (non-parameterized) queries, and the entailment relation (rather than being individually specified as a component of the query structure) is specified for all queries as a separate component of the scheme.

A main result of Tripunitara and Li's framework closely mirrors Proposition 1, except for TL schemes and the state-matching reduction rather than GLH schemes and the state-matching implementation. We now present the definition of state-matching reduction and the related theorem to Proposition 1.

Definition 18 (State-Matching Reduction [5]) Given two access control schemes $\mathcal{A} = \langle \Gamma^{\mathcal{A}}, \Psi^{\mathcal{A}}, Q^{\mathcal{A}}, \vdash^{\mathcal{A}} \rangle$ and $\mathcal{B} = \langle \Gamma^{\mathcal{B}}, \Psi^{\mathcal{B}}, Q^{\mathcal{B}}, \vdash^{\mathcal{B}} \rangle$, and a mapping from $\mathcal{A}$ to $\mathcal{B}$, $\sigma : (\Gamma^{\mathcal{A}} \times \Psi^{\mathcal{A}}) \cup Q^{\mathcal{A}} \to (\Gamma^{\mathcal{B}} \times \Psi^{\mathcal{B}}) \cup Q^{\mathcal{B}}$, we say that two states $\gamma^{\mathcal{A}}$ and $\gamma^{\mathcal{B}}$ are *equivalent* under the mapping σ when for every $q^{\mathcal{A}} \in Q^{\mathcal{A}}$, $\gamma^{\mathcal{A}} \vdash^{\mathcal{A}} q^{\mathcal{A}}$ if and only if $\gamma^{\mathcal{B}} \vdash^{\mathcal{B}} \sigma(q^{\mathcal{A}})$. A mapping σ from $\mathcal{A}$ to $\mathcal{B}$ is said to be a *state-matching reduction* if for every $\gamma^{\mathcal{A}} \in \Gamma^{\mathcal{A}}$ and every $\psi^{\mathcal{A}} \in \Psi^{\mathcal{A}}$, $\langle \gamma^{\mathcal{B}}, \psi^{\mathcal{B}} \rangle = \sigma(\langle \gamma^{\mathcal{A}}, \psi^{\mathcal{A}} \rangle)$ has the following two properties:

1) For every state $\gamma_1^{\mathcal{A}}$ in scheme $\mathcal{A}$ such that $\gamma^{\mathcal{A}} \overset{*}{\mapsto}_{\psi^{\mathcal{A}}} \gamma_1^{\mathcal{A}}$, there exists a state $\gamma_1^{\mathcal{B}}$ such that $\gamma^{\mathcal{B}} \overset{*}{\mapsto}_{\psi^{\mathcal{B}}} \gamma_1^{\mathcal{B}}$ and $\gamma_1^{\mathcal{A}}$ and $\gamma_1^{\mathcal{B}}$ are equivalent under σ.
2) For every state $\gamma_1^{\mathcal{B}}$ in scheme $\mathcal{B}$ such that $\gamma^{\mathcal{B}} \overset{*}{\mapsto}_{\psi^{\mathcal{B}}} \gamma_1^{\mathcal{B}}$, there exists a state $\gamma_1^{\mathcal{A}}$ such that $\gamma^{\mathcal{A}} \overset{*}{\mapsto}_{\psi^{\mathcal{A}}} \gamma_1^{\mathcal{A}}$ and $\gamma_1^{\mathcal{A}}$ and $\gamma_1^{\mathcal{B}}$ are equivalent under σ. $\Diamond$

Theorem 6 (Rephrased, from [5]) *Given two schemes $\mathcal{A}$ and $\mathcal{B}$, and a mapping, σ, from $\mathcal{A}$ to $\mathcal{B}$, σ is a state-matching reduction if and only if it is strongly security-preserving; that is, every compositional security analysis instance in $\mathcal{A}$ is true if and only if the image of the instance under σ is true in $\mathcal{B}$.*

Next, we restate the definition of state-matching implementation from Section IV-C.

Definition 7 Given an access control workload $W = \langle \mathcal{W}, I^{\mathcal{W}} \rangle$ in which $\mathcal{W} = \langle \Gamma^{\mathcal{W}}, \Psi^{\mathcal{W}}, Q^{\mathcal{W}} \rangle$, an access control scheme, $\mathcal{S} = \langle \Gamma^{\mathcal{S}}, \Psi^{\mathcal{S}}, Q^{\mathcal{S}} \rangle$, and an implementation $\sigma = \langle \sigma_\Gamma, \sigma_\Psi, \sigma_Q \rangle$ of $\mathcal{W}$ in $\mathcal{S}$, we say that two states $\gamma^{\mathcal{W}}$ and $\sigma_\Gamma(\gamma^{\mathcal{W}}) = \gamma^{\mathcal{S}}$ are *equivalent* with respect to the implementation σ (and denote this equivalence as $\gamma^{\mathcal{W}} \sim_\sigma \gamma^{\mathcal{S}}$) when for every $q^{\mathcal{W}} = \langle n, P, \vdash \rangle \in Q^{\mathcal{W}}$ (with $q^{\mathcal{S}} = \sigma_Q(q^{\mathcal{W}})$) and every $p^{\mathcal{W}} \in P^*$ (with $p^{\mathcal{S}} = \sigma_\Gamma(p^{\mathcal{W}})$), $\gamma^{\mathcal{W}} \vdash q^{\mathcal{W}}(p^{\mathcal{W}})$ if and only if $\gamma^{\mathcal{S}} \vdash q^{\mathcal{S}}(p^{\mathcal{S}})$.

An implementation σ of $\mathcal{W}$ in $\mathcal{S}$ is said to be a *state-matching implementation* if for every $\gamma^{\mathcal{W}} \in \Gamma^{\mathcal{W}}$, with $\gamma^{\mathcal{S}} = \sigma_\Gamma(\gamma^{\mathcal{W}})$, the following two properties hold:

1) For every state $\gamma_1^{\mathcal{W}} \in \Gamma^{\mathcal{W}}$ such that $\gamma^{\mathcal{W}} \overset{*}{\mapsto}_{\Psi^{\mathcal{W}}} \gamma_1^{\mathcal{W}}$, there exists a state $\gamma_1^{\mathcal{S}} \in \Gamma^{\mathcal{S}}$ such that $\gamma^{\mathcal{S}} \overset{*}{\mapsto}_{\Psi^{\mathcal{S}}} \gamma_1^{\mathcal{S}}$ and $\gamma_1^{\mathcal{W}} \sim_\sigma \gamma_1^{\mathcal{S}}$.
2) For every state $\gamma_1^{\mathcal{S}} \in \Gamma^{\mathcal{S}}$ such that $\gamma^{\mathcal{S}} \overset{*}{\mapsto}_{\Psi^{\mathcal{S}}} \gamma_1^{\mathcal{S}}$, there exists a state $\gamma_1^{\mathcal{W}} \in \Gamma^{\mathcal{W}}$ such that $\gamma^{\mathcal{W}} \overset{*}{\mapsto}_{\Psi^{\mathcal{W}}} \gamma_1^{\mathcal{W}}$ and $\gamma_1^{\mathcal{W}} \sim_\sigma \gamma_1^{\mathcal{S}}$. $\Diamond$

Finally, we restate and prove Proposition 1.

Proposition 1 *Given an access control workload $W = \langle \mathcal{W}, I^{\mathcal{W}} \rangle$ in which $\mathcal{W} = \langle \Gamma^{\mathcal{W}}, \Psi^{\mathcal{W}}, Q^{\mathcal{W}} \rangle$, an access control scheme, $\mathcal{S} = \langle \Gamma^{\mathcal{S}}, \Psi^{\mathcal{S}}, Q^{\mathcal{S}} \rangle$, and an implementation $\sigma = \langle \sigma_\Gamma, \sigma_\Psi, \sigma_Q \rangle$ of $\mathcal{W}$ in $\mathcal{S}$, σ is a state-matching implementation if and only if it is strongly security-preserving; that is, every compositional security analysis instance in $\mathcal{W}$ is true if and only if the image of the instance under σ is true in $\mathcal{S}$.*

PROOF Consider workload operational component $\mathcal{W}$, scheme $\mathcal{S}$, and implementation $\sigma^{\mathcal{W}}$ of $\mathcal{W}$ using $\mathcal{S}$. We assume that $\sigma^{\mathcal{W}}$ is a state-matching implementation and show that it must be strongly security-preserving.

Construct TL scheme $\mathcal{A}$ from $\mathcal{W}$ (and, similarly, $\mathcal{B}$ from $\mathcal{S}$) as follows. Make $\Gamma^{\mathcal{A}}$ equal to $\Gamma^{\mathcal{W}}$ (preserve state information exactly). Map each query-parameter pair q, P in $\mathcal{W}$ to the single query q_P in $Q^{\mathcal{A}}$ in scheme $\mathcal{A}$. Encode these queries' individual entailment relations (over parameters) in scheme $\mathcal{A}$'s entailment relation (over queries). Collapse all commands in $\mathcal{W}$ into a binary relation over states. Encode this binary relation as a single state transition rule in $\Psi^{\mathcal{A}}$, where $\langle \gamma_1, \gamma_2 \rangle \in \psi^{\mathcal{A}}$ if and only if $\exists \psi \in \Psi^{\mathcal{W}}, P \in P^* : e(\gamma_1, P) = \gamma_2$.

Construct the reduction $\sigma^{\mathcal{A}} : (\Gamma^{\mathcal{A}} \times \Psi^{\mathcal{A}} \to \Gamma^{\mathcal{B}} \times \Psi^{\mathcal{B}}) \cup (Q^{\mathcal{A}} \to Q^{\mathcal{B}})$ from the implementation $\sigma^{\mathcal{W}} : (\Gamma^{\mathcal{W}} \to \Gamma^{\mathcal{S}}) \cup (\Psi^{\mathcal{W}} \to \Psi^{\mathcal{S}}) \cup (Q^{\mathcal{W}} \to Q^{\mathcal{S}})$ as follows. Encode the (state, transition rule) mapping to be equivalent to the state mapping of $\sigma^{\mathcal{W}}$ (trivial since there is only one transition rule). Copy the query mapping in the obvious way.

Since $\mathcal{A}$ and $\mathcal{B}$ are crafted to encode the states, queries, and reachability properties of $\mathcal{W}$ and $\mathcal{S}$, and $\sigma^{\mathcal{A}}$ encodes $\sigma^{\mathcal{W}}$, it is clear that $\sigma^{\mathcal{A}}$ is strongly security-preserving if and only if $\sigma^{\mathcal{W}}$ is. Thus, if we show that $\sigma^{\mathcal{A}}$ is a state-matching reduction, then by Theorem 6, it is strongly security-preserving, and thus $\sigma^{\mathcal{W}}$ is as well. These equivalences in encoding also make it clear that $\sigma^{\mathcal{A}}$ is, indeed, a state-matching reduction (Definition 18), by observation of the properties of $\sigma^{\mathcal{W}}$ as defined in Definition 7.

Thus, we have shown that an implementation is a state-matching implementation *only if* it is strongly security-preserving.

Next we assume that $\sigma^{\mathcal{W}}$ is strongly security-preserving and show that it is a state-matching implementation. Construct $\mathcal{A}$, $\mathcal{B}$, and $\sigma^{\mathcal{A}}$ as above. Since $\sigma^{\mathcal{W}}$ is strongly security-preserving, so is $\sigma^{\mathcal{A}}$. Thus, $\sigma^{\mathcal{A}}$ is a state-matching reduction by Theorem 6. Finally, using an argument as above, $\sigma^{\mathcal{W}}$ is a state-matching implementation.

Thus, we have shown that an implementation is a state-matching implementation *if* it is strongly security-preserving, and thus an implementation is state-matching *if and only if* it is strongly security-preserving. □

B. Proof of Proposition 2

First, we present several requisite definitions.

Definition 19 (Abelian Monoid) An *abelian monoid*, $\mathbf{S} = \langle S, \bullet \rangle$, is a set, S, together with a binary operation, $\bullet$, that satisfies the following properties.

1) **Closure** $\forall a, b \in S, a \bullet b \in S$
2) **Associativity** $\forall a, b, c \in S, (a \bullet b) \bullet c = a \bullet (b \bullet c)$
3) **Commutativity** $\forall a, b \in S, a \bullet b = b \bullet a$
4) **Identity** $\exists 0 \in S, \forall a \in S, a \bullet 0 = a$ ◊

Definition 20 (Partially Ordered Set) A *partially ordered set*, $\mathbf{S} = \langle S, \preceq \rangle$, is a set, S, together with a binary relation, $\preceq$, that satisfies the following properties.

1) **Reflexivity** $\forall a \in S, a \preceq a$
2) **Antisymmetry** $\forall a, b \in S, a \preceq b \wedge b \preceq a \Rightarrow a = b$
3) **Transitivity** $\forall a, b, c \in S, a \preceq b \wedge b \preceq c \Rightarrow a \preceq c$ ◊

Definition 21 (Ordered Abelian Monoid) An *ordered abelian monoid*, $\mathbf{S} = \langle S, \bullet, \preceq \rangle$, is a set, S, together with a binary operator, $\bullet$, and binary relation, $\preceq$, that satisfies the following properties.

1) $\langle S, \bullet \rangle$ is an abelian monoid
2) $\langle S, \preceq \rangle$ is a partially ordered set ◊

Now, we restate the definition of a vector of cost measures from Section V-B.

Definition 13 (Vector of Cost Measures) Given cost measures $\mathbf{N}_1 = \langle N_1, \bullet_1, \preceq_1 \rangle$, $\mathbf{N}_2 = \langle N_2, \bullet_2, \preceq_2 \rangle$, ..., $\mathbf{N}_i = \langle N_i, \bullet_i, \preceq_i \rangle$, let $\mathbf{M} = \langle M, \bullet_*, \preceq_* \rangle$ be the *vector* of cost measures $\mathbf{N}_1, \mathbf{N}_2, \ldots, \mathbf{N}_i$, where:

- $M = N_1 \times N_2 \times \cdots \times N_i$.
- Given $a_1, b_1 \in \mathbf{N}_1$, $a_2, b_2 \in \mathbf{N}_2$, ..., $a_i, b_i \in \mathbf{N}_i$, $\langle a_1, a_2, \ldots, a_i \rangle \bullet_* \langle b_1, b_2, \ldots, b_i \rangle = \langle a_1 \bullet_1 b_1, a_2 \bullet_2 b_2, \ldots, a_i \bullet_i b_i \rangle$.
- Given $a_1, b_1 \in \mathbf{N}_1$, $a_2, b_2 \in \mathbf{N}_2$, ..., $a_i, b_i \in \mathbf{N}_i$, $\langle a_1, a_2, \ldots, a_i \rangle \preceq_* \langle b_1, b_2, \ldots, b_i \rangle$ if and only if $a_1 \preceq_1 b_1 \wedge a_2 \preceq_2 b_2 \wedge \ldots \wedge a_i \preceq_i b_i$. ◊

Proposition 2 *Given cost measures* $\mathbf{N}_1 = \langle N_1, \bullet_1, \preceq_1 \rangle$, $\mathbf{N}_2 = \langle N_2, \bullet_2, \preceq_2 \rangle$, ..., $\mathbf{N}_i = \langle N_i, \bullet_i, \preceq_i \rangle$, *and their vector,* $\mathbf{M} = \langle M, \bullet_*, \preceq_* \rangle$, $\mathbf{M}$ *is a cost measure.*

PROOF All of $\mathbf{N}_1, \mathbf{N}_2, \ldots, \mathbf{N}_i$ are cost measures. By the definition of cost measure, they are all abelian monoids, and thus are all closed, associative, and commutative, and all have identities. Using $\mathbf{N}_1$ as an example, this implies:

1) $\forall a, b \in N_1, a \bullet_1 b \in N_1$
2) $\forall a, b, c \in N_1, (a \bullet_1 b) \bullet_1 c = a \bullet_1 (b \bullet_1 c)$
3) $\forall a, b \in N_1, a \bullet_1 b = b \bullet_1 a$
4) $\exists 0_1 \in N_1, \forall a \in N_1, a \bullet_1 0_1 = a$

Let $A, B, C \in \mathbf{M}$. By the definition of vector of cost measures,

$$A = \langle a_1, a_2, \ldots, a_i \rangle$$

where

$$a_1 \in \mathbf{N}_1, a_2 \in \mathbf{N}_2, \ldots, a_i \in \mathbf{N}_i$$

and similarly for B and C.

By the definition of vector,

$$A \bullet_* B = \langle a_1 \bullet_1 b_1, a_2 \bullet_2 b_2, \ldots, a_i \bullet_i b_i \rangle$$

By the closure of $\mathbf{N}_1, \mathbf{N}_2, \ldots, \mathbf{N}_i$,

$$a_1 \bullet_1 b_1 \in \mathbf{N}_1, a_2 \bullet_2 b_2 \in \mathbf{N}_2, \ldots, a_i \bullet_i b_i \in \mathbf{N}_i$$

$$A \bullet_* B \in \mathbf{M}$$

Thus, $\mathbf{M}$ satisfies the property of *closure*.

By the definition of vector,

$$(A \bullet_* B) \bullet_* C = \langle (a_1 \bullet_1 b_1) \bullet_1 c_1, (a_2 \bullet_2 b_2) \bullet_2 c_2, \ldots, (a_i \bullet_i b_i) \bullet_i c_i \rangle$$

By the associativity of $\mathbf{N}_1, \mathbf{N}_2, \ldots, \mathbf{N}_i$,

$$(A \bullet_* B) \bullet_* C = \langle a_1 \bullet_1 (b_1 \bullet_1 c_1), a_2 \bullet_2 (b_2 \bullet_2 c_2), \ldots, a_i \bullet_i (b_i \bullet_i c_i) \rangle$$

$$(A \bullet_* B) \bullet_* C = A \bullet_* (B \bullet_* C)$$

Thus, $\mathbf{M}$ satisfies the property of *associativity*.

By the definition of vector,

$$A \bullet_* B = \langle a_1 \bullet_1 b_1, a_2 \bullet_2 b_2, \ldots, a_i \bullet_i b_i \rangle$$

By the commutativity of $\mathbf{N}_1, \mathbf{N}_2, \ldots, \mathbf{N}_i$,

$$A \bullet_* B = \langle b_1 \bullet_1 a_1, b_2 \bullet_2 a_2, \ldots, b_i \bullet_i a_i \rangle$$

$$A \bullet_* B = B \bullet_* A$$

Thus, $\mathbf{M}$ satisfies the property of *commutativity*.

By the definition of vector,

$$0 \bullet_* A = \langle 0_1 \bullet_1 a_1, 0_2 \bullet_2 a_2, \ldots, 0_i \bullet_i a_i \rangle$$

By the identity of $\mathbf{N}_1, \mathbf{N}_2, \ldots, \mathbf{N}_i$,

$$0 \bullet_* A = \langle a_1, a_2, \ldots, a_i \rangle$$

$$0 \bullet_* A = A$$

Thus, $\mathbf{M}$ satisfies the property of *identity*.

Since $\mathbf{M}$ satisfies closure, associativity, commutativity, and identity, $\langle M, \bullet_* \rangle$ is an abelian monoid.

All of $\mathbf{N}_1, \mathbf{N}_2, \ldots, \mathbf{N}_i$ are cost measures. Thus, they are all partially ordered sets, and thus are all reflexive, antisymmetric, and transitive. Using $\mathbf{N}_1$ as an example, this implies:

1) $\forall a \in N_1, a \preceq_1 a$
2) $\forall a, b \in N_1, a \preceq_1 b \wedge b \preceq_1 a \Rightarrow a = b$
3) $\forall a, b, c \in N_1, a \preceq_1 b \wedge b \preceq_1 c \Rightarrow a \preceq_1 c$

Let $A, B, C \in \mathbf{M}$. By the definition of vector of cost measures,

$$A = \langle a_1, a_2, \ldots, a_i \rangle$$

where

$$a_1 \in \mathbf{N}_1, a_2 \in \mathbf{N}_2, \ldots, a_i \in \mathbf{N}_i$$

and similarly for B and C.

By the reflexivity of $\mathbf{N}_1, \mathbf{N}_2, \ldots, \mathbf{N}_i$,

$$a_1 \preceq_1 a_1, a_2 \preceq_2 a_2, \ldots, a_i \preceq_i a_i$$

$$A \preceq_* A$$

Thus, $\mathbf{M}$ satisfies the property of *reflexivity*.

Assume $A \preceq_* B \wedge B \preceq_* A$. By the definition of vector,

$$a_1 \preceq_1 b_1 \wedge b_1 \preceq_1 a_1 \wedge a_2 \preceq_2 b_2 \wedge b_2 \preceq_2 a_2 \wedge \ldots \wedge a_i \preceq_i b_i \wedge b_i \preceq_i a_i$$

By the antisymmetry of $\mathbf{N}_1, \mathbf{N}_2, \ldots, \mathbf{N}_i$,

$$a_1 = b_1 \wedge a_2 = b_2 \wedge \ldots \wedge a_i = b_i$$

$$A = B$$

$$A \preceq_* B \wedge B \preceq_* A \Rightarrow A = B$$

Thus, $\mathbf{M}$ satisfies the property of *antisymmetry*.

Assume $A \preceq_* B \wedge B \preceq_* C$. By the definition of vector,

$$a_1 \preceq_1 b_1 \wedge b_1 \preceq_1 c_1 \wedge a_2 \preceq_2 b_2 \wedge b_2 \preceq_2 c_2 \wedge \ldots \wedge a_i \preceq_i b_i \wedge b_i \preceq_i c_i$$

By the transitivity of $\mathbf{N}_1, \mathbf{N}_2, \ldots, \mathbf{N}_i$,

$$a_1 \preceq_1 c_1 \wedge a_2 \preceq_2 c_2 \wedge \ldots \wedge a_i \preceq_i c_i$$

$$A \preceq_* C$$

$$A \preceq_* B \wedge B \preceq_* C \Rightarrow A \preceq_* C$$

Thus, $\mathbf{M}$ satisfies the property of *transitivity*.

Since $\mathbf{M}$ satisfies reflexivity, antisymmetry, and transitivity, $\langle M, \preceq \rangle$ is a partially ordered set.

Since $\langle M, \bullet_* \rangle$ is an abelian monoid and $\langle M, \preceq_* \rangle$ is a partially ordered set, $\mathbf{M} = \langle M, \bullet_*, \preceq_* \rangle$ is an ordered abelian monoid.

All of $\mathbf{N}_1, \mathbf{N}_2, \ldots, \mathbf{N}_i$ are cost measures. Thus, they all satisfy *non-negativity*. Using $\mathbf{N}_1$ as an example, this means:

$$\forall a, b \in N_1, a \preceq_1 a \bullet_1 b$$

By the definition of vector,

$$A \bullet_* B = \langle a_1 \bullet_1 b_1, a_2 \bullet_2 b_2, \ldots, a_i \bullet_i b_i \rangle$$

By the non-negativity of $\mathbf{N}_1, \mathbf{N}_2, \ldots, \mathbf{N}_i$,

$$a_1 \preceq_1 a_1 \bullet_1 b_1 \wedge a_2 \preceq_2 a_2 \bullet_1 b_2 \wedge \cdots \wedge a_i \preceq_i a_i \bullet_i b_i$$

By the definition of vector,

$$A \preceq_* B$$

Thus, $\mathbf{M}$ satisfies the property of *non-negativity*.

Since $\mathbf{M}$ is an ordered abelian monoid and satisfies non-negativity, $\mathbf{M}$ is a cost measure. □

C. Proof of Theorem 3

Theorem 3 *Assuming that workflow constraints are restricted to the binary operators $\{=, \neq\}$ (i.e., constraints expressing binding of duty and separation of duty), the simulation procedure described in Algorithm 1 is pseudo-polynomial in the number of simulated steps and* FPT *with parameter α, the number of actions in the largest task (i.e., the size of the largest disjoint subgraph of the workflow graph).*

PROOF Our proof is by observation of Algorithm 1. The first loop (**for all** $\mathcal{S} = \ldots$) handles assignments and initializations. The final loop (**for all** $\mathcal{S} \in \ldots$) outputs results. The main loop, then, contains all of the computationally intensive code.

The expensive section of the algorithm starts after several nested loops, adding multiplicative factors for number of time steps (T_f/t), number of schemes ($|\Sigma|$), and number of actors. The steps with computational overhead are NEXTACTION, which polls an actor machine for the next action, and WSAT, which calculates whether a particular action can be taken by an actor without causing any workflow instances to become unsatisfiable. We defer in-depth discussion of the WSP problem and its complexity to previous work [17], [18], but it is an NP-complete problem with known algorithms that run in fixed parameterized time with parameter α, the largest number of steps in a workflow task.

By previous work [17], WSP can be solved in $\mathcal{O}(C \cdot A^{\alpha})$, where C is the number of constraints, A is the maximum number of actors, and α is the number of steps in the largest task (i.e., the size of the largest disjoint subgraph of the workflow graph). This greatly exceeds NEXTACTION, which executes a single step in a continuous-time probabilistic machine (polynomial in actor machine size). Thus, the dominant factor in the complexity of Algorithm 1 is $\mathcal{O}(S \cdot C \cdot T \cdot A^{\alpha+1})$, where S is the number of schemes and T is the number of time steps to simulate (T_f/t). Since T is an input, this means the algorithm is pseudo-polynomial in T and FPT in α. Since some consider FPT to be a generalization of pseudo-polynomial time [40], we refer to the complexity of Algorithm 1 as FPT, thus meeting our definition of tractable. □

D. Proof of Theorem 4

Theorem 4 *Given access control scheme $\mathcal{S} = \langle \Gamma^{\mathcal{S}}, \Psi^{\mathcal{S}}, Q^{\mathcal{S}} \rangle$ and access control auxiliary machine $\mathcal{U} = \langle \Gamma^{\mathcal{U}}, \Psi^{\mathcal{U}}, Q^{\mathcal{U}} \rangle$, there exists a state-matching implementation of $\mathcal{S}$ in $\mathcal{S} \circ \mathcal{U}$.*

PROOF By construction. Presented is a mapping, and proof that the mapping satisfies the two properties for it to be a state-matching implementation.

Let $\mathcal{Q} = \mathcal{S} \circ \mathcal{U}$.

The mapping, σ, needs to be able to map every $\gamma \in \Gamma^{\mathcal{S}}$, $\psi \in \Psi^{\mathcal{S}}$, and $q \in Q^{\mathcal{S}}$ in scheme $\mathcal{S}$ to $\gamma^{\mathcal{Q}} \in \Gamma^{\mathcal{Q}}$, $\psi^{\mathcal{Q}} \in \Psi^{\mathcal{Q}}$, and $q^{\mathcal{Q}} \in Q^{\mathcal{Q}}$ in scheme $\mathcal{Q} = \mathcal{S} \circ \mathcal{U}$.

Let $\sigma(\gamma) = \langle \gamma, \gamma_* \rangle$, where γ_* is an arbitrary auxiliary machine-state for AM $\mathcal{U}$. That is, let the AM component of the state be arbitrary, but maintain the original scheme component of the state.

Let $\sigma(\psi) = \psi$ and $\sigma(q) = q$, since by Definition 16 the commands and queries in $\mathcal{S}$ exist unaltered in $\mathcal{S} \circ \mathcal{U}$.

Let γ_0 be a start state in $\mathcal{S}$. Produce $\gamma_0^{\mathcal{Q}}$ in $\mathcal{S} \circ \mathcal{U}$ using σ. Given γ_k such that $\gamma_0 \overset{*}{\mapsto}_{\psi} \gamma_k$, we show that there exists $\gamma_k^{\mathcal{Q}}$ such that $\gamma_0^{\mathcal{Q}} \overset{*}{\mapsto}_{\psi^{\mathcal{Q}}} \gamma_k^{\mathcal{Q}}$ where, for all q and all parameterizations P, $\gamma_k^{\mathcal{Q}} \vdash_{q^{\mathcal{Q}}} (\sigma(P))$ if and only if $\gamma_k \vdash_q (P)$.

From $\gamma_0^{\mathcal{Q}} = \langle \gamma_0, \gamma_* \rangle$, construct $\gamma_k^{\mathcal{Q}}$ by following the same string of commands that were executed in transitioning from γ_0 to γ_k. Since, by Definition 16, commands in $\mathcal{S}$ exist unaltered in $\mathcal{Q}$, the resulting state is $\gamma_k^{\mathcal{Q}} = \langle \gamma_k, \gamma_* \rangle$. Thus, since $\mathcal{S}$'s queries also exist in $\mathcal{Q}$, $\gamma_k^{\mathcal{Q}} \vdash_{q^{\mathcal{Q}}} (\sigma(P))$ if and only if $\gamma_k \vdash_q (P)$.

Therefore, we have proven property (1) for the state-matching implementation.

We prove that property (2) for a state-matching implementation is satisfied by our mapping also by construction. Let $\gamma_0^{\mathcal{Q}}$ be the start-state in $\mathcal{S} \circ \mathcal{U}$ corresponding to γ_0, the start-state in $\mathcal{S}$. Then, if $\gamma_k^{\mathcal{Q}}$ is a state reachable from $\gamma_0^{\mathcal{Q}}$ and $q^{\mathcal{Q}}$ is a query in $\mathcal{S} \circ \mathcal{U}$ whose corresponding query in $\mathcal{S}$ is q, we construct γ_k from γ_0 by executing each $\psi_i \in \Psi^{\mathcal{Q}} = \langle \psi_1, \ldots, \psi_k \rangle$ such that $\exists \psi_i' \in \Psi^{\mathcal{S}} : \delta(\psi_i') = \psi_i$. That is, we execute the same string of commands used in transitioning from $\gamma_0^{\mathcal{Q}}$ to $\gamma_k^{\mathcal{Q}}$, excluding the commands that are a part of $\Psi^{\mathcal{U}}$. By Definition 16, and by an argument similar to above, $\gamma_k \vdash q$ if and only if $\gamma_k^{\mathcal{Q}} \vdash q^{\mathcal{Q}}$.

Therefore, we have proven property (2) for state-matching implementations, and proven that our mapping σ is a state-matching implementation. □

Appendix B
Expressiveness Evaluation Details

A. GMS

The GMS scheme is defined as $\mathcal{G} = \left\langle \Gamma^{\mathcal{G}}, \Psi^{\mathcal{G}}, Q^{\mathcal{G}} \right\rangle$. Its states, $\Gamma^{\mathcal{G}}$, are defined by the sets $\langle U, G, M, T, T_c, O, A, R, TX \rangle$, where:

- U is the set of users
- G is the set of groups
- M is the set of messages
- T is the ordered set of timestamps, including special timestamp ∞
- T_c is the current timestamp
- $O \subseteq U \times G$ is the group ownership relation
- $A \subseteq U \times G$ is the group administration relation
- $R \subseteq U \times G \times T \times T$ is the group membership record
- $TX \subseteq G \times M \times T$ is the messaging transcript

GMS's commands, $\Psi^{\mathcal{G}}$, include the following.

CreateGroup(u, g)
  $G \leftarrow G \cup \{g\}$
  $O \leftarrow O \cup \{\langle u, g \rangle\}$
  $A \leftarrow A \cup \{\langle u, g \rangle\}$
  $R \leftarrow R \cup \{\langle u, g, 0, \infty \rangle\}$

GrantAdmin(o, u, g)
  if $\langle o, g \rangle \in O$
    $A \leftarrow A \cup \{\langle u, g \rangle\}$

RevokeAdmin(o, u, g)
  if $\langle o, g \rangle \in O \ \vee \ o = u$
    $A \leftarrow A - \{\langle u, g \rangle\}$

SAddMember(a, u, g)
  if $\langle a, g \rangle \in A$
    $R \leftarrow R \cup \{\langle u, g, T_c, \infty \rangle\}$
    $T_c \leftarrow T_c + 1$

LAddMember(a, u, g)
  if $\langle a, g \rangle \in A$
    $R \leftarrow R \cup \{\langle u, g, 0, \infty \rangle\}$

SRemoveMember(a, u, g)
  if $\langle a, g \rangle \in A \ \vee \ a = u$
    $R \leftarrow R - \{\langle u, g, t, t' \rangle : \langle u, g, t, t' \rangle \in R\}$

LRemoveMember(a, u, g)
  if $\langle a, g \rangle \in A$
    $R \leftarrow R \cup \{\langle u, g, t, T_c \rangle : \langle u, g, t, \infty \rangle \in R\}$
    $R \leftarrow R - \{\langle u, g, t, \infty \rangle : \langle u, g, t, \infty \rangle \in R\}$

Post(u, g, m)
  if $\exists t \in T : \langle u, g, t, \infty \rangle \in R$
    $TX \leftarrow TX \cup \{\langle g, m, T_c \rangle\}$
    $T_c \leftarrow T_c + 1$

Finally, GMS's queries, $Q^{\mathcal{G}}$, include the following.

Access(u, m)
  $\exists g \in G, t_l, t, t_u \in T :$
    $\langle u, g, t_l, t_u \rangle \in R \ \wedge \ \langle g, m, t \rangle \in TX \ \wedge \ t_l \leq t \leq t_u$

B. RBAC

RBAC is a role-based access control scheme,[6] $\mathcal{R} = \left\langle \Gamma^{\mathcal{R}}, \Psi^{\mathcal{R}}, Q^{\mathcal{R}} \right\rangle$. Its states, $\Gamma^{\mathcal{R}}$, are defined by the sets $\langle U, R, P, UA, PA \rangle$, where:

- U is the set of users
- R is the set of roles
- P is the set of permissions
- $UA \subseteq U \times R$ is the user-assignment relation
- $PA \subseteq P \times R$ is the permission-assignment relation

RBAC's commands, $\Psi^{\mathcal{R}}$, include the following.

AddRole(a, r)
  if $\langle a, \text{admin} \rangle \in UA$
    $R \leftarrow R \cup \{r\}$

DeleteRole(a, r)
  if $\langle a, \text{admin} \rangle \in UA$
    $R \leftarrow R - \{r\}$

AssignUser(a, u, r)
  if $\langle a, \text{admin} \rangle \in UA$
    $UA \leftarrow UA \cup \{\langle u, r \rangle\}$

DeassignUser(a, u, r)
  if $\langle a, \text{admin} \rangle \in UA$
    $UA \leftarrow UA - \{\langle u, r \rangle\}$

GrantPermission(a, p, r)
  if $\langle a, \text{admin} \rangle \in UA$
    $PA \leftarrow PA \cup \{\langle p, r \rangle\}$

RevokePermission(a, p, r)
  if $\langle a, \text{admin} \rangle \in UA$
    $PA \leftarrow PA - \{\langle p, r \rangle\}$

Finally, RBAC's queries, $Q^{\mathcal{R}}$, include the following.

[6] There are many competing definitions for role-based access control schemes in the literature. We derive our definition of RBAC from NIST RBAC [26]. We exclude from the state elements to maintain sessions as well as several derived relations, changes which have also been suggested by others [41].

```
Access(u, p)
    ∃r ∈ R : ⟨u,r⟩ ∈ UR  ∧  ⟨r,p⟩ ∈ PA

Assigned(u, r)
    ⟨u,r⟩ ∈ UR
```

We extend RBAC with AM $\mathcal{U} = \langle \Gamma^{\mathcal{U}}, \Psi^{\mathcal{U}}, Q^{\mathcal{U}} \rangle$. The AM's states, $\Gamma^{\mathcal{U}}$, are defined by the sets $\langle G, GM \rangle$, where:

- G is the set of groups
- $GM \subseteq G \times P$ is the group-message relation

The extension's commands, $\Psi^{\mathcal{U}}$, include the following.

```
CreateGroup(u, g)
    if ⟨u,admin⟩ ∈ UA
        G ← G ∪ {g}

AssociateWithGroup(u, g, p)
    if ⟨u,admin⟩ ∈ UA
        GM ← GM ∪ {⟨g,p⟩}
```

Finally, $Q^{\mathcal{U}} = \emptyset$, and thus the extension does not add any queries to the scheme.

We can now demonstrate the implementation of GMS using RBAC $\circ\, \mathcal{U}$. To describe an implementation $\sigma^{\mathcal{R}}$ of GMS in RBAC $\circ\, \mathcal{U}$, we must describe the state-to-state mapping ($\sigma^{\mathcal{R}}_{\Gamma}$), the command-to-command mapping ($\sigma^{\mathcal{R}}_{\Psi}$), and the query-to-query mapping ($\sigma^{\mathcal{R}}_{Q}$).

First, we describe $\sigma^{\mathcal{R}}_{\Gamma}$, which maps a state in GMS, $\gamma^{\mathcal{G}} \in \Gamma^{\mathcal{G}}$, to a state in RBAC$\circ\mathcal{U}$, $\sigma^{\mathcal{R}}_{\Gamma}(\gamma^{\mathcal{G}}) = \gamma^{\mathcal{R}} \in \Gamma^{\mathcal{R}}$, as follows. Users are mapped in the obvious way. Each message m posted to any group is mapped to a permission $m \in P$, which grants read access to the message. Each such permission is then assigned to role $r^m \in R$. Each user is assigned to role r^m for each message m she has access to. We also store and assign roles m^g, o^g, and a^g for current membership, ownership, and administration of group $g \in G$, respectively. These roles allow certain commands to be executed, but do not correspond to a permission in P.

Now, we describe $\sigma^{\mathcal{R}}_{\Psi}$, which maps commands in GMS to strings of commands in RBAC $\circ\, \mathcal{U}$.

- `CreateGroup(u, g)` in GMS is mapped to the sequence `CreateGroup(u, g)`, `AddRole(u, m^g)`, `AssignUser(u, u, m^g)`, `AddRole(u, o^g)`, `AssignUser(u, u, o^g)`, `AddRole(u, a^g)`, `AssignUser(u, u, a^g)` in RBAC $\circ\, \mathcal{U}$.
- `GrantAdmin(u, u_2, g)` in GMS is mapped to `AssignUser(u, u_2, a^g)` in RBAC $\circ\, \mathcal{U}$.
- `RevokeAdmin(u, u_2, g)` in GMS is mapped to `DeassignUser(u, u_2, a^g)` in RBAC $\circ\, \mathcal{U}$.
- `SAddMember(u, u_2, g)` in GMS is mapped to `AssignUser(u, u_2, m^g)` in RBAC $\circ\, \mathcal{U}$.
- `LAddMember(u, u_2, g)` in GMS is mapped to `AssignUser(u, u_2, m^g)` in RBAC $\circ\, \mathcal{U}$, followed by `AssignUser(u, u_2, r^m)` for each m such that $\langle g, m \rangle \in GM$.
- `SRemoveMember(u, u_2, g)` in GMS is mapped to `DeassignUser(u, u_2, m^g)` in RBAC $\circ\, \mathcal{U}$, followed by `DeassignUser(u, u_2, r^m)` for each m such that $\langle g, m \rangle \in GM$.
- `LRemoveMember(u, u_2, g)` in GMS is mapped to `DeassignUser(u, u_2, m^g)` in RBAC $\circ\, \mathcal{U}$.
- `Post(u, g, m)` in GMS is mapped to `AssociateWithGroup(u, g, m)` in RBAC $\circ\, \mathcal{U}$, followed by `AssignUser(u, u_2, r^m)` for each u_2 such that $\langle u_2, m^g \rangle \in UA$.

Finally, $\sigma^{\mathcal{R}}_{Q}$ maps `Access(u, m)` in GMS to `Access(u, p^m)` in RBAC $\circ\, \mathcal{U}$.

Theorem 7 *$\sigma^{\mathcal{R}}$ is a state-matching implementation of GMS in RBAC $\circ\, \mathcal{U}$.*

PROOF First, we prove property (1) for state-matching implementations.

Let γ_0 be a start state in GMS. Produce $\gamma^{\mathcal{R}}_0$ in RBAC $\circ\, \mathcal{U}$ using $\sigma^{\mathcal{R}}_{\Gamma}$. Given γ_k such that $\gamma_0 \overset{*}{\mapsto} \gamma_k$, we show that there exists $\gamma^{\mathcal{R}}_k$ such that $\gamma^{\mathcal{R}}_0 \overset{*}{\mapsto} \gamma^{\mathcal{R}}_k$ where, for all queries $q = \langle n, P, \vdash \rangle \in Q^{\mathcal{G}}$ and parameterizations $p \in P^*$, $\gamma^{\mathcal{R}}_k \vdash q^{\mathcal{R}}(p)$ if and only if $\gamma_k \vdash q(p)$.

Consider the case where $\gamma_k = \gamma_0$, then let $\gamma^{\mathcal{R}}_k = \gamma^{\mathcal{R}}_0$. By inspection of the procedure for $\sigma^{\mathcal{R}}_{\Gamma}$, $\gamma_k \vdash q(p)$ if and only if $\gamma^{\mathcal{R}}_k \vdash q^{\mathcal{R}}(p)$.

Next, consider some arbitrary γ_k reachable from γ_0. We construct $\gamma^{\mathcal{R}}_k$ that is reachable from $\gamma^{\mathcal{R}}_0$ and that answers every $q^{\mathcal{R}}(p)$ in the same way that γ_k answers $q(p)$, as per $\sigma^{\mathcal{R}}_{\Psi}$. Since $\gamma_0 \overset{*}{\mapsto} \gamma_k$, there exists a sequence of commands $\langle \psi_1 = \langle n_1, P_1, e_1 \rangle, \ldots, \psi_k = \langle n_k, P_k, e_k \rangle \rangle$ and a sequence of parameterizations $\langle p_1 \in P^*_1, \ldots, p_k \in P^*_k \rangle$ of these commands such that $\gamma_k = e_k(\ldots e_1(\gamma_0, p_1), \ldots, p_k)$. For each command/parameterization pair $\langle \psi_i, p_i \rangle$, we show that the same queries change value between γ_{i-1} and $\gamma_i = e_i(\gamma_{i-1}, p_i)$ and between $\gamma^{\mathcal{R}}_{i-1} = \sigma^{\mathcal{R}}_{\Psi}(\gamma_{i-1})$ and $\gamma^{\mathcal{R}}_i = \sigma^{\mathcal{R}}_{\Psi}(\gamma_i)$. Thus, by induction it will be clear that $\gamma_k \vdash q(p)$ if and only if $\gamma^{\mathcal{R}}_k \vdash q^{\mathcal{R}}(p)$.

- If $\langle \psi_i, p_i \rangle$ is an instance of `CreateGroup`, `GrantAdmin`, `RevokeAdmin`, `SAddMember`, or `LRemoveMember`, no queries are changed between γ_{i-1} and γ_i. Since the corresponding operations in RBAC $\circ\, \mathcal{U}$ alter only the role relation for roles with no permissions, similarly no queries are changed between $\gamma^{\mathcal{R}}_{i-1}$ and $\gamma^{\mathcal{R}}_i$.
- If ψ_i is `LAddMember`, let $p_i = \langle u, u_2, g \rangle$, then `Access` queries are changed to TRUE for user u_2 and all messages in group g. These same `Access` queries are explicitly made TRUE by $\sigma^{\mathcal{R}}_{\Psi}$ by adding u_2 to roles that grant precisely these permissions.
- If ψ_i is `SRemoveMember`, let $p_i = \langle u, u_2, g \rangle$, then `Access` queries are made FALSE for user u_2 and all messages in group g. These same `Access` queries are explicitly made FALSE by $\sigma^{\mathcal{R}}_{\Psi}$ by removing u_2 from the only roles with these permissions.
- If ψ_i is `Post`, let $p_i = \langle u, g, m \rangle$, then `Access` queries are changed to TRUE for all users in group g and message m. These same `Access` queries are explicitly made TRUE by $\sigma^{\mathcal{R}}_{\Psi}$ by adding all users in group g to the role with the permission corresponding to m.

Thus, we have proven property (1) for state-matching implementations, and we proceed to prove property (2).

Let $\gamma^{\mathcal{R}}_0$ be the start-state in RBAC $\circ\, \mathcal{U}$ corresponding to γ_0, the start-state in GMS. Then, if $\gamma^{\mathcal{R}}_k$ is a state reachable from

$\gamma_0^{\mathcal{R}}$, we construct γ_k, a state in GMS reachable from γ_0, as follows.

1) Consider each `Access` query changed to TRUE (i.e., each permission granted) between $\gamma_0^{\mathcal{R}}$ and $\gamma_k^{\mathcal{R}}$. Let $p = \langle u, m \rangle$ be the parameterization of the `Access` query in question. If permission m corresponds to a message in GMS, execute `CreateGroup` to create a new group, and use `SAddMember` to add u to this group (note that no queries have changed yet, since the new group has no messages). Finally, `Post` message m in the new group, granting only the access in question.
2) Consider each `Access` query changed to FALSE (i.e., each permission revoked) between $\gamma_0^{\mathcal{R}}$ and $\gamma_k^{\mathcal{R}}$. Let $p = \langle u, m \rangle$ be the parameterization of the `Access` query in question. If permission m corresponds to a message in GMS, then since u can access m in γ_0, there exists group g that u access to m through (i.e., $\exists t_l, t, t_u \in T : \langle u, g, t_l, t_u \rangle \in R \wedge \langle g, m, t \rangle \in TX \wedge t_l \leq t \leq t_u$). Execute `CreateGroup` to create a new group, and use `SAddMember` to add u to this group. Next, `Post` all messages that u has access to through g to this new group, with the exception of m (note that no queries have changed yet; user u has not gained or lost any accesses). Finally, use `SRemoveMember` to remove u from g, revoking only the access in question.

These changes to transition between γ_0 and γ_k in GMS allow γ_k to answer each query in the same way as $\gamma_k^{\mathcal{R}}$. Thus, $\gamma_k \vdash q(p)$ if and only if $\gamma_k^{\mathcal{R}} \vdash q^{\mathcal{R}}(p)$. Therefore, we have proven property (2) for state-matching implementations, and proven that the implementation $\sigma^{\mathcal{R}}$ is a state-matching implementation. □

C. DAC

DAC is a discretionary access control scheme based on the Graham-Denning scheme [25], $\mathcal{D} = \langle \Gamma^{\mathcal{D}}, \Psi^{\mathcal{D}}, Q^{\mathcal{D}} \rangle$. Its states, $\Gamma^{\mathcal{D}}$, are defined by the sets $\langle S, O, I, M \rangle$, where:

- S is the set of subjects
- O is the set of objects
- I is the set of access rights
- $M : S \times O \to 2^I$ is the access matrix

DAC's commands, $\Psi^{\mathcal{D}}$, include the following.

`Grant(`s, t, o, i`)`
  `if` $\texttt{own} \in M(s,o) \;\wedge\; i \neq \texttt{own}$
    $M(t,o) \leftarrow M(t,o) \cup \{i\}$

`Revoke(`s, t, o, i`)`
  `if` $\texttt{own} \in M(s,o) \;\wedge\; i \neq \texttt{own}$
    $M(t,o) \leftarrow M(t,o) - \{i\}$

Finally, DAC's queries, $Q^{\mathcal{D}}$, include the following.

`Access(`s, o, i`)`
  $i \in M(s,o)$

We extend DAC with AM $\mathcal{V} = \langle \Gamma^{\mathcal{V}}, \Psi^{\mathcal{V}}, Q^{\mathcal{V}} \rangle$. The AM's states, $\Gamma^{\mathcal{V}}$, are defined by the sets $\langle G, GM, W, A, B \rangle$, where:

- G is the set of groups
- $GM \subseteq G \times O$ is the group-message relation
- $W \subseteq S \times G$ is the group ownership relation
- $A \subseteq S \times G$ is the group administration relation
- $B \subseteq S \times G$ is the group membership relation

The extension's commands, $\Psi^{\mathcal{V}}$, include the following.

`CreateGroup(`s, g`)`
  $G \leftarrow G \cup \{g\}$
  $W \leftarrow W \cup \{\langle s, g \rangle\}$
  $A \leftarrow A \cup \{\langle s, g \rangle\}$
  $B \leftarrow B \cup \{\langle s, g \rangle\}$

`AssociateWithGroup(`s, g, o`)`
  `if` $\langle s, g \rangle \in B$
    $GM \leftarrow GM \cup \{\langle g, o \rangle\}$

`GrantAdmin(`s, t, g`)`
  `if` $\langle s, g \rangle \in W$
    $A \leftarrow A \cup \{\langle t, g \rangle\}$

`RevokeAdmin(`s, t, g`)`
  `if` $\langle s, g \rangle \in W \;\vee\; s = t$
    $A \leftarrow A - \{\langle t, g \rangle\}$

`GrantMember(`s, t, g`)`
  `if` $\langle s, g \rangle \in A$
    $B \leftarrow B \cup \{\langle u, g \rangle\}$

`RevokeMember(`s, t, g`)`
  `if` $\langle s, g \rangle \in A$
    $B \leftarrow B - \{\langle u, g \rangle\}$

Finally, $Q^{\mathcal{V}} = \emptyset$, and thus the extension does not add any queries to the scheme.

We can now demonstrate the implementation of GMS using DAC $\circ\, \mathcal{V}$. To describe an implementation $\sigma^{\mathcal{D}}$ of GMS in DAC $\circ\, \mathcal{V}$, we must describe the state-to-state mapping ($\sigma_{\Gamma}^{\mathcal{D}}$), the command-to-command mapping ($\sigma_{\Psi}^{\mathcal{D}}$), and the query-to-query mapping ($\sigma_{Q}^{\mathcal{D}}$).

First, we describe $\sigma_{\Gamma}^{\mathcal{D}}$, which maps a state in GMS, $\gamma^{\mathcal{G}} \in \Gamma^{\mathcal{G}}$, to a state in DAC $\circ\, \mathcal{V}$, $\sigma_{\Gamma}^{\mathcal{D}}(\gamma^{\mathcal{G}}) = \gamma^{\mathcal{D}} \in \Gamma^{\mathcal{D}}$, as follows. Users in GMS are mapped to subjects in DAC $\circ\, \mathcal{V}$. Each message m posted to any group is mapped to an object $m \in O$. Since GMS considers only read access, DAC's I is statically set to $\{\texttt{r}\}$. The group-message relation is stored in $\mathcal{V}$ along with relations for group ownership, administration, and membership. DAC's M maintains a "flattened" view of the current accesses, and thus $M(s, o) = \{\texttt{r}\}$ if the GMS user corresponding to s has access to the GMS message corresponding to object o. The projection of the accesses maintained in M will be updated by $\sigma_{\Psi}^{\mathcal{D}}$ whenever the more semantically meaningful structures in $\mathcal{V}$'s state are changed.

Next, we describe $\sigma_{\Psi}^{\mathcal{D}}$, which maps commands in GMS to strings of commands in DAC $\circ\, \mathcal{V}$.

- `CreateGroup(`u, g`)` in GMS is mapped to `CreateGroup(`u, g`)` in DAC $\circ\, \mathcal{V}$.
- `GrantAdmin(`u, u_2, g`)` in GMS is mapped to `GrantAdmin(`u, u_2, g`)` in DAC $\circ\, \mathcal{V}$.
- `RevokeAdmin(`u, u_2, g`)` in GMS is mapped to `RevokeAdmin(`u, u_2, g`)` in DAC $\circ\, \mathcal{V}$.
- `SAddMember(`u, u_2, g`)` in GMS is mapped to `GrantMember(`u, u_2, g`)` in DAC $\circ\, \mathcal{V}$.
- `LAddMember(`u, u_2, g`)` in GMS is mapped to `GrantMember(`u, u_2, g`)` in DAC $\circ\, \mathcal{V}$, followed

by `Grant(`u`,` u_2`,` m`, r)` for each m such that $\langle g, m\rangle \in GM$.

- `SRemoveMember(`u`,` u_2`,` g`)` in GMS is mapped to `RevokeMember(`u`,` u_2`,` g`)` in DAC $\circ\, \mathcal{V}$, followed by `Revoke(`u`,` u_2`,` m`, r)` for each m such that $\langle g, m\rangle \in GM$.
- `LRemoveMember(`u`,` u_2`,` g`)` in GMS is mapped to `RevokeMember(`u`,` u_2`,` g`)` in DAC $\circ\, \mathcal{V}$.
- `Post(`u`,` g`,` m`)` in GMS is mapped to `AssociateWithGroup(`u`,` g`,` m`)` in DAC $\circ\, \mathcal{V}$, followed by `Grant(`u`,` u_2`,` m`, r)` for each u_2 such that $\langle u_2, g\rangle \in B$.

Finally, $\sigma_Q^{\mathcal{D}}$ maps `Access(`u`,` m`)` in GMS to `Access(`u`,` m`, r)` in DAC $\circ\, \mathcal{V}$.

Theorem 8 *$\sigma^{\mathcal{D}}$ is a state-matching implementation of GMS in DAC $\circ\, \mathcal{V}$.*

PROOF First, we prove property (1) for state-matching implementations.

Let γ_0 be a start state in GMS. Produce $\gamma_0^{\mathcal{D}}$ in DAC$\circ\mathcal{V}$ using $\sigma_\Gamma^{\mathcal{D}}$. Given γ_k such that $\gamma_0 \overset{*}{\mapsto} \gamma_k$, we show that there exists $\gamma_k^{\mathcal{D}}$ such that $\gamma_0^{\mathcal{D}} \overset{*}{\mapsto} \gamma_k^{\mathcal{D}}$ where, for all queries $q = \langle n, P, \vdash\rangle \in Q^{\mathcal{G}}$ and parameterizations $p \in P^*$, $\gamma_k^{\mathcal{D}} \vdash q^{\mathcal{D}}(p)$ if and only if $\gamma_k \vdash q(p)$.

Consider the case where $\gamma_k = \gamma_0$, then let $\gamma_k^{\mathcal{D}} = \gamma_0^{\mathcal{D}}$. By inspection of the procedure for $\sigma_\Gamma^{\mathcal{D}}$, $\gamma_k \vdash q(p)$ if and only if $\gamma_k^{\mathcal{D}} \vdash q^{\mathcal{D}}(p)$.

Next, consider some arbitrary γ_k reachable from γ_0. We construct $\gamma_k^{\mathcal{D}}$ that is reachable from $\gamma_0^{\mathcal{D}}$ and that answers every $q^{\mathcal{D}}(p)$ in the same way that γ_k answers $q(p)$, as per $\sigma_\Psi^{\mathcal{D}}$. Since $\gamma_0 \overset{*}{\mapsto} \gamma_k$, there exists a sequence of commands $\langle \psi_1 = \langle n_1, P_1, e_1\rangle, \ldots, \psi_k = \langle n_k, P_k, e_k\rangle\rangle$ and a sequence of parameterizations $\langle p_1 \in P_1^*, \ldots, p_k \in P_k^*\rangle$ of these commands such that $\gamma_k = e_k(\ldots e_1(\gamma_0, p_1), \ldots, p_k)$. For each command/parameterization pair $\langle \psi_i, p_i\rangle$, we show that the same queries change value between γ_{i-1} and $\gamma_i = e_i(\gamma_{i-1}, p_i)$ and between $\gamma_{i-1}^{\mathcal{D}} = \sigma_\Psi^{\mathcal{D}}(\gamma_{i-1})$ and $\gamma_i^{\mathcal{D}} = \sigma_\Psi^{\mathcal{D}}(\gamma_i)$. Thus, by induction it will be clear that $\gamma_k \vdash q(p)$ if and only if $\gamma_k^{\mathcal{D}} \vdash q^{\mathcal{D}}(p)$.

- If $\langle \psi_i, p_i\rangle$ is an instance of `CreateGroup`, `GrantAdmin`, `RevokeAdmin`, `SAddMember`, or `LRemoveMember`, no queries are changed between γ_{i-1} and γ_i. Since the corresponding operations in DAC $\circ\, \mathcal{V}$ alter only the extension state (not granting any new accesses), similarly no queries are changed between $\gamma_{i-1}^{\mathcal{D}}$ and $\gamma_i^{\mathcal{D}}$.
- If ψ_i is `LAddMember`, let $p_i = \langle u, u_2, g\rangle$, then `Access` queries are changed to TRUE for user u_2 and all messages in group g. These same `Access` queries are explicitly made TRUE by $\sigma_\Psi^{\mathcal{D}}$ through executions of the `Grant` command.
- If ψ_i is `SRemoveMember`, let $p_i = \langle u, u_2, g\rangle$, then `Access` queries are made FALSE for user u_2 and all messages in group g. These same `Access` queries are explicitly made FALSE by $\sigma_\Psi^{\mathcal{D}}$ through executions of the `Revoke` command.
- If ψ_i is `Post`, let $p_i = \langle u, g, m\rangle$, then `Access` queries are changed to TRUE for all users in group g and message m. These same `Access` queries are explicitly made TRUE by $\sigma_\Psi^{\mathcal{D}}$ through executions of the `Grant` command.

Thus, we have proven property (1) for state-matching implementations, and we proceed to prove property (2).

Let $\gamma_0^{\mathcal{D}}$ be the start-state in DAC $\circ\, \mathcal{V}$ corresponding to γ_0, the start-state in GMS. Then, if $\gamma_k^{\mathcal{D}}$ is a state reachable from $\gamma_0^{\mathcal{D}}$, we construct γ_k, a state in GMS reachable from γ_0, as follows.

1) Consider each `Access` query changed to TRUE (i.e., each access granted) between $\gamma_0^{\mathcal{D}}$ and $\gamma_k^{\mathcal{D}}$. Let $p = \langle u, m, \texttt{r}\rangle$ be the parameterization of the `Access` query in question (if the access is any right but `r`, it will not affect the GMS state). If object m corresponds to a message in GMS, execute `CreateGroup` to create a new group, and use `SAddMember` to add u to this group (note that no queries have changed yet, since the new group has no messages). Finally, `Post` message m in the new group, granting only the access in question.
2) Consider each `Access` query changed to FALSE (i.e., each access revoked) between $\gamma_0^{\mathcal{D}}$ and $\gamma_k^{\mathcal{D}}$. Let $p = \langle u, m, \texttt{r}\rangle$ be the parameterization of the `Access` query in question. If object m corresponds to a message in GMS, then since u can access m in γ_0, there exists group g that u has access to m through (i.e., $\exists t_l, t, t_u \in T : \langle u, g, t_l, t_u\rangle \in R \wedge \langle g, m, t\rangle \in TX \wedge t_l \leq t \leq t_u$). Execute `CreateGroup` to create a new group, and use `SAddMember` to add u to this group. Next, `Post` all messages that u has access to through g to this new group, with the exception of m (note that no queries have changed yet; user u has not gained or lost any accesses). Finally, use `SRemoveMember` to remove u from g, revoking only the access in question.

These changes to transition between γ_0 and γ_k in GMS allow γ_k to answer each query in the same way as $\gamma_k^{\mathcal{D}}$. Thus, $\gamma_k \vdash q(p)$ if and only if $\gamma_k^{\mathcal{D}} \vdash q^{\mathcal{D}}(p)$. Therefore, we have proven property (2) for state-matching implementations, and proven that the implementation $\sigma^{\mathcal{D}}$ is a state-matching implementation. □

D. SD3-GM

SD3-GM is the group-messaging instantiation of the SD3 trust management scheme, $\mathcal{S} = \langle \Gamma^{\mathcal{S}}, \Psi^{\mathcal{S}}, Q^{\mathcal{S}}\rangle$. Its states, $\Gamma^{\mathcal{S}}$, are defined by the set P, the set of policy sentences written in the SD3 policy language. The following static policy sentences enforce the access semantics and the current membership semantics.

```
ACCESS(U, M) :- MEMBER(U, G, T1, T2),
                POST(G, M, T),
                LESSEQ(T1, T),
                LESSEQ(T, T2)
CURRMEMBER(U, G) :- MEMBER(U, G, T, ∞)
```

Here, `LESSEQ` is the inherent "less-than-or-equal" predicate for timestamps.

SD3-GM's commands, $\Psi^{\mathcal{S}}$, include the following.

```
CreateGroup(u,g)
  P ← P ∪ {"OWN(u,g)","ADMIN(u,g)","MEMBER(u,g,0,∞)"}

GrantAdmin(o,u,g)
  if eval("OWN(o,g)")
    P ← P ∪ {"ADMIN(u,g)"}
```

```
RevokeAdmin(o,u,g)
    if eval("OWN(o,g)") ∨ o = u
        P ← P − {"ADMIN(u,g)"}

SAddMember(a,u,g)
    if eval("ADMIN(a,g)")
        P ← P ∪ {"MEMBER(u,g,T_c,∞)"}
        P ← P ∪ {"TIME(T_c+1)"} − {"TIME(T_c)"}

LAddMember(a,u,g)
    if eval("ADMIN(a,g)")
        P ← P ∪ {"MEMBER(u,g,0,∞)"}

SRemoveMember(a,u,g)
    if eval("ADMIN(o,g)") ∨ o = u
        P ← P − {"MEMBER(u,g,*,*)"}

LRemoveMember(a,u,g)
    if eval("ADMIN(a,g)")
        P ← P∪{"MEMBER(u,g,t,T_c)"}−{"MEMBER(u,g,t,∞)"}
        (where "MEMBER(u,g,t,∞)" ∈ P)

Post(u,g,m)
    if eval("CURRMEMBER(u,g)")
        P ← P ∪ {"POST(g,m,T_c)"}
        P ← P ∪ {"TIME(T_c+1)"} − {"TIME(T_c)"}
```

Finally, SD3-GM's queries, $Q^{\mathcal{S}}$, include the following.

```
Access(u,m)
    eval("ACCESS(u,m)")
```

To describe an implementation $\sigma^{\mathcal{S}}$ of GMS in SD3-GM, we must describe the state-to-state mapping ($\sigma^{\mathcal{S}}_{\Gamma}$), the command-to-command mapping ($\sigma^{\mathcal{S}}_{\Psi}$), and the query-to-query mapping ($\sigma^{\mathcal{S}}_{Q}$).

First, we describe $\sigma^{\mathcal{S}}_{\Gamma}$, which maps a state in GMS, $\gamma^{\mathcal{G}} \in \Gamma^{\mathcal{G}}$, to a state in SD3-GM, $\sigma^{\mathcal{S}}_{\Gamma}(\gamma^{\mathcal{G}}) = \gamma^{\mathcal{S}} \in \Gamma^{\mathcal{S}}$, as follows.

- For T_c in GMS, "TIME(T_c)" is added to P in SD3-GM.
- For each $\langle u, g\rangle \in O$ in GMS, "OWN(u,g)" is added to P in SD3-GM.
- For each $\langle u, g\rangle \in A$ in GMS, "ADMIN(u,g)" is added to P in SD3-GM.
- For each $\langle u, g, t_1, t_2\rangle \in R$ in GMS, "MEMBER(u,g,t_1,t_2)" is added to P in SD3-GM.
- For each $\langle g, m, t\rangle \in TX$ in GMS, "POST(g,m,t)" is added to P in SD3-GM.

Then, $\sigma^{\mathcal{S}}_{\Psi}$ and $\sigma^{\mathcal{S}}_{Q}$ are both identity mappings. That is, commands and queries are both mapped to their identically-named versions in SD3-GM.

Theorem 9 *$\sigma^{\mathcal{S}}$ is a state-matching implementation of GMS in SD3-GM.*

PROOF First, we prove property (1) for state-matching implementations.

Let γ_0 be a start state in GMS. Produce $\gamma^{\mathcal{S}}_0$ in SD3-GM using $\sigma^{\mathcal{S}}_{\Gamma}$. Given γ_k such that $\gamma_0 \overset{*}{\mapsto} \gamma_k$, we show that there exists $\gamma^{\mathcal{S}}_k$ such that $\gamma^{\mathcal{S}}_0 \overset{*}{\mapsto} \gamma^{\mathcal{S}}_k$ where, for all queries $q = \langle n, P, \vdash\rangle \in Q^{\mathcal{G}}$ and parameterizations $p \in P^*$, $\gamma^{\mathcal{S}}_k \vdash q^{\mathcal{S}}(p)$ if and only if $\gamma_k \vdash q(p)$.

Consider the case where $\gamma_k = \gamma_0$, then let $\gamma^{\mathcal{S}}_k = \gamma^{\mathcal{S}}_0$. By inspection of the procedure for $\sigma^{\mathcal{S}}_{\Gamma}$, $\gamma_k \vdash q(p)$ if and only if $\gamma^{\mathcal{S}}_k \vdash q^{\mathcal{S}}(p)$.

Next, consider some arbitrary γ_k reachable from γ_0. We construct $\gamma^{\mathcal{S}}_k$ this is reachable from $\gamma^{\mathcal{S}}_0$ and that answers every $q^{\mathcal{S}}(p)$ in the same way that γ_k answers $q(p)$, as per $\sigma^{\mathcal{S}}_{\Psi}$. Since $\gamma_0 \overset{*}{\mapsto} \gamma_k$, there exists a sequence of commands $\langle \psi_1 = \langle n_1, P_1, e_1\rangle, \ldots, \psi_k = \langle n_k, P_k, e_k\rangle\rangle$ and a sequence of parameterizations $\langle p_1 \in P^*_1, \ldots, p_k \in P^*_k\rangle$ of these commands such that $\gamma_k = e_k(\ldots e_1(\gamma_0, p_1), \ldots, p_k)$. For each command/parameterization pair $\langle \psi_i, p_i\rangle$, we show that the same queries change value between γ_{i-1} and $\gamma_i = e_i(\gamma_{i-1}, p_i)$ and between $\gamma^{\mathcal{S}}_{i-1} = \sigma^{\mathcal{S}}_{\Psi}(\gamma_{i-1})$ and $\gamma^{\mathcal{S}}_i = \sigma^{\mathcal{S}}_{\Psi}(\gamma_i)$. Thus, by induction it will be clear that $\gamma_k \vdash q(p)$ if and only if $\gamma^{\mathcal{S}}_k \vdash q^{\mathcal{S}}(p)$. In the case of $\sigma^{\mathcal{S}}$, the implementation is a strict bisimulation, and GMS and SD3-GM move in strict lock-step.

- If $\langle \psi_i, p_i\rangle$ is an instance of `CreateGroup`, `GrantAdmin`, `RevokeAdmin`, `SAddMember`, or `LRemoveMember`, no queries are changed between γ_{i-1} and γ_i. Since the corresponding operations in SD3-GM behave identically, no queries are changed between $\gamma^{\mathcal{R}}_{i-1}$ and $\gamma^{\mathcal{R}}_i$.
- If ψ_i is `LAddMember`, let $p_i = \langle u, u_2, g\rangle$, then `Access` queries are changed to TRUE for user u_2 and all messages in group g. These same `Access` queries are also made TRUE by $\sigma^{\mathcal{S}}_{\Psi}$.
- If ψ_i is `SRemoveMember`, let $p_i = \langle u, u_2, g\rangle$, then `Access` queries are made FALSE for user u_2 and all messages in group g. These same `Access` queries are also made FALSE by $\sigma^{\mathcal{S}}_{\Psi}$.
- If ψ_i is `Post`, let $p_i = \langle u, g, m\rangle$, then `Access` queries are changed to TRUE for all users in group g and message m. These same `Access` queries are also made TRUE by $\sigma^{\mathcal{S}}_{\Psi}$.

Thus, we have proven property (1) for state-matching implementations, and we proceed to prove property (2).

Let $\gamma^{\mathcal{S}}_0$ be the start-state in SD3-GM corresponding to γ_0, the start-state in GMS. Then, if $\gamma^{\mathcal{S}}_k$ is a state reachable from $\gamma^{\mathcal{S}}_0$, we construct γ_k, a state in GMS reachable from γ_0, as follows. Since both $\sigma^{\mathcal{S}}_{\Gamma}$ and $\sigma^{\mathcal{S}}_{Q}$ are the identity mapping, for each command and parameterization executed between $\gamma^{\mathcal{S}}_0$ and $\gamma^{\mathcal{S}}_k$, we can execute the identically-named command with the same parameterization in GMS, leading to a state in which all queries are answered in the same way.

Thus, $\gamma_k \vdash q(p)$ if and only if $\gamma^{\mathcal{S}}_k \vdash q^{\mathcal{S}}(p)$. Therefore, we have proven property (2) for state-matching implementations, and proven that the implementation $\sigma^{\mathcal{S}}$ is a state-matching implementation. □