TECHNIQUES FOR APPLICATION-AWARE SUITABILITY ANALYSIS OF ACCESS CONTROL SYSTEMS

by

William C. Garrison III

B.Sc. in Computer Science, Clarkson University, 2009

Submitted to the Graduate Faculty of

the Kenneth P. Dietrich School of Arts and Sciences in partial

fulfillment

of the requirements for the degree of

Doctor of Philosophy

University of Pittsburgh

2015

UNIVERSITY OF PITTSBURGH

KENNETH P. DIETRICH SCHOOL OF ARTS AND SCIENCES

This dissertation was presented

by

William C. Garrison III

It was defended on

November 13, 2015

and approved by

Adam J. Lee, Associate Professor, University of Pittsburgh

Panos K. Chrysanthis, Professor, University of Pittsburgh

Taieb Znati, Professor, University of Pittsburgh

Lenore Zuck, Associate Professor, University of Illinois at Chicago

Dissertation Director: Adam J. Lee, Associate Professor, University of Pittsburgh

TECHNIQUES FOR APPLICATION-AWARE SUITABILITY ANALYSIS OF ACCESS CONTROL SYSTEMS

William C. Garrison III, PhD

University of Pittsburgh, 2015

Access control, the process of selectively restricting access to a set of resources, is so fundamental to computer security that it has been called the field's traditional center of gravity. As such, a wide variety of systems have been proposed for representing, managing, and enforcing access control policies. Prior work on evaluating access control systems has primarily relied on relative expressiveness analysis, which proves that one system has greater capabilities than another. Although expressiveness is a meaningful basis for comparing access control systems, it does not consider the application in which the system will be deployed. Furthermore, expressiveness is not necessarily a useful way to rank systems; if two systems are expressive *enough* for a given application, little benefit is derived from choosing the one that has greater expressiveness. On the contrary, many of the concerns that arise when choosing an access control system can be negatively impacted by additional expressiveness: a system that is too complex is often harder to specify policies in, less efficient, or harder to reason about from the perspective of security guarantees.

To address these shortcomings, we propose the *access control suitability analysis problem*, and present a series of techniques for solving it. Suitability analysis evaluates access control systems against the specific demands of the application within which they will be used, and considers a wide range of both expressiveness and ordered cost metrics. To conduct suitability analysis, we present a two-phase framework consisting of formal reductions for proving qualitative suitability and simulation techniques for evaluating quantitative suitability. In support of this framework we present a fine-grained lattice of reduction properties, as

well as Portuno, a flexible simulation engine for conducting cost analysis of access control systems. We evaluate our framework formally, by proving that it satisfies a series of technical requirements, and practically, by presenting several case studies demonstrating its use in conducting analysis in realistic scenarios.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ALGORITHMS

1.0 INTRODUCTION

Access control, the process of selectively restricting access to a set of resources, is one of the most fundamental aspects of computer security. In essence, an access control system consists of a *policy* describing which active entities, or *subjects*, can access which resources, or *objects*, and in which ways (i.e., using which *rights*). This policy is then enforced by a trusted *reference monitor*. Physical access control systems have existed throughout history. Gates have been guarded by humans playing reference monitor, and later lock/key systems delegated this authority to a trusted mechanism. The development of digital equivalents to these physical systems has been of great concern since the advent of timesharing. Access control has been called the traditional center of gravity of computer security [5], due to the reliance that most other security mechanisms have on the basic premise of restricting access to various resources.

As such a longstanding, important area in computer security, a wide variety of systems have been proposed for representing, managing, and enforcing access control policies (e.g., [9, 10, 13, 22, 23, 36, 39, 57, 58, 67, 101–104]). A class of systems that have grown to epitomize the field utilize an *access matrix* (see, e.g., [57]), a large table with a row for each subject and a column for each object. Cell $\langle s, o \rangle$ in this table is populated with the rights that subject s can employ in accessing o. In practice, each object o can be stored alongside its column from the access matrix (i.e., its *access control list*), possibly using a sparse representation where appropriate. Alternatively, each subject s can manage a (tamper-proof) copy of its row from the matrix (i.e., its *capability list*). This introduces a trade-off in efficiency: both options are able to efficiently answer single requests (e.g., should subject s be granted right r to object o?), but for bulk queries (e.g., what subjects can access object o?), their behavior is quite different, since access control lists are indexed by object, and capability lists are indexed by

subject. In addition, since a subject manages their own capability list, revocation in these environments becomes non-trivial [21].

Further developments in access control models incorporated various features to enable a wider range of policy management techniques. The Bell–LaPadula model [9, 10] enables a form of mandatory access control, where subjects are assigned clearances, and no subject can be granted access to information classified higher than their own clearance. Role-based access control [36, 103, 104] introduces a level of indirection in the policy between subjects and objects. Roles are sets of permissions that are commonly assigned together to users with a particular job description, allowing them to be assigned and revoked as a unit. Trust management [13, 67] introduced the concept of delegating trust, allowing the expression of access conditions such as, "allow access to object o only for subjects that authority a grants credential c." Attribute-based access control [68, 99, 116, 118] assigns permissions based on attributes assigned to subjects and objects. Relationship-based access control [22, 23, 39] allows subjects in a social networking environment to specify permissions based on their relationships to the requesting subjects. This is hardly an expansive list of access control models and systems, but even this small sample demonstrates the wide range of techniques for representing and managing policies.

As we have noted, even the decision between access control lists and capability lists carries practical consequences in terms of the costs of various operations. It stands to reason that this issue is exacerbated when considering the completely unwieldy decision between the dozens of different access control models proposed in the literature. Besides the efficiency of operations, which access control system should be used in a particular scenario may depend on the ease of proving a policy's correctness, the effort required to specify the policy, and the flexibility the system has in adjusting when the policy must evolve. When considering this wide range of issues, and the full range of access control options available, it becomes clear that the act of choosing which access control system to use in a given scenario is impossible without analysis techniques to evaluate them.

This need arises in a number of situations. First, consider the task of deploying a system (software and/or hardware) that requires an access control solution. We refer to such a system as an access control *application*. An access control application can range from a

family's personal computer that requires separation of parents' files from their children's, to a company network housing millions of resources distributed between thousands of servers accessible by tens of thousands of employees and millions of customers. Choosing an access control system can be a decision made at deployment time (e.g., "which software package should I install?") or development time (e.g., "which access control model should I use in this program?"). Furthermore, it is common for an application's needs to evolve over time, and thus administrators of an access control application are often continuously deciding whether to migrate to another solution. Finally, the techniques used in evaluating access control systems can guide the development of new access control systems; it is desirable to show that a newly-developed system is better-suited to certain scenarios than the existing alternatives.

As such, there has been a vast body of work on the formal analysis of access control systems (e.g., [2, 3, 11, 21, 41, 77, 85, 89, 101, 105–107, 113, 114]). The first step in these works is developing a mathematical representation of access control systems; the most common such formalism is the *state machine*. In the state machine representation of an access control system, the set of states represents the possible organizations of the system's data structures (i.e., all possible static policies). In order to interpret these states, a state machine must also include a procedure for determining which authorization requests are granted in a given state (in some cases, additional queries are also included). Finally, a set of commands allows the machine to transition between states (i.e., these commands formalize the ways in which the policy can be manipulated by the system's administrators).

The state machine representation of an access control system enables a powerful means of comparing two or more systems: *relative expressiveness analysis*. This is conducted by constructing a mapping between two access control state machines $\mathcal{S}$ and $\mathcal{T}$ called a *reduction*. A reduction shows how $\mathcal{T}$ can represent each of the policies that $\mathcal{S}$ can represent, and in doing so proves that $\mathcal{T}$ is at least as expressive as $\mathcal{S}$. This has been used in the literature to show that $\mathcal{T}$ can be used in place of $\mathcal{S}$ [85, 89, 101, 107].

Although expressiveness is a meaningful basis for comparing access control systems, it is not without its shortcomings. It is well known that access control is not an area in which one size fits all [63]. The best access control system for the example applications described above, a family computer and the corporate network, are overwhelmingly unlikely to be the

same system. Yet, expressiveness analysis can only reveal that $\mathcal{T}$ can do everything $\mathcal{S}$ can do. This analysis result is independent of any particular usage, and as such cannot give full consideration to the needs of any particular application.

Furthermore, expressiveness is not necessarily a useful way to rank access control systems. When deciding between two systems that are both capable of representing the policies that the application requires (i.e., two systems that are expressive *enough*), little benefit is derived from choosing the one that has greater absolute expressiveness. Many of the concerns detailed above are unaddressed, such as efficiency, flexibility, and ease of policy specification. In fact, additional expressiveness can often negatively impact these other measures: a system that is too complex is often harder to specify policies in, and a system with more features is often harder to reason about from the security guarantees standpoint.

In this dissertation, we show that relative expressiveness analysis is not sufficient for evaluating access control systems relative to how they will be used in practice. We then support the argument that a better type of analysis is possible by justifying the following thesis statement:

> *We can develop techniques to evaluate access control systems against the specific demands of the application in which they will be used, while considering a wide range of both expressiveness and ordered cost metrics.*

To support this thesis, we will first discuss in detail why expressiveness analysis is insufficient for the analysis goals set forth in this thesis statement. In doing so, we make the following contributions.

- We present the first discussion of the shortcomings of current, application-agnostic access control evaluation methods.
- We discuss the key analysis questions that need to be considered in order to conduct a more expansive analysis of access control systems, which we call suitability analysis. We propose a general workflow for conducting suitability analysis, the components of which will be fleshed out in detail in later chapters.
- We identify the key challenges in instantiating a full suitability analysis framework. Of particular note, we motivate the development of a formalism for *access control workloads*,

a novel concept that we introduce to capture an application's specific access control needs. Furthermore, we describe the range of access control cost measures that are desirable to consider, and thus that suitability analysis frameworks must be able to represent and utilize when evaluating access control systems.

Next, we use these motivations to formally define the suitability analysis problem, which formalizes the type of analysis question referred to in our thesis. Further, we propose the first suitability analysis framework to conduct such analyses. We accomplish this through the following contributions.

- We formalize the access control suitability analysis problem and articulate a set of requirements that should be satisfied by frameworks that aim to provide solutions to it.
- We develop the first two-phase suitability analysis framework. Utilizing this framework first establishes whether candidate systems are expressive enough to safely implement the functionality of the workload via reduction. It then utilizes a constrained, actor-based representation of the workload to sample usage patterns and drive a simulation-based cost analysis to explore the expected costs of deployment.
- We evaluate our framework *formally* by proving that our simulation procedure is fixed-parameter tractable, and *practically* via a case study demonstrating how our framework can be effectively used to gain insight into a realistic scenario based on an academic conference management system.

Armed with the context of the general suitability analysis framework, we describe in detail Portuno, a Java-based simulation engine for conducting the cost analysis portion of suitability analysis.

- To enable the generation of traces of representative workload usage, we present a way to represent constrained, actor-based workloads within Portuno, allowing analysts to encode the behavior of the entities that will use the access control application.
- We present an extensible, multi-component measurement system for Portuno that can measure the wide variety of costs incurred by each candidate access control system when executing the generated traces.

- Portuno enables analysts to explore trends in costs using the Monte Carlo driver, or to determine a particular expected cost to within a specific confidence interval using the confidence-bounding driver.
- To demonstrate the usage of Portuno, we discuss the aforementioned conference management case study with details regarding its implementation within the Portuno framework.

To demonstrate further that suitability analysis enables application-sensitive evaluations of access control systems between which the previous work is unable to distinguish, we present a broader case study into the effectiveness of classic (so-called dissemination-centric) access control systems when applied to group-centric workloads. In doing so, we answer the following open questions [71, 72] from the literature.

- Which systems based on the group-centric information sharing (g-SIS) models can be *safely* implemented within, or emulated by, dissemination-centric access control systems?
- How *strong* are the security properties that can be guaranteed by dissemination-centric systems when implementing workloads based on the g-SIS models?
- How *efficiently* can dissemination-centric systems implement workloads based on the g-SIS models?
- What practically interesting instantiations of the g-SIS models *cannot* by safely and efficiently implemented by dissemination-centric access control systems?

To demonstrate our suitability analysis framework's ability to effectively solve the desired analysis problem in other security domains, we present an additional case study that explores a fundamentally different scenario. In particular, we investigate the suitability of constructions based on identity-based encryption schemes for enforcing dynamic access controls on an untrusted cloud storage provider. Through this analysis, we use our suitability analysis framework to make the following contributions.

- We develop constructions using either IBE/IBS or public-key cryptographic paradigms to enable outsourced role-based access controls and prove that these constructions correctly implement the RBAC_0 specification. In an effort to lower-bound deployment costs, we make design choices in our constructions that emphasize efficiency over the strongest possible security.

- We use real-world RBAC datasets and stochastic models of administrative behavior to quantify the costs of using our constructions in the desired application. Our findings demonstrate scenarios in which IBE/IBS and public key cryptography are effective means of implementing RBAC access controls, and many situations in which severe overheads are incurred through the use of these techniques. We show that the inflection points in performance are a function of organization size, role density, and administrative operational mix.
- Our findings provide a number of insights into promising future research directions that could lead to better support for cryptographic access controls in dynamic environments.

We then note the flexibility of our framework in terms of cost metrics, and the relative inflexibility of the existing notions of expressiveness. Although many types of expressiveness reductions have been proposed, they are difficult to compare, and there is little guidance for which notion of reduction to use for a particular analysis. To allow a similarly wide range of expressiveness metrics as we do cost metrics, we define a minimal set of properties for an access control reduction to be used in practical scenarios and then define a wide range of properties spread across several dimensions that can be enforced on top of this minimum definition. These properties define a taxonomy that can be used to compare existing types of expressiveness reduction as well as craft new types of expressiveness reduction. In proposing this properties lattice, we make the following contributions.

- We propose a general definition of an *implementable* access control mapping that is broad enough to encompass much of the wide range of existing access control reductions, yet precise enough to guarantee implementability. Intuitively, an *implementable* reduction from $\mathcal{S}$ to $\mathcal{T}$ shows that $\mathcal{T}$ can accomplish everything $\mathcal{S}$ can, and deterministically shows *how*.
- We decompose and expand upon the properties enforced by various access control reductions from the literature, forming a lattice relating the range of access control reductions to one another. This lattice allows us to formally compare the guarantees offered by existing notions of access control reduction (many of which were not formerly known to be comparable) and points to unexplored combinations of properties that can

yield different expressiveness results.

- We construct formal proofs positioning existing notions of access control reduction within our lattice of reduction properties, including a comparative discussion of reductions that previously seemed incomparable. We thus systematize the formal relationships between previously-published reductions, allowing reconciliation of previously disparate expressiveness knowledge and enabling analysts to make informed choices about which notion of reduction to use for their analyses.
- We observe that many of the dimensions upon which our reduction property lattice is built have implications for the use of reductions for satisfying real-world requirements using existing access control systems (e.g., required storage, whether data structures must be locked for concurrent usage). Thus, in addition to positioning existing notions of reduction within our lattice of properties, we assist in creating new notions of reduction by selecting the properties that should be enforced in an expressiveness analysis based upon the scenario in which an eventual access control deployment will occur. To this end, we discuss in detail various interactions between reduction properties, the results of enforcing different properties, and how a specific deployment scenario dictates which properties are relevant.

Finally, we further support our lattice of reduction properties as a fine-grained tool for choosing a type of reduction for an analysis. We conduct a case study considering the state-matching reduction [113, 114], a recent type of expressiveness reduction that has all but superceded prior notions due to its preservation of a very expressive analysis question. We compare the security analysis questions preserved by both the state-matching reduction and several alternative forms of reduction that we introduce. In this case study, we make the following contributions.

- We present several alternative reductions to the state-matching reduction represented within the framework of implementable mappings and reduction properties lattice. These variants highlight design decisions that were made by Tripunitara and Li with the state-matching reduction that are not explicitly discussed, that are newly apparent given this lattice of reduction properties.

- We show that each of our variants of the state-matching reduction can be shown to satisfy the characteristic that was previously thought to be the defining strength of the original: its preservation of compositional security properties. We propose an accompanying variant on *strong security preservation* for each of our alternative reductions, thus showing what it means to preserve compositional security analysis instances within the context of each. This shows the relative imprecision of using preservation of security questions to evaluate a notion of reduction, and shows that our properties lattice is able to more precisely separate expressiveness metrics.
- To show that these separations are meaningful, we investigate the effects on expressiveness results of different reductions that are equally capable of preserving the same security questions. We show that considering a slightly different notion of reduction can yield vastly different expressiveness results, even if those perturbations that do not alter the security analysis questions that are preserved. In particular, we show that within the context of the alternative reductions to the state-matching reduction (which also preserve compositional security), several main conclusions proved using the original are false.
- We identify the reduction properties that seem to have the greatest impact on expressiveness results based on our case study and prior work. That is, we identify those dimensions of properties whose values have the greatest effect on whether a particular reduction is possible or not. We discuss how each changes the meaning of an expressiveness result from the perspective of the application, highlighting the application-based issues that are ignored entirely when evaluating the state-matching reduction in an application-agnostic way (i.e., considering only preserved security questions).

The remainder of this dissertation is organized as follows. In Chapter 2, we describe in detail the history of formal analysis of access control systems, particularly relative expressiveness analysis, to give context for the rest of the dissertation. In Chapter 3, we justify the shortcomings inherent in relying on expressiveness analysis for choosing an access control system for an application, and propose a general workflow for conducting a more appropriate analysis. We then formally define the suitability analysis problem in Chapter 4 and propose a more formal framework for conducting suitability analysis. In Chapter 5, we present Portuno, a simulation engine that enables the cost analysis phase of suitability analysis. In Chapters 6

and 7, we present case studies, investigating the suitability of dissemination-centric access control systems for group-centric sharing workloads, and the suitability of identity-based encryption for enforcing cryptographic access controls in cloud storage systems, respectively. To enable a wider range of expressiveness metrics, in Chapter 8 we propose the notion of implementable access control mapping and a set of reduction properties that can be enforced atop it. In Chapter 9, we conduct a case study on the state-matching reduction and prove that using our reduction properties is more precise and fine-grained than relying on preservation of security questions, in ways that have a great impact on expressiveness results. Finally, we present our conclusions and directions for future work in Chapter 10.

2.0 BACKGROUND AND RELATED WORK

In this chapter, we provide necessary background for context in interpreting the remainder of this dissertation. Section 2.1 presents the prerequisite formalism for representing access control systems, based on state machines. This formalism enables relative expressiveness, the primary form of access control evaluation described in the literature up to now. This is accomplished through the construction of reductions between two systems. In Section 2.2, we detail the history of access control expressiveness analysis, focusing on the ways in which the proposed expressiveness reductions have gotten more and more rigorous.

2.1 MODELING ACCESS CONTROL

An access control *model* formalizes the way in which a class of access control systems will store and interpret information to make access control decisions. Its data structures are formalized as a set of access control *states*, and its methods for determining whether to allow or deny accesses as a set of *authorization requests*. The value of all requests in a state (whether they are allowed or denied) defines the access control policy, or *theory*, being enforced in that state.

Definition 1 (Access Control Model)**.** An *access control model* is defined as $\mathcal{M} = \langle \Gamma, \mathcal{R} \rangle$, where Γ is the set of states and $\mathcal{R}$ is the set of authorization requests, where each request $r \in \mathcal{R}$ is a function $\Gamma \to \{\text{TRUE}, \text{FALSE}\}$. The entailment ($\vdash$) of a request is defined as $\gamma \vdash r \triangleq r(\gamma) = \text{TRUE}$. $\Diamond$

As an example, we consider the model to which RBAC_0 [103, 104], the simplest form of

role-based access control, belongs.

Example 1. The RBAC_0 model is defined as $\langle \Gamma^{\mathsf{R}}, \mathcal{R}^{\mathsf{R}} \rangle$. Its states, Γ^{R}, are defined by the sets $\langle U, R, P, UR, PA \rangle$, where:

- U is the set of users
- R is the set of roles
- P is the set of permissions
- $UR \subseteq U \times R$ is the user-role relation, which describes users' membership in roles
- $PA \subseteq R \times P$ is the role-permission relation, which describes permissions' assignment to roles

A request in $\mathcal{R}^{\mathsf{R}}$ is of the form $\langle u, p \rangle$, in which $u \in U$ requests access to $p \in P$, and is defined as TRUE in $\gamma = \langle U^\gamma, R^\gamma, P^\gamma, UR^\gamma, PA^\gamma \rangle \in \Gamma^{\mathsf{R}}$ when $\exists x \in R : \langle u, x \rangle \in UR^\gamma \wedge \langle x, p \rangle \in PA^\gamma$ and FALSE otherwise. ◊

An access control *system* expands on a model by providing methods of transforming the current state and additional methods of querying states. These additional queries allow the user to ask additional boolean questions of the system for which a value of TRUE does not indicate an authorization was granted.

Definition 2 (Access Control System)**.** Given access control model $\mathcal{M} = \langle \Gamma, \mathcal{R} \rangle$, an *access control system* within $\mathcal{M}$ is a state transition system, $\mathcal{S} = \langle \Gamma, \Psi, Q \rangle$, where Ψ is the set of commands, each command $\psi \in \Psi$ is a function $\Gamma \to \Gamma$, $Q \supseteq \mathcal{R}$ is the set of queries, and each query $q \in Q$ is a function $\Gamma \to \{\text{TRUE}, \text{FALSE}\}$. ◊

We use the notation $next(\gamma, \psi)$ to denote the state resulting from executing ψ in γ (that is, $\psi(\gamma)$), and $terminal(\gamma, \psi_1 \circ \ldots \circ \psi_n)$ to denote the final state produced by repeatedly applying *next* to the commands $\psi_1, \ldots, \psi_n$ starting from state γ: $next(\ldots next(\gamma, \psi_1), \ldots, \psi_n)$.

The RBAC_0 model from Example 1 encompasses the following RBAC_0 system.

Example 2. Given the RBAC_0 model $\langle \Gamma^{\mathsf{R}}, \mathcal{R}^{\mathsf{R}} \rangle$, the RBAC_0 system is defined as $\langle \Gamma^{\mathsf{R}}, \Psi^{\mathsf{R}}, Q^{\mathsf{R}} \rangle$. Its commands, Ψ^{R}, include all of the following forms, which follow the convention that the first parameter is the executing entity.

- addUser(a, u): $U \leftarrow U \cup \{u\}$

- removeUser(a, u): $U \leftarrow U \setminus \{u\}$
- addRole(a, r): $R \leftarrow R \cup \{r\}$
- removeRole(a, r): $R \leftarrow R \setminus \{r\}$
- addPermission(u, p): $P \leftarrow P \cup \{p\}$
- removePermission(u, p): $P \leftarrow P \setminus \{p\}$
- assignUser(a, u, r): $UR \leftarrow UR \cup \{\langle u, r\rangle\}$
- revokeUser(a, u, r): $UR \leftarrow UR \setminus \{\langle u, r\rangle\}$
- assignPermission(a, r, p): $PA \leftarrow PA \cup \{\langle r, p\rangle\}$
- revokePermission(a, r, p): $PA \leftarrow PA \setminus \{\langle r, p\rangle\}$

Its queries Q^{R} include the requests $\mathcal{R}^{\mathsf{R}}$ as well as "user in role?" queries of the form $\langle u, r\rangle$, which are TRUE if and only if $\langle u, r\rangle \in UR$. $\Diamond$

This state-machine representation of access control enables much of the formal study in the literature. In the next section, we overview such formal analysis techniques.

2.2 ACCESS CONTROL ANALYSIS

The formal study of access control systems began with a seminal paper by Harrison, Ruzzo, and Ullman [57]. This paper formalized the general access matrix model that has since been called the HRU model. States in this model are represented by a set, R, of access rights; a set, S, of subjects; a set, O, of objects; and an access matrix, P, with a row for each subject $s \in S$ and a column for each object $o \in O$. Each cell $P(s, o)$ contains a subset of R, designating which access rights to o are possessed by s. Each system based on the HRU model defines a series of commands, each of which accepts a list of parameters, checks a conjunction of preconditions (rights that must be in certain cells of P for the command to be executed), and executes a sequence of *primitive operations*. The primitive operations in the HRU model include entering/removing rights in P and creating/destroying subjects/objects.

Harrison, Ruzzo, and Ullman went on to consider the analysis question of *simple safety* (presented here in simplified form as per Li, Mitchell, and Winsborough [75]):

Definition 3 (Simple Safety Analysis)**.** Given access control system $\mathcal{S} = \langle \Gamma, \Psi, Q \rangle$ based on model $\mathcal{M} = \langle \Gamma, \mathcal{R} \subset Q \rangle$, a *simple safety* question is one of the following form:

> Does there exist a state γ_1 reachable from state $\gamma_0 \in \Gamma$ via commands Ψ in which request $r \in \mathcal{R}$ is granted? $\Diamond$

The work concluded that simple safety was undecidable in the general HRU model, a fact that was proved using a construction whereby the potentially-complex commands allowed in an HRU system could be used to simulate general computation (with the detection of rights leakage reducing to the halting problem). However, they also showed that simple safety was decidable if the commands were restricted to executing at most one primitive operation each.

Shortly after, Lipton and Snyder [80] analyzed the simpler *Take-Grant* access control model. States in this system are typically visualized as directed graphs, where subjects and objects are vertices, and edges are labeled with the rights that the origin vertex has over the destination vertex. If a subject has the *take* right over an object, the subject can receive a copy of any permission that the object has. If a subject has the *grant* right over an object, the subject can give the object a copy of any permission that the subject has. Lipton and Snyder showed that, in this system, simple safety was not only decidable, but decidable in linear time.

These two results from the latter half of the 1970s have epitomized the notion that the most expressive system is not always the right choice—that restricting our system can yield higher efficiency and greater ease in solving relevant security problems. However, this high-level insight seemed to have little effect on the direction of research in access control evaluation; as we will see, expressiveness continued to be the primary focus of such works for decades.

From this point, there have been a number of results investigating the *relative expressiveness* of various access control systems. A relative expressiveness result is typically of the form, "System $\mathcal{T}$ is at least as expressive as system $\mathcal{S}$." Informally, this means that $\mathcal{T}$ can emulate the behavior of $\mathcal{S}$, and is meant to assure us that we can use $\mathcal{T}$ in any scenario in which $\mathcal{S}$ can be used. The primary way of proving such a statement is by constructing a *mapping* from $\mathcal{S}$ to $\mathcal{T}$, and writing a formal proof that the mapping satisfies the requirements

for a particular form of *reduction*. Various notions of reduction in the literature have enforced very different properties (e.g., What type of behavior must be simulated? How closely must $\mathcal{T}$ represent the information in $\mathcal{S}$?), and as a result have been used to prove qualitatively different types of expressiveness results.

The works of Sandhu, Ganta, Munawer, and Osborn [85, 89, 101, 105–107] include some of the earliest access control reductions. In these works, a reduction from $\mathcal{S}$ to $\mathcal{T}$ must show that a permission can be granted in $\mathcal{S}$ if and only if the corresponding permission can also be granted in $\mathcal{T}$. This concept is expressed as follows.

Definition 4 (SMG Reduction)**.** An access control model $\mathcal{N}$ *subsumes* model $\mathcal{M}$ if, for every system $\mathcal{S}$ which can be specified in $\mathcal{M}$, we can construct a system $\mathcal{T}$ in model $\mathcal{N}$ and a mapping σ such that subject s can have access r to object o in $\mathcal{S}$ if and only if subject $\sigma(s)$ can have access $\sigma(r)$ to object $\sigma(o)$ in $\mathcal{T}$. ◊

A variety of relative expressiveness results have been proved using this notion of reduction. The Schematic Protection Model [100] was shown to emulate the Bell–LaPadula multilevel security model [9, 10], the take-grant model [80], and grammatical protection systems [101]. The typed access matrix model (TAM) [102] is shown to be as expressive as the *augmented* typed access matrix model (ATAM) [105], which allows testing for the *absence* of rights), and thus it is concluded that the testing for absence of rights is theoretically unnecessary [105]. In support of role-based access control, it was shown to be capable of emulating strict discretionary access control [107], the augmented typed access matrix [85], and mandatory access control [89].

No formal properties beyond Definition 4 are enforced by these early reductions, though in some cases additional properties become part of the *de facto* definition of reduction. For instance, while there is no requirement for $\mathcal{T}$ to have a state equivalent to each $\mathcal{S}$ state (merely for $\mathcal{T}$ to be able to grant each access that $\mathcal{S}$ does, in some state), the example reductions all include methods for mapping each $\mathcal{S}$ state to a $\mathcal{T}$ state (as this is the simplest way to show the required property). In addition, although the definition does not prohibit the use of an unbounded number of $\mathcal{T}$ commands to simulate a single $\mathcal{S}$ command, Sandhu and Munawer [107] only use reductions in which an $\mathcal{S}$ command is simulated using a constant

number of $\mathcal{T}$ commands. We note that Sandhu recognized this underspecification [101], stating that:

> *The actual constructions given... establish equivalence in a stronger sense than [Definition 4]. It is beyond the scope of this paper to give a formal definition of "stronger" in this context. The intuition is that "stronger" means "behavioral equivalence." That is, every state transition in $\mathcal{S}$, say from state α to state β, can be mimicked by one or more state transitions in $\mathcal{T}$ which applied to state $\sigma(\alpha)$ result in state $\sigma(\beta)$.*

Ganta's PhD dissertation [41] attempts to formalize a more rigorous notion of expressiveness reduction. In his form of reduction, the state correspondence is explicit, requiring that each state in $\mathcal{S}$ have a corresponding state in $\mathcal{T}$ that grants all the same accesses (at least, all those that exist in $\mathcal{S}$—those that exist in $\mathcal{T}$ but not in $\mathcal{S}$ are unconstrained). In addition, to ensure that $\mathcal{T}$ cannot grant accesses that $\mathcal{S}$ cannot, any state that can be entered in $\mathcal{T}$ must also have a corresponding reachable state in $\mathcal{S}$. Finally, to ensure accesses in $\mathcal{T}$ cannot be combined in ways that cannot occur in $\mathcal{S}$, the following restriction is made: when simulating a $\mathcal{T}$ command in $\mathcal{S}$, multiple commands may be used, but each state along the way must allow *either* a subset of the accesses of the start state or a subset of the accesses of the end state. Thus, no two accesses can be allowed in the same state in $\mathcal{T}$ that are not allowed in a single state in $\mathcal{S}$.

Ammann, Lipton, and Sandhu [2,3] took a different (and much more strict) approach to more rigorously defining a reduction. First, they describe a strict state correspondence that requires $\mathcal{T}$ to represent its states with the same sets and relations as $\mathcal{S}$, and for these sets to have identical contents in corresponding $\mathcal{T}$ and $\mathcal{S}$ states.

Definition 5 (ALS State Correspondence)**.** A state in system $\mathcal{S}$, an original system, and a state in system $\mathcal{T}$, a simulating system, *correspond* if and only if the graph defining the state in $\mathcal{S}$ is identical to the subgraph obtained by taking the state in $\mathcal{T}$ and discarding all nodes and edges of any type not defined in $\mathcal{S}$. ◊

In other words, $\mathcal{T}$ cannot include additional elements of any type that $\mathcal{S}$ uses (although additional, distinct types may be stored). For example, one could simulate the state $\{U = \{a, b\}, V = \{c\}\}$ with state $\{U = \{a, b\}, V = \{c\}, W = \{\langle a, d\rangle, \langle b, d\rangle\}\}$, but not with

$\{U = \{a, b\}, V = \{c, d\}\}$. Given this notion of state correspondence, the ALS reduction must shows that $\mathcal{T}$ can reach a state corresponding to each reachable $\mathcal{S}$ state, and cannot reach any state that does not have a reachable corresponding state in $\mathcal{S}$.

Definition 6 (ALS Reduction)**.** System $\mathcal{T}$ *emulates* system $\mathcal{S}$ if and only if the following conditions hold:

1. For every state s reachable by $\mathcal{S}$, there exists some state t reachable by $\mathcal{T}$ such that s and t correspond.
2. For every state t reachable by $\mathcal{T}$, either:
   a. The state s that corresponds to t is reachable by system $\mathcal{S}$, or
   b. There exists some successor state t' of t such that the state s' that corresponds to t' is reachable by system $\mathcal{S}$. ◊

This strict notion of reduction is used to show that monotonic, multi-parent systems are more expressive than monotonic, single-parent systems (e.g., there are monotonic multi-parent systems that cannot be simulated by any monotonic single-parent system).

Chander, Dean, and Mitchell [21] restrict the definition of reduction in yet a different way. Rather than force a more strict state correspondence (the *static* portion of the reduction), they more tightly restrict the way the reduction handles the system as it executes (i.e., the command mapping). In these reductions, the state correspondence is comparatively lax: to simulate an $\mathcal{S}$ state, a $\mathcal{T}$ state must allow and deny all the same authorization requests as its corresponding $\mathcal{S}$ state. Additional requests can exist in $\mathcal{T}$ and are unconstrained, but all requests corresponding to those in $\mathcal{S}$ must have the same value in corresponding states.

Definition 7 (CDM State Access-Containment)**.** Given access control models $\mathcal{M}_1$ and $\mathcal{M}_2$, and states γ_1 and γ_2, we say that γ_1 is access-contained in γ_2 if for any subject s in γ_1, the access decisions $\gamma_1 \vdash s \rightarrow (o, r)$ and $\gamma_2 \vdash s \rightarrow (o, r)$ yield the same result. ◊

However, the process for *simulating* an $\mathcal{S}$ command using $\mathcal{T}$ commands must be independent of the state: it cannot, e.g., execute a $\mathcal{T}$ command for each user, or otherwise inspect the state when determining what commands should be executed.

Definition 8 (CDM Weak Reduction)**.** Given systems $\mathcal{S}$ and $\mathcal{T}$ within models $\mathcal{M}_1$ and $\mathcal{M}_2$, respectively, $\mathcal{S}$ is *weakly simulated* by $\mathcal{T}$ if the access containment relation between $\mathcal{S}$ and $\mathcal{T}$

is a many-step simulation (i.e., if γ_1 is access-contained in γ_2 and γ_1' is reachable from γ_1 via action α, then there exists γ_2' such that reachable from γ_2 via sequence of actions $f(\alpha)$, and γ_1' is access-contained in γ_2'). ◊

In addition, in the strong form of reduction, each $\mathcal{S}$ command must be simulated with a single $\mathcal{T}$ command.

Definition 9 (CDM Strong Reduction)**.** Given systems $\mathcal{S}$ and $\mathcal{T}$ within models $\mathcal{M}_1$ and $\mathcal{M}_2$, respectively, $\mathcal{S}$ is *strongly simulated* by $\mathcal{T}$ if the access containment relation between $\mathcal{S}$ and $\mathcal{T}$ is a one-step simulation (i.e., if γ_1 is access-contained in γ_2 and γ_1' is reachable from γ_1 via action α, then there exists γ_2' such that reachable from γ_2 via single action $f(\alpha)$, and γ_1' is access-contained in γ_2'). ◊

Chander, Dean, and Mitchell use these definitions to compare the expressiveness of access control lists, trust management, and two forms of capability systems (each of which is considered in forms with and without revocation and delegation). Of particular note, capabilities in the form of unforgeable bit strings are shown to be as expressive as the access matrix when not considering revocation, but strictly less expressive when revocation is considered.

Tripunitara and Li [113,114] noted that the existing notions of reduction did not correspond directly to any particular safety analysis questions, and thus a reduction of any of these types does not make any particular safety guarantees. They formalize *compositional security analysis* (intuitively, determining whether a certain set of access control queries will always, never, or sometimes become true in any reachable state), which is a generalization of simple safety analysis (Definition 3).

Definition 10 (Compositional Security Analysis)**.** Given an access control system $\langle \Gamma, \Psi, Q \rangle$, a *compositional security analysis instance* has the form $\langle \gamma, \phi, \psi, \Pi \rangle$, where $\gamma \in \Gamma$ is a state, ϕ is a propositional formula over Q, $\psi \in \Psi$ is a state-transition rule, and $\Pi \in \{\forall, \exists\}$ is a quantifier.

An instance $\langle \gamma, \phi, \psi, \exists \rangle$ is said to be *existential*: it asks whether there exists state γ_1 such that $\gamma \overset{*}{\mapsto}_{\psi} \gamma_1$ and $\gamma_1 \vdash \phi$.

An instance $\langle \gamma, \phi, \psi, \forall \rangle$ is said to be *universal*: it asks whether for every state γ_1 such

that $\gamma \overset{*}{\mapsto}_{\psi} \gamma_1$, $\gamma_1 \vdash \phi$. $\Diamond$

They then present a notion of reduction tailor-made to preserve these types of analysis questions. Their reduction, called the state-matching reduction, considers a broader range of queries than only authorization requests, placing the strictness of its state correspondence somewhere between the work of Ammann, Lipton, and Sandhu and that of Chander, Dean, and Mitchell. The state-matching reduction maps each query $q^{\mathcal{S}}$ in $\mathcal{S}$ to a single query $q^{\mathcal{T}}$ in $\mathcal{T}$, and the reduction must determine the value of $q^{\mathcal{S}}$ in any state in $\mathcal{T}$ by checking the value of $q^{\mathcal{T}}$. Finally, reachability constraints ensure that $\mathcal{T}$ can reach a state corresponding to each reachable $\mathcal{S}$ state, and cannot reach any state that does not have a reachable corresponding state in $\mathcal{S}$.

Definition 11 (State-Matching Reduction)**.** Given a mapping from $\mathcal{S}$ to $\mathcal{T}$, $\sigma : (\Gamma^{\mathcal{S}} \times \Psi^{\mathcal{S}}) \cup Q^{\mathcal{S}} \to (\Gamma^{\mathcal{T}} \times \Psi^{\mathcal{T}}) \cup Q^{\mathcal{T}}$, we say that two states $\gamma^{\mathcal{S}}$ and $\gamma^{\mathcal{T}}$ are *equivalent* under the mapping σ when, for every $q^{\mathcal{S}} \in Q^{\mathcal{S}}$, $\gamma^{\mathcal{S}} \vdash q^{\mathcal{S}}$ if and only if $\gamma^{\mathcal{T}} \vdash \sigma\big(q^{\mathcal{S}}\big)$. A mapping σ from $\mathcal{S}$ to $\mathcal{T}$ is said to be a *state-matching reduction* if, for every $\gamma^{\mathcal{S}} \in \Gamma^{\mathcal{S}}$ and every $\psi^{\mathcal{S}} \in \Psi^{\mathcal{S}}$, $\langle \gamma^{\mathcal{T}}, \psi^{\mathcal{T}} \rangle = \sigma\big(\langle \gamma^{\mathcal{S}}, \psi^{\mathcal{S}} \rangle\big)$ has the following two properties:

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

Tripunitara and Li prove that this notion of reduction preserves compositional security analysis instances: that is, if there exists a state-matching reduction from $\mathcal{S}$ to $\mathcal{T}$, then any compositional security analysis instance has the same truth value in both systems. They then use this reduction to prove a series of expressiveness results, including the first formal evidence of a limit to the expressive power of HRU, a limit to the expressive power of a role-based access control system in comparison to a discretionary access control system, and a proof that ATAM is indeed more expressive than TAM. Tripunitara and Li's reductions have since been used to analyze role-based access control [77] and prove that a newly-proposed tag-based access control system is more expressive than SDCO, ARBAC97, and the Bell–LaPadula model [58].

Although there is a wide variety of work in relative expressiveness of access control systems in the literature, there is little consideration for additional metrics beyond this measure of raw capabilities, or for the specific demands that an application will place on these systems. In the remainder of this dissertation, we will develop techniques to fill these voids. Specifically, we will justify the need to consider analysis techniques beyond relative expressiveness (Chapter 3), and define a framework to conduct what we call the suitability analysis (Chapter 4). We will then present techniques within this framework for evaluating the suitability of access control systems for their intended usage, including fine-grained, application-sensitive metrics for both cost analysis (Chapter 5) and expressiveness analysis (Chapter 8). We will also present several case studies (Chapters 6, 7 and 9) to demonstrate the potential that these techniques have to enable analyses that are not possible given only the existing work on application-agnostic relative expressiveness.

3.0 THE NEED TO MOVE BEYOND EXPRESSIVENESS

Access control is an area where one size does *not* fit all. However, previous work in access control analysis has focused primarily on expressiveness as an absolute measure. Thus, in this chapter we discuss and justify a central premise of our thesis statement: a new type of evaluation framework is needed for access control systems, one that is application-aware and considers both expressiveness and costs. To this end, we apply previous work in access control evaluation, as well as lessons learned from evaluation frameworks used in other domains. We describe the analysis components that are required by such a framework, the challenges involved in building it, and the general workflow upon which it can be based.[1]

3.1 INTRODUCTION

In Chapter 2, we have overviewed a wide range of existing work on the formal analysis of access control systems. all of which focused on comparing the *relative expressive power* of two or more access control systems. Although expressive power is an interesting and meaningful basis for comparing access control systems, in practice it is not a sufficient indicator of suitability to any particular application. That is, the knowledge that a $\mathcal{T}$ is more expressive than another system $\mathcal{S}$ provides no assurance that $\mathcal{T}$ is the best access control system 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 $\mathcal{T}$ would in the same situation. Furthermore, 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

[1] The material presented in this chapter was first published as [45].

be evaluated and compared relative to application-aware metrics [63]. 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 systems.

Given these considerations, we discuss the need for an *application-aware* evaluation paradigm for access control systems. Informally, we might state the problem behind application-aware access control evaluation as follows.

> *Given a description of an application's access control needs and a collection of access control systems, which access control system best meets the needs of the application?*

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, application-aware analysis of access control could help developers sort through the myriad available security frameworks (e.g., WPL [82], Spring [111], Shiro [6], etc.) and the multiple access control systems embedded in each.

Despite the vast differences in approach between existing access control expressiveness evaluation and the application-aware evaluation process that we propose (in this chapter and this dissertation as a whole), expressiveness must remain a key component in the process. Specifically, we propose to use expressiveness-based techniques for ensuring that the access control systems considered for an application possess the necessary capabilities. That is, unlike prior work, we evaluate the expressiveness of an access control system only for the purposes of showing it is expressive *enough* to satisfy the needs of a particular application, rather than considering its expressiveness relative to the other candidates. Furthermore, we study techniques used for cost analysis in other domains to formulate requirements for a formal notion of an access control workload and offer guidelines for workload construction and access control cost analysis.

Thus, in this chapter, we motivate the development of an application-aware access control evaluation framework. We first describe motivating scenarios for this type of analysis. We

then give an informal overview of the desired workflow for use in analyzing access control systems with respect to a particular application. We refer to this process as *suitability analysis*. Finally, we consider the challenges that the remainder of this dissertation needs to address in order to develop a more formal framework inspired by this workflow. In particular, we discuss the following key challenges.

- We must develop a mathematical formalism for representing the access-control-relevant needs of an application. This formalism will be used to prove that candidate access control systems are expressive enough to satisfy the application's requirements.
- We must develop a series of analysis techniques for determining the expected costs of using each candidate access control system within the context of the application in question.
- Given the wide range of notions of expressiveness reduction, we need techniques for determining which should be used for the current analysis. In the event that none of the existing reductions is appropriate for the analysis, it should be possible to craft a new notion of reduction specifically to the requirements of the application.
- We consider the task of extending these ideas to develop similar techniques and tools in the broader security domain.

The remainder of this chapter will be structured as follows. In Section 3.2, we describe scenarios that motivate the development of an application-aware access control evaluation framework, and reiterate the reasons why past work in access control expressiveness evaluation is insufficient for these scenarios. In Section 3.3, we describe an overview of the application-aware access control evaluation workflow, which we call suitability analysis. In Section 3.4, we consider the challenges mentioned above that the remaining chapters of this dissertation will address. Finally, we summarize in Section 3.5.

3.2 MOTIVATING SCENARIOS

In this section, we present several scenarios to motivate an application-aware analysis of access control, and discuss why past work is insufficient for addressing these scenarios.

One such scenario is the establishment of new access control *applications*: systems with access control components. For example, suppose a new academic conference is created, and its organizers are considering the various access control needs of their submission and review system. We will refer to this application as the program committee (PC) workload. In this scenario, authors should be able to submit papers and access their submissions and reviews thereof (though the latter may not be available until a particular notification time). The program committee should be able to submit reviews on their assigned submissions, and these reviews should be accessible to the program chair and other assigned reviewers. Discussion about papers in contention may take place among the committee at large, which will require members who have a conflict of interest with this submission to temporarily lose access to this wider discussion. An application-aware access control analysis framework would enable the organizing committee to formalize the requirements of their academic conference workload, as well as the available candidate security packages' mechanisms, and use these formalizations to choose the package that best meets their needs with minimal overheads.

We note that none of the most popular access control systems is an obvious best fit for this scenario. Access matrix systems are capable of enforcing most static policies, but the program committee workload is highly dynamic—both papers and reviews are added over time, access to which must be assigned to potentially large numbers of users. Granting these accesses in an access matrix system must be done individually for each (subject, object) pair. A common solution to this is to utilize the concept of roles or groups, which provide a level of abstraction between users and resources. This allows multiple accesses to be granted at once. However, it is difficult to assign roles in this environment, since each user may have access to a completely different set of resources due to conflicts of interest and different assigned reviews. Relative expressiveness analysis may reveal, e.g., that a particular role-based system is more expressive than a particular access matrix system, but this knowledge does not necessarily assist analysts in using these additional capabilities in this particular application.

Another scenario that highlights the need for application-aware evaluation for access control is re-evaluating the access control needs of existing applications whose requirements or operating assumptions have evolved. For example, two MITRE technical reports [60, 115] have highlighted the fact that the Bell–LaPadula access control system relied upon by the

U.S. Armed Forces is beginning to show signs of age. In particular, [60] cites several examples of improper and unauthorized out-of-band data sharing that have occurred because it is "easier to ask for forgiveness than for permission," given the high delays and human costs associated with utilizing the proper channels. Further, [115] posits that this phenomenon is a byproduct of an increasingly dynamic military that relies on a dizzying array of data sources and ever-changing coalitions, but bases access decisions on a static classification system. In short, the changing military workload has led to confidentiality breaches due to an ill-fitting access control system. An application-aware analysis could help identify the root causes of these failures and assess the utility of alternate access control approaches.

As we noted in Section 2.2, it has been known for quite some time that the most expressive access control system is not always the right choice—that restricting our system can yield higher efficiency and greater ease in solving relevant security problems. The latter was made easily apparent by noting that, in the very expressive HRU model, simple safety in undecidable [57], while in the more restricted take-grant system, simple safety is decidable in linear time [80].

Unfortunately, this means that, despite the wide range of reduction-based techniques for relative expressiveness, the most we can learn from them is an absolute ranking in expressiveness, irrespective of the requirements of the application; none can support a comparison of access control systems with regards to their ability to perform *well* within any particular environment. It is unclear whether there is any benefit to expressiveness beyond "expressive enough," and these existing techniques do not even provide us with a method for ensuring that a system is expressive *enough* for an application, since the application is not considered at all. Thus, it seems clear that a new, application-aware access control evaluation paradigm is needed.

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" [63]. 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 the next section, we describe an informal workflow for conducting

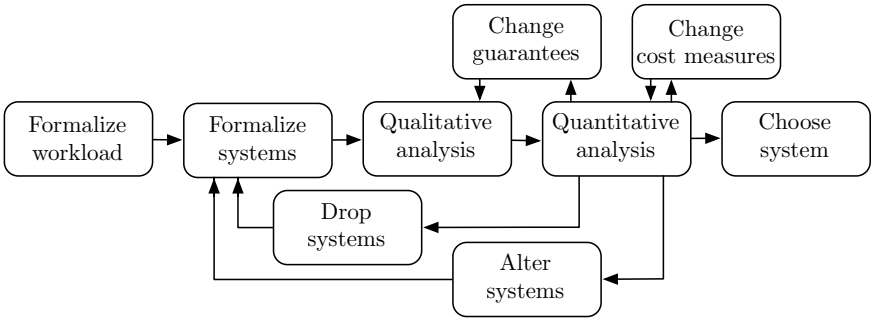


Figure 1: Workflow of an application-aware suitability analysis framework for access control

this type of analysis, in order to assist us in identifying the key challenges that must be addressed in subsequent chapters of this dissertation.

3.3 ANALYSIS WORKFLOW OVERVIEW

Figure 1 presents an overview of the steps involved in the type of application-aware access control evaluation that we envision. First, the application's access-control-relevant properties and requirements are formalized as an access control *workload*. The workload formalism must contain two components. The *operational component* specifies the capabilities that the application requires of its access control solution. This component will allow the analyst to determine whether an access control system is expressive enough for an application. The *invocational component*, then, describes the usage pattern under which the access control system will be expected to perform. This component will allow the analyst to determine the expected cost of using each access control system under the particular usage that is anticipated within the application.

Next, the access control systems to be evaluated are formalized. We will use the well-established state machine formalism described in Definition 2. In qualitative analysis, each access control system (formalized as a state machine) must be shown to be expressive

enough to implement the required capabilities described by the operational component of the workload. This will be accomplished using a variant of access control reduction that maps from a workload to a system (rather than the more traditional system-to-system).

In quantitative analysis, the costs are evaluated of using each (expressive enough) access control system within the context of the application. Cost analysis must be done relative to the usage pattern as specified by the invocational component of the workload. This workload usage can be translated into actions in each candidate system using the reductions constructed during qualitative analysis. The cost of executing the traces in each access control system is evaluated, yielding a cost for each system. Given our application-aware motivation, the cost measures used may vary between applications, from operational costs such as data management overheads, to human-centric costs such as administrative overhead.

At this point, it is possible that the analysis has yielded enough insight to choose a system. However, we also include in our workflow several possible methods of feedback that may refine the analysis to collect more information or consider additional options. Systems may be dropped from consideration and/or added to the analysis. The guarantees required by the expressiveness reductions used can be altered (e.g., if the cost results are too high, the analyst may consider weakening the security requirements to acquire a feasible compromise). The cost measures under consideration may also be changed, possibly to consider additional information and "break ties," or to take into account other changes made (such as newly introduced systems) in the analysis.

This workflow represents what we call *suitability analysis*. We will formally define the suitability analysis in Chapter 4 based on insights motivated by the remainder of this chapter.

3.4 KEY CHALLENGES

In this section, we identify the key challenges in instantiating the workflow described informally in Section 3.3 into a framework that can be used to conduct practical suitability analyses. These challenges will be addressed in subsequent chapters of this dissertation.

3.4.1 Workloads and Application-Aware Expressiveness Analysis

A main challenge in instantiating suitability analysis is developing a formalism for representing the demands of the application: the access control workload. This structure also needs to facilitate application-aware expressiveness analysis, the process that demonstrates an access control system is expressive enough to be used in a particular application. Here, we describe the intuition behind the structure of a workload and the basic process of using it in application-aware expressiveness analysis, leaving the formal details for Chapter 4, where we instantiate our suitability analysis framework.

The *operational component* of an access control workload must describe the set of capabilities that a suitable access control system needs to support in order to properly operate within the application of interest. Thus, it should describe, at a high level, the following components.

- The policy information that the system must maintain
- The procedures that the system requires for modifying the policy
- The access control questions that the system needs to answer based on the policy

We note that we can satisfy these requirements naturally by describing an abstract access control system that meets the applications' requirements. We note that, while access control workloads and systems can be formalized using the same mathematical structure, they differ in their intention: a system represents a functioning piece of software, while a workload's operational component is built by the analyst to represent the higher-level *desired* functionality of a system, without necessarily being appropriate for direct implementation.

The structural similarity between a workload's operational component and an access control system will enable us to construct mappings from the former to the latter. We call this type of mapping an *implementation* of a workload in a candidate system. Implementations will serve a related role to that of expressiveness reductions: the existence of such a mapping (and proof that it preserves certain required properties) shows that the system is capable of satisfying the application described by the workload. Furthermore, an implementation should provide a "recipe" that describes *how* to use the system to satisfy the requirements of the workload.

Given their close relationship to relative expressiveness reductions, and the mathematical similarities between the representations of operational components and systems, it is natural to consider using existing notions of reduction to represent implementations. In subsequent chapters, we will describe the precise requirements for a notion of reduction to be used as an access control implementation.

Describing the capabilities that an application requires of its access control system via the operational component will allow us to decide which systems are *capable* of operating within the application under consideration. However, it does not offer full insight into which is most *suitable* for the application. Towards this goal, an access control workload should also contain an *invocational component* describing the ways in which the capabilities of the workload are to be used.

The main role of the invocational component in suitability analysis is to provide sets of traces of access control actions that are typical for the application in question, and as such this component will need to be able to accurately formalize a representative picture of workload usage. At a minimum, the invocation component should be able to dictate the order in which commands are executed and which queries are asked during which paths of execution. We identify the set of *actions* as the set of commands combined with the set of queries (i.e., the full set of operations that can be executed within the access control system). A simple invocation structure, then, might simply describe the probability distribution among actions. Unfortunately, this does not allow us to describe many realistic scenarios, such as sequences of actions that are often executed together. Another option for an invocation structure is a set of recorded traces of actions. However, it is not clear that this is always realistic to obtain, and past traces may not necessarily be representative of future usage.

Given the lack of access control evaluation tools that are application-aware, there is little work in the field for generating traces that are representative of usage within an application. Thus, for inspiration in designing the first access control workload invocational components, we turn to work in other domains.

In the field of disk benchmarking, Ganger [40] tests several methods for generating synthetic traces of usage, ranging from *simple* (primarily using measured averages) to *interleaved* (interleaving several traces, each generated using a Markov model with measured

transition rates). The key observation was that, in most scenarios, interleaved workloads provided the most accurate approximation of recorded traces. Thus, we postulate that mechanisms for representing workload invocational components must be capable of simulating the interleaved actions of multiple actors.

This view is reenforced by the design of IBM's SWORD workload generator for stream processing systems [4,18]. 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 a formalism for 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. In Chapter 4, we will need to consider the best way of representing the invocational component that takes these factors into account.

3.4.2 Conducting Cost Analysis

In order to conduct cost analysis of access control, one must first formalize the relevant measure (or measures) within which to evaluate the cost. Our formalism for cost measures must remain flexible enough to encode the wide variety of possible access control cost metrics, including those noted in a recent NIST report on the assessment of access control systems [63]. This includes costs such as "steps required for assigning and dis-assigning user capabilities" and "number of relationships required to create an access control policy." Additional desirable costs may include metrics for human work such as "personnel-hours per operation" and "proportion of administrative work to data-entry work," and computational costs such as maximum memory usage and storage overhead. For applications in which multiple metrics are relevant, vectors of cost measures may be needed.

In order to calculate the cost of a particular implementation of a workload, we will need to determine the cost of a trace of actions in the implementing system, which necessitates our determining the costs of executing the system's individual commands and queries. In addition to these cost functions for determining individual actions' costs, cost analysis will require

methods for combining this information with the representative workload traces generated by the invocation component to determine a total expected cost for each implementation. To accomplish this, the sequence of actions generated by the workload's invocation component must be translated into actions in each of the implementing systems, using the specification of each system's implementation of the workload. The resulting actions can then be executed (or simulated) in each system, accruing costs using the corresponding cost functions, and tracking changes to the system's state. An algorithm for conducting this series of analysis steps will be developed in Chapter 4.

3.4.3 Application-Aware Expressiveness Metrics

We have noted that we can use some existing notions of reduction to satisfy the implementation structure, and that our unified framework for suitability analysis will include minimum required properties that a notion of reduction must satisfy to be used in such a scenario. However, we might desire that this mapping be flexible enough to allow a wide variety of expressiveness metrics that are application-aware in the same way that a flexible notion of costs allows application-aware cost measures. In the context of implementations, this flexibility may include selecting which reduction properties should be required for evaluation with respect to a particular application, based on which properties represent security issues to which the application is sensitive.

Thus, on top of a minimal set of requirements that an implementation must satisfy, we will develop a fine-grained representation of the wide range of expressiveness mappings based on the properties that each enforces on top of this minimum. This fine-grained representation will allow us to decompose and formally compare existing notions of reduction, assisting analysts in choosing which of these notions is appropriate for a particular analysis. In addition, this will assist analysts in crafting *new* notions of reduction specific to the requirements of the desired analysis. We present this flexible, fine-grained taxonomy of expressiveness reductions in Chapters 8 and 9.

3.4.4 Wider Security Applications

We note that there is a range of computer security problems that follow a similar structure to suitability analysis: solutions first need to be formally proven to be capable of satisfying the application's needs, then experimentally evaluated with regards to their expected costs while operating within that application. As such, we will investigate the feasibility of applying the techniques developed for use in the suitability analysis workflow to these more general security problems, allowing analysts to model the application as a workload, candidate solutions as systems, and the execution of the former using the latter as an implementation.

For instance, given the rise of cloud computing, we consider the challenges of enforcing access controls for the scenario in which the provider is (partially) untrusted. There has been much discussion about how to achieve access control on the cloud, but in situations where confidentiality or integrity with respect to the cloud provider itself is necessary, cryptographic systems are a natural—if not the only—solution. This represents a shift from traditional access control, where we assume a trusted computing base fields all requests. However, we believe that it can be represented as a suitability analysis problem, due to its structure of requiring a solution that is efficient while also being expressive enough to make formal security guarantees. We investigate the problem of evaluating the suitability of cryptographic solutions to this workload in Chapter 7.

3.5 SUMMARY

In this chapter, we have motivated the need for new evaluation techniques for access control that transcend expressiveness and account for both the differences between applications and ordered cost metrics. In doing so, we support our thesis statement by motivating the types of analysis questions it claims we can answer. We gave an informal description of the suitability analysis workflow and discussed how it can be utilized to determine which access control system is best suited to an application. In a framework that instantiates this workflow, an application's requirements are formalized as a workload, a novel structure that

enables evaluation of systems' ability and cost to operate within the application. We then discussed how we will proceed to tackle the key challenges in fully realizing a suitability analysis framework.

4.0 INSTANTIATING SUITABILITY ANALYSIS

In this chapter, we formalize the access control *suitability analysis problem*, which gives rigor to the type of analysis described in our thesis statement and informally in Section 3.3. We then present a series of requirements (e.g., efficiency, accuracy) for suitability analysis frameworks that solve this problem. We then develop our own mathematical framework to conduct suitability analysis in two phases, first through reductions to prove that candidate systems are *capable* of implementing the workload, and then through simulations to calculate expected *costs* of using each candidate scheme to do so. We then evaluate this framework both formally, by showing it satisfies the aforementioned requirements, and practically, by demonstrating how our framework can be used to evaluate the suitability of several popular access control systems to the program committee workload described in Section 3.2.[1]

4.1 INTRODUCTION

In this chapter, we address several of the challenges identified in Chapter 3, and in doing so we take our first concrete step toward developing the techniques that will enable suitability analysis. Namely, we propose a framework that enables us to carry out application-aware expressiveness analysis as well as cost analysis of access control.

We first identify and formalize the *access control suitability analysis problem*, which considers both qualitative and quantitative (expressiveness and cost) metrics. We then propose a formal mathematical framework to carry out this type of analysis. We first develop a precise formalism for access control workloads. This structure enables us to formalize an

[1]The material presented in this chapter was first published as [46].

application's access control needs and the expected uses of these functionalities. Analysis then consists of two orthogonal tasks: (i) demonstrate that each candidate access control system can *safely* implement the workload (qualitative analysis), and (ii) quantify the *costs* associated with using each candidate system (quantitative analysis). Toward carrying out such an evaluation, we will develop techniques for representatively sampling from the workload's functionality, guidelines and mathematical structures for formally specifying access control cost metrics, and a simulation algorithm for carrying out cost analysis. 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 two-phase suitability analysis frameworks.
- We formalize the *access control workload*. This enables analysts to clearly and concisely specify the functionalities that must be provided by access control systems that are to be used within a given context, as well as identify the ways in which these systems are expected to be used.
- We develop the first *two-phase suitability analysis framework*. Within this framework, qualitative analysis is conducted by constructing and formally proving the correctness of implementations of the workload in each of the candidate systems. Then, quantitative analysis is conducted by sampling traces from the workload's expected usage patterns and translating these traces into behavior in each candidate system.
- We propose a constrained, actor-based workload invocation structure that allows us to generate representative traces of workload usage patterns based on the interleaved execution of multiple users' actions in the system.
- We present a simulation-based cost analysis algorithm for exploring the expected costs of deployment.
- We evaluate our framework *formally* by proving that our simulation procedure is fixed-parameter tractable, and *practically* via a case study demonstrating how our framework can be effectively used to gain insight into a realistic scenario based on an academic conference management system.

In Section 4.2, we formalize the suitability analysis problem and overview our two-phase

approach to solving it. Further, we articulate a set of requirements for suitability analysis frameworks. Sections 4.3 and 4.4 describe in depth our approach to a two-phase solution to the suitability analysis problem. We describe a case study investigating the use of our framework in Section 4.5. We then discuss the properties upheld by our framework (Section 4.6) and summarize (Section 4.7).

4.2 THE SUITABILITY ANALYSIS PROBLEM

As was discussed in Section 3.3, in order to evaluate against the specific demands of the application in question, an analyst must construct a workload that represents that application, including both required capabilities and expected usage of those capabilities. To conduct quantitative evaluation that is application-aware, a reduction must be constructed from the workload to each candidate system, showing that each is capable of satisfying the workload's requirements. To conduct application-aware qualitative evaluation, costs must be evaluated with respect to the expected usage described by the workload.

To enable the analysis to consider whichever cost metrics are of the greatest consequence to the application, we present the minimal structure that a cost measure must have. We also prove that a vector of cost measures is a cost measure, allowing multiple costs to be considered. One of the inputs to the analysis problem, then, is the set of costs that should be considered in quantitative analysis.

Similarly, to enable the analysis to consider whichever of the wide range of expressiveness metrics is most appropriate for the application, the analysis problem takes as input a set of *security guarantees* that may be enforced on the mapping constructed during qualitative analysis. The suitability analysis problem takes each of these components into account, and is stated as follows.

Suitability Analysis Problem. Given an access control workload $\mathcal{W}$, a set of candidate access control systems $\mathfrak{Y}$, a set of security guarantees $\mathcal{G}$, and a set of ordered cost measures $\mathfrak{C}$, determine:

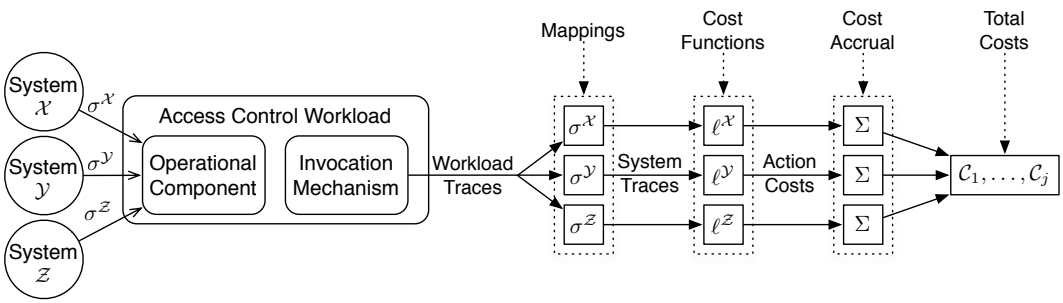


Figure 2: Overview of an application-aware analysis framework for access control

(i) a set of implementations $\mathfrak{I}$, each describing how a candidate system $\mathcal{S}_i \in \mathfrak{Y}$ can implement $\mathcal{W}$ while satisfying a maximal subset of $\mathcal{G}$

(ii) for each $\mathcal{I} \in \mathfrak{I}$, the expected costs relative to $\mathfrak{C}$ of using $\mathcal{I}$ to implement $\mathcal{W}$ $\Diamond$

Figure 2 depicts an overview of the two-phase approach to solving the suitability analysis problem. First, in expressiveness analysis, the analyst formalizes a workload, candidate systems, and implementations, and then proves that the implementations satisfy the desired security guarantees. Next, to start cost analysis, the analyst formalizes the cost measures that represent the quantitative metrics the application is sensitive to, and labels each action within each system with its respective costs. Cost analysis is completed by generating representative traces of usage from the workload, translating them into equivalent system traces using the mappings constructed during expressiveness analysis, and simulating the system traces while recording the costs of each action.

We briefly mention the final (sometimes implicit) step following suitability analysis: selecting the appropriate system to deploy. While it is appealing to imagine the suitability analysis problem as an instance of the question, "Which of these systems is most suitable to this workload?," in reality it is a bit more subtle. In order to give an absolute answer to the former informal question, we would need to reconcile implementations that preserve incomparable maximal subsets of implementation properties, implementations that incur lower costs in some measures but higher in others, and those that can preserve stronger reduction properties with those of lower cost. That is, the only time this informal question

can have a definitive answer is when one implementation admits security properties consisting of at least the sum of all other implementations *and* incurs the minimum cost in all cost measures. While we can artificially restrict the framework in order to create this scenario (e.g., by forcing the cost lattice to be totally ordered), we have instead opt to define the suitability analysis problem such that Phase 1 identifies the possible implementations and the security guarantees that each upholds, and Phase 2 determines the cost of each implementation in each of the cost measures. The analyst utilizing the suitability analysis framework will then need to consider these results when selecting the most appropriate system for the application.

We now explore *solution requirements* (each denoted SR#) for suitability analysis frameworks. First, we consider requirements in how representative traces through an access control workload ($\mathcal{W}$) are chosen for exploration in cost analysis. Because exploring all traces of possible usages during cost analysis will likely be impractical, we must sample from this set in a way that selects traces that are representative of the expected behavior. Our first two requirements ensure that the framework can generate traces that accurately model the tasks carried out within an organization, and the interactions required to support and process these tasks.

SR1: Domain exploration Large organizations are complex systems with subtle interactions, the emergent behaviors of which 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.

SR2: Cooperative interaction Tasks within large organizations typically require the interaction of multiple individuals. As such, suitability analysis frameworks should support operational workflows and constraints on their execution.

Next, we must ensure that the suitability analysis framework can be tuned to meet the specific needs of an application via choosing the metrics used to assess the suitability of an access control system for a given workload. This includes both the security guarantees used in expressiveness evaluation ($\mathcal{G}$) and the cost metrics used in cost evaluation ($\mathfrak{C}$).

SR3: Tunable safety There may be many different ways for a system to implement a

given workload. Without enforcing structure on the mapping encoding this implementation, even the most under-expressive systems can appear to implement a workload [113, 114]. It must be possible for an analyst to specify the security guarantees required for implementations of their workload.

SR4: Tunable cost There is no single notion of cost that is sensible for use in every access control analysis [63]. Suitability analysis frameworks must be capable of representing many types of costs (e.g., computational, communication, and administrative), and examining multiple costs simultaneously.

Finally, we consider requirements that ensure that the suitability analysis framework remains practical to use, even for large-scale application workloads.

SR5: 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 large systems.

SR6: Accuracy Since exploring all possible traces is impractical, it must be possible to approximate the expected error of costs obtained by exploring only a specific subset of traces.

These requirements guide the development of our suitability analysis framework; we will discuss our ability to achieve these requirements in Section 4.6.

4.3 PHASE 1: EXPRESSIVENESS ANALYSIS

In this section, we describe the first phase of suitability analysis, in which the analyst constructs mappings proving that the each candidate system is expressive enough for the workload. A prerequisite to this is formalizing the access control systems themselves. We now restate the definition of access control system, the state-machine formalism that will represent the candidate systems.

Definition 2 (Access Control System, restated)**.** Given access control model $\mathcal{M} = \langle \Gamma, \mathcal{R} \rangle$, an *access control system* within $\mathcal{M}$ is a state transition system, $\mathcal{S} = \langle \Gamma, \Psi, Q \rangle$, where Ψ is

the set of commands, each command $\psi \in \Psi$ is a function $\Gamma \to \Gamma$, $Q \supseteq \mathcal{R}$ is the set of queries, and each query $q \in Q$ is a function $\Gamma \to \{\text{TRUE}, \text{FALSE}\}$. $\Diamond$

Recall the discussion in Section 3.4.1 of the workload formalism, used for representing the demands of the application. We noted that this structure should contain two components. The operational component describes the required capabilities, and can be represented as an abstract access control system. The invocational component is used to represent the usage of this abstract access control system. It will be used in cost analysis to provide the traces of actions whose costs will be evaluated. More formally, each trace defines an initial state and a sequence of commands and queries that are executed. Given these components, we now formally define the access control workload.

Definition 12 (Access Control Workload)**.** An *access control workload* is defined by $\langle \mathcal{A}, \mathcal{T} \rangle$, where:

- $\mathcal{A} = \langle \Gamma, \Psi, Q \rangle$ is the operational component: an abstract access control system
- $\mathcal{T}$ is the invocational component: a set of pairs $\langle \gamma_0, \tau \rangle$ where $\gamma_0 \in \Gamma$ and $\tau = a_1 \circ a_2 \circ \ldots$ is a sequence where $\forall a_i \in \tau : a_i \in (\Psi \cup Q)$. $\Diamond$

The common structure between the workload's operational description and access control systems enables us to utilize a special type of access control reduction: an *implementation* of a workload maps the desired behavior in the workload to the capabilities of a candidate system. Like other access control reductions, a proof that an implementation is correct and satisfies the desired security guarantees must be manually constructed. As we discussed in detail in Section 2.2, various prior works propose different sets of security guarantees defining their particular notion of expressive power. Rather than dictate a particular notion of reduction for Phase 1 of suitability analysis, we seek to enable a wide range of application-sensitive expressiveness metrics, and thus we lay out the minimal requirements imposed on a form of reduction in order for it to be usable in suitability. For a notion of reduction to be usable as an implementation, the existence of such a reduction would allow one to determine how to replace one system with another. This will allow us to translate workload traces into system traces for the purposes of cost analysis. We refer to these requirements as the *implementability requirements*, each denoted IR#.

IR1: State mapping In order to use system $\mathcal{S}$ to satisfy workload $\mathcal{W}$, it must be possible to (uniquely) determine which $\mathcal{S}$ state to use in place of a particular $\mathcal{W}$ state. Thus, an implementation must include a *state mapping*, a function from workload states to simulating system states.

IR2: Command mapping To use $\mathcal{S}$ to satisfy $\mathcal{W}$, it must be possible to transform $\mathcal{S}$'s state in ways that are equivalent to the required functionalities described in $\mathcal{W}$. That is, we must be able to determine the commands in $\mathcal{S}$ that perform an equivalent state transformation to each of the abstract commands in $\mathcal{W}$. It is not necessarily the case that each $\mathcal{W}$ command can be simulated using a *single* $\mathcal{S}$ command, so an implementation requires a function from $\mathcal{W}$ commands to *sequences of* $\mathcal{S}$ commands. In addition, it may be necessary to map a $\mathcal{W}$ command differently depending upon the state in which it is intended to be executed. Since using $\mathcal{S}$ for workload $\mathcal{W}$ means we have a $\mathcal{S}$ state to inspect during execution, this function should map a $\mathcal{W}$ command and an $\mathcal{S}$ state to a sequence of $\mathcal{S}$ commands.[2]

IR3: Query decider Just as we need to transform any $\mathcal{W}$ command to functionally equivalent $\mathcal{S}$ commands, we also need to be able to provide answers to any *query* that $\mathcal{W}$ requires. In some workloads, this may include only the authorization requests that grant or revoke access to resources in $\mathcal{W}$, while in other cases additional types of queries are allowed, such as "Is user u a member of role r?". To remain general, we simply require that a $\mathcal{W}$ query can be answered using a $\mathcal{S}$ state. Thus, an implementation must include a function that maps each $\mathcal{W}$ query and $\mathcal{S}$ state to either TRUE or FALSE.[3]

The definition of implementation formalizes the implementability requirements requirements IR1–3. Thus, any type of access control reduction of the following form can be used to solve the suitability analysis problem.

Definition 13 (Access Control Implementation)**.** Given a workload $\mathcal{W} = \langle \mathcal{A}, \mathcal{T} \rangle$ where $\mathcal{A} = \langle \Gamma^{\mathcal{A}}, \Psi^{\mathcal{A}}, Q^{\mathcal{A}} \rangle$, and a system $\mathcal{S} = \langle \Gamma^{\mathcal{S}}, \Psi^{\mathcal{S}}, Q^{\mathcal{S}} \rangle$, an implementation of $\mathcal{W}$ in $\mathcal{S}$ is a triple of functions $\sigma = \langle \sigma_\Gamma, \sigma_\Psi, \sigma_Q \rangle$, where:

[2] As these requirements are only the minimum required by suitability analysis, specific types of implementations can include special cases of the command mapping. For instance, under certain notions of implementation, a $\mathcal{W}$ command may be mapped to the same series of $\mathcal{S}$ commands regardless of the state; in others, only particular portions of the state may be considered.

[3] A more specific query decider may map each $\mathcal{W}$ query to a single $\mathcal{S}$ query q, then return the value of q in the current $\mathcal{S}$ state.

- $\sigma_\Gamma : \Gamma^{\mathcal{A}} \to \Gamma^{\mathcal{S}}$ is the state mapping
- $\sigma_\Psi : \Psi^{\mathcal{A}} \times \Gamma^{\mathcal{S}} \to \left(\Psi^{\mathcal{S}}\right)^*$ is the command mapping
- $\sigma_Q = Q^{\mathcal{A}} \times \Gamma^{\mathcal{S}} \to \{\text{TRUE}, \text{FALSE}\}$ is the query decider $\Diamond$

Many notions of expressiveness in the literature meet these requirements; notable examples include the state matching reduction (Definition 11, restated from [113, 114]), as well as the reductions defined by Chander et al. (Definitions 8 and 9, restated from [21]) and Ammann et al. (Definition 6, restated from [2, 3]). We leave the selection of reduction form to the analyst, who can make such a selection among those proposed in the literature based on the security guarantees the workload requires. In Chapter 8, we will revisit the implementability requirements and provide greater guidance in selecting, for a particular analysis, the additional properties to be enforced atop this minimal definition.

4.4 PHASE 2: COST ANALYSIS

In this section, we describe the second phase of suitability analysis, in which the expected cost is determined for each implementation of the workload.

4.4.1 Trace Generation

In cost analysis, we are interested in determining the cost of using a particular implementation *under the expected usage of the system*. For this reason, cost analysis must evaluate costs with respect to the particular usage described in the invocation mechanism of an access control workload (Definition 12). This is represented as a set of traces through the system's actions (commands and queries). In cost analysis, we need to sample from this set of traces in a way that is representative of the *expected* usage. We also need to do so in a way that satisfies the solution requirements set forth in Section 4.2. For example, *Domain Exploration* requires that we are able to alter input parameters. Implicitly in this requirement is the assumption that the trace reacts to these initial state parameters (e.g., more users typically means more frequent execution of commands and queries).

A first attempt at tackling this problem may not be trace generation at all, but rather exhaustive search of the space of traces. However, we quickly abandon such approaches for several reasons. The most obvious is *Tractability*, which requires tractable solutions (recall that the set of all valid traces is most likely infinite). More subtly, we note that *Tunable Costs* requires that the cost analysis is tuned to the particular scenario in which the system will be deployed; determining which system is most efficient across all possible usages violates this requirement.

Next, one might consider recording real-life traces and playing them back within the simulation environment. After all, real traces are guaranteed to be realistic. However, collecting such traces can be expensive, and recorded traces are not necessarily *representative*. Further, recording and playing back static traces violates *Domain Exploration*, which requires the exploration of a variety of initial conditions.

Thus, inspired by work in disk benchmarking [40] and stream processing [4, 18] (see Section 3.4.1), we opt to conduct trace generation using an interleaved agent-based technique. This allows the analyst to model usage of the access control system at the level of the individual user. To this end, we define an extension of the invocation mechanism that utilizes 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 in ways that are described by *actor machines*. We express the various ways in which actors cooperate to complete a task using *constrained workflows*. This structure specifies dependencies between related actions, and utilizes constraints to restrict which user can execute each action. Together, these structures enable the modeling and simulation of complex and concurrent behaviors of the entities in a workload.

We now formalize *actions*, the basic units of work executed by an actor in the system. An action is a *parameterized* generalization of queries and commands. This allows us to specify the generic description of the action (e.g., check an access, assign a role) and separately assign the precise parameters (e.g., the specific users, documents, and roles involved). These parameters can then be assigned statically by the executing actor machine or dynamically upon execution.

Definition 14 (Access Control Action)**.** Given an access control system, $\mathcal{S} = \langle \Gamma, \Psi, Q \rangle$, an

access control *action* for $\mathcal{S}$ is a function from a set of parameter spaces (derived from Γ) to the system's set of operations, and is defined as $a : P_1 \times \cdots \times P_j \to \Psi \cup Q \cup \{\varnothing\}$, where:

- $P = \langle P_1, \ldots, P_j \rangle$ is the set of parameter spaces from which the actions's j parameters are drawn (e.g., the set of subjects, objects, roles).[4] We denote $P_1 \times \cdots \times P_j$ as P^*.
- $a : P^* \to \Psi \cup Q \cup \{\varnothing\}$ maps each parameterization to a command or query in $\mathcal{S}$; or to $\varnothing$, which designates no command or query is to be executed ◊

To describe the behavior of actors, we employ state machines that we call *actor machines*. The primary goal of an actor machine is, at each instant in time, to produce a candidate action for the entity that it represents to execute (or to pass on executing an action in this instant). Each state in an actor machine is labeled with an action and a (possibly incomplete) parameterization. Transitions in this state machine are labeled with *rates* akin to those used in continuous-time Markov processes (e.g., [79]). We generate representative traces of actor behavior by probabilistically walking this machine, following transitions with probabilities proportional to their rates.

Definition 15 (Actor Machine)**.** Let $\mathcal{S}$ be an access control system, A a set of actions from $\mathcal{S}$, and $\mathfrak{V}$ a set of variable symbols. An *actor machine* for $\mathcal{S}$ is the state machine $\langle S, \Phi, R \rangle$, where:

- S is the set of actor machine states
- $\Phi : S \to A \times (P_1 \cup \mathfrak{V}) \times \ldots \times (P_j \cup \mathfrak{V})$ labels each actor machine state in S with an action and a (perhaps partial) parameterization of that action. That is, parameters can be assigned a static value or a variable to be assigned dynamically during execution.
- $R : S \times S \to \mathbb{R}$ is the set of transition rates ◊

The semantics of the execution of an actor machine are as follows. R describes the rates of transitioning from one state to another. To achieve the Markov property, the time spent waiting to exit a state is exponentially distributed, with an exponential rate parameter that is proportional to the sum of the rates of all exiting transitions. An actor carries out a state's

[4] Unless otherwise noted, we use the first parameter of an action to represent the executing entity. For queries, this allows different responses for different queriers. For commands, this allows restrictions on the entities permitted to execute.

action upon entering the state (possibly after a pause, e.g., an action to submit a comment to a forum may pause for several minutes while the message is composed).

Example actor machines are demonstrated in Figure 3a. Here, we classify actors into administrators and users. Users generate documents and occasionally request that a document be declassified for public consumption, while administrators approve declassification requests. Due to the labeled rates on this machine, an administrator is expected to approve a declassification request on average in one day, and roughly 10% of users request a declassification each month. Transitions labeled with ∞ occur immediately.

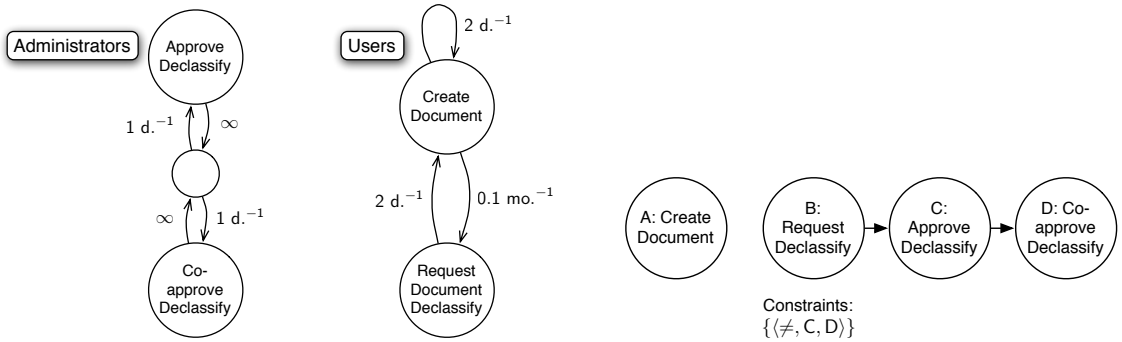
To describe dependencies between actions taken by one or more actors, we make use of the notion of a *constrained access control workflow*, which organizes the execution of sequences of actions. Intuitively, the goal of this structure is to approve or deny a candidate action returned from an actor machine on the basis of actions taken by other actors in the system. Formally, this structure specifies the partial order describing action dependence as well as constraints that restrict the set of users that can execute various actions.

Definition 16 (Constrained Workflow)**.** Let $\mathcal{S}$ be an access control system and $\mathfrak{A}$ a set of access control actor machines for $\mathcal{S}$. We say that $W = \langle A, \prec, C \rangle$ is a *constrained access control workflow* over the system $\mathcal{S}$, where:

- A is the set of actions from $\mathcal{S}$
- $\prec \subset A \times A$ is the partial order describing action dependencies. If $a_1 \prec a_2$, then a_2 depends on a_1, and a_2 cannot be executed until after an execution of a_1.
- C is the set of constraints, each of the form $\langle \rho, a_1, a_2 \rangle$ (with $a_1, a_2 \in A$). Here, ρ is a binary operator of the form $\mathfrak{A} \times \mathfrak{A} \to \{\text{TRUE}, \text{FALSE}\}$. For example, $\langle \neq, a_1, a_2 \rangle$ says that a_1 and a_2 must be executed by different actors. $\Diamond$

Figure 3b displays a constrained workflow with two *tasks* (disjoint subsets of the workflow): document creation and declassification. The former is a degenerate task containing a single unconstrained action. Declassifying a document, on the other hand, requires the approval of two different administrators. The workflow allows administrators to approve declassification only after the request, and the approvals must be executed by distinct administrators.

The *actor-based invocation mechanism* combines the components discussed above; it



(a) Actor machines (b) Workflow

Figure 3: Example invocational structures

refines the set of traces included in an access control workload (Definition 12) using a constrained workflow (an instance of Definition 16), a set of actor machines (each an instance of Definition 15), and a method for extracting the active actor machines from an access control state.

Definition 17 (Actor-Based Invocation)**.** Let $\mathcal{S} = \langle \Gamma, \Psi, Q \rangle$ be an access control system. We say that $I^{\mathcal{S}} = \langle W, \mathfrak{A}, M \rangle$ is a *constrained, actor-based access control invocation mechanism* over the system $\mathcal{S}$, where:

- W is a constrained workflow over $\mathcal{S}$
- $\mathfrak{A}$ is a set of actor machines over $\mathcal{S}$
- $M : \Gamma \to \wp(\mathfrak{A})$ is the actor machine liveness function (i.e., a function that maps each access control state to the set of actor machines active in that state) ◊

Workload traces are generated from this structure by combining the elements discussed above. In a given time step, each actor is consulted for a candidate action. If that actor returns an action, its missing parameters are filled in. Then, the action is checked against each live workflow instance, including a new, blank workflow instance if needed. If allowed, the action is added to the trace, and the next actor is consulted. If any action in a timestep would add or remove an actor, the corresponding change is mirrored in the set of currently

active actor machines.

4.4.2 Calculating the Costs of Traces

Once representative workload traces are generated and mapped into candidate system traces, the second main task of cost analysis is calculating the costs of these traces. In order to do so, the analyst must first identify the relevant cost measures—that is, what types of costs should be included in consideration when determining expected costs. There is a wide variety of cost measures that one may consider, as discussed in Section 3.4.2 These can include any of the various computational, storage, personnel, and throughput costs associated with maintaining a particular implementation of a workload in a particular access control system.

Due to this wide variety among cost measures and the vast differences among applications requiring access control systems, which of these cost measures should be considered in a particular analysis is very application-dependent: administrators of a high-security military access control system will likely value the cost of trained, security-cleared labor much more highly than any computational cost, while the access control system for a lightweight web server may have a much greater concern for computational resources. The considered measures should be descriptive of what types of costs are important to the analysis, while also enabling the analyst to easily label actions with costs. For example, while "operational cost per day" may be representative of evaluation goals in industry, it is hard to assign costs in this measure to any access control action. A measure such as "average administrative personnel-hours," on the other hand, is more easily quantified and enables the same types of analyses. In this dissertation, we do not commit to a particular cost measure, but rather develop a framework that operates on any measure satisfying a number of simple properties.

Definition 18 (Cost Measure)**.** A *cost measure* is defined by the ordered abelian monoid $\mathcal{C} = \langle C, \bullet, \preceq \rangle$, where C is the set of *costs*, $\bullet$ is the closed, associative, commutative *accrual operator* over C with identity 0_C, and $\preceq$ is a *partial order* over C such that $\forall a, b \in C : a \preceq a \bullet b \wedge b \preceq a \bullet b$. $\lozenge$

Definition 18 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

systems [63]. 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 $\langle \mathbb{N}, \max, \leq \rangle$.

In order to accomplish the aggregation step and calculate the total cost of a particular implementation, more information is needed than simply the measure; the framework needs a way of calculating the cost of executing a particular action within an implementing system. Sometimes, the cost of any execution of an action is constant (e.g., creating a document requires a constant amount of I/O). In other cases, the parameters of the action affect the cost (e.g., adding a user 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 currently in the system). Thus, in the general case, a cost function must map each (action, parameterization, state) tuple to an element of the relevant cost measure.

Definition 19 (Cost Function)**.** Let $\mathcal{S} = \langle \Gamma, \Psi, Q \rangle$ be an access control system, A a set of actions from $\mathcal{S}$, P_a^* the set of parameter spaces for $a \in A$, and $\mathcal{C} = \langle C, \bullet, \preceq \rangle$ a cost measure. A *cost function* for $\mathcal{C}$ in $\mathcal{S}$ is a function $\ell_{\mathcal{C}}^{\mathcal{S}} : A \times P_a^* \times \Gamma \to C$, which maps each action a, parameterization $p_a^* = \langle p_1, p_2, \ldots, p_j \rangle$, and state γ to the member of the cost set C that best represents the costs associated with executing $a(p_1, p_2, \ldots, p_j)$ in γ. $\Diamond$

Thus, given system $\mathcal{S} = \langle \Gamma, \Psi, Q \rangle$ and a trace $\langle \gamma_0, a_1 \circ a_2 \circ \ldots \rangle$ from $\mathcal{S}$, say that the state of the system after executing a_i is $\gamma_i = \mathit{next}(\gamma_{i-1}, a_i)$. Then, given measure $\mathcal{C} = \langle C, \bullet, \preceq \rangle$, and cost function ℓ for $\mathcal{C}$ in $\mathcal{S}$, the cost of the first i actions in trace A is described by the following recurrence.

$$c_i = \begin{cases} 0_C & i = 0 \\ c_{i-1} \bullet \ell(a_i, \gamma_i) & i > 0 \end{cases}$$

In addition to the cost functions that are of specific interest to the analyst, our cost analysis simulation process also requires the specification of each system's *time function*. The time function is a cost function with measure $\langle \mathbb{R}, +, \leq \rangle$, describing the duration of time required to complete each access control action. This time corresponds to the duration that an actor pauses before completing an action when entering a state in the actor machine.

4.4.3 Simulation Procedure

Once the analyst has defined the trace generation structures, a set of cost measures, and cost functions for each candidate system, she can conduct cost analysis via simulation. Our main simulation procedure, ACCostEvalSim (shown in Algorithm 1), conducts a single, randomized run of the system. First, the workload's initial state is chosen by sampling from a distribution provided by the analyst, and this initial workload state is translated into an initial state for each candidate system. An actor machine is then launched for each actor in the workload's invocation mechanism. At each time step, the clock is incremented and each actor machine is inspected for the next action, as per the execution semantics of the actor machine (Section 4.4.1).

Since a workload's invocation mechanism includes a constrained workflow, we must consider an actor machine's actions with respect to those that have been completed thus far within the related task. This includes not only ensuring that a particular action is allowed by the workflow, but also that it will not render the remaining actions in a task impossible. For instance, if actions a_1 and a_2 must be completed by different actors, and actor α is the only user able to complete a_2, then α should not be allowed to carry out action a_1 since this would leave no possible actor to carry out a_2. The general case of this problem is known as the *workflow satisfiability problem* (WSP) [24, 28, 117].

Thus, if the actor machine returns an action at a given time step, an instance of WSP is instantiated and solved by procedure WSat to ensure that the actor can execute the action without rendering the constrained workflow instance unsatisfiable. For independent actions (i.e., those in $\{a_1 \mid \nexists\, a_2.(a_2 \prec a_1)\}$), a new workflow instance is created and added to the pool of partially-executed workflows. Otherwise, the action is taken in the context of an existing

Algorithm 1 ACCostEvalSim: A simulation procedure for application-aware cost analysis of access control

Input: $\mathfrak{Y}$, set of candidate systems
Input: Σ, set of implementations $(\forall \mathcal{S} \in \mathfrak{Y} : \sigma_{\mathcal{S}} \in \Sigma)$
Input: $\mathfrak{C}$, set of cost measures $(\mathcal{C} = \langle \mathbb{R}, +, \leq \rangle \in \mathfrak{C})$
Input: L, set of cost functions $(\forall \mathcal{S} \in \mathfrak{Y}, \mathcal{C} \in \mathfrak{C} : \ell_{\mathcal{C}}^{\mathcal{S}} \in L)$
Input: $I = \langle W, \mathfrak{A}, M \rangle$, invocation mechanism
Input: $\gamma_0 \in \Gamma^{\mathcal{W}}$, start state
Input: T_f, goal time
Input: t, time step

procedure ACCostEvalSim$(\mathfrak{Y}, \Sigma, \mathfrak{C}, L, I, \gamma_0, T_f, t)$
  $\mathbf{S} \leftarrow \{\}$ ▷ Initialize set of running AC systems
  $T \leftarrow 0$ ▷ Initialize master clock
  for all $\mathcal{S} \in \mathfrak{Y}$ **do** ▷ Initialize each system's state
    $\mathbf{S} \leftarrow \mathbf{S} \cup \{\mathcal{S}\}$
    $\gamma_{\mathcal{S}} \leftarrow \sigma_{\mathcal{S}}(\gamma_0)$ ▷ Initial state of system $\mathcal{S}$
    for all $\mathcal{C} \in \mathfrak{C}$ **do**
      $c_{\mathcal{C}}^{\mathcal{S}} \leftarrow 0_{\mathcal{C}}$ ▷ Total cost of system $\mathcal{S}$ in $\mathcal{C}$
    $\mathbf{A}_{\mathcal{S}} \leftarrow M(\gamma_0)$ ▷ Set of running actor machines
    for all $\alpha \in \mathbf{A}_{\mathcal{S}}$ **do**
      $T_\alpha \leftarrow 0$ ▷ Per-actor clocks
  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 $T_\alpha < T$ **then** ▷ Check actor busy state
          $\langle k, P_k \rangle \leftarrow$ nextAction(α)
          if $k \neq \varnothing \wedge$ WSat$(k, \alpha, P_k) \neq \varnothing$ **then**
            $T_\alpha \leftarrow T + \ell_{\mathcal{C}}^{\mathcal{S}}(k)$ ▷ 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
        $\langle k', P'_k \rangle \leftarrow \sigma_{\mathcal{S}}(\langle k, P_k \rangle)$ ▷ Translate workload action to system action
        for all $\mathcal{C} \in \mathfrak{C}$ **do**
          $c_{\mathcal{C}}^{\mathcal{S}} \leftarrow c_{\mathcal{C}}^{\mathcal{S}} \bullet_{\mathcal{C}} \ell_{\mathcal{C}}^{\mathcal{S}}(\langle k', P'_k, \gamma_{\mathcal{S}} \rangle)$
        if k is a command **then**
          $\gamma_{\mathcal{S}} \leftarrow next(\gamma_{\mathcal{S}}, k'(P'_k))$ ▷ Update state
  for all $\mathcal{S}' \in \mathfrak{Y}$ **do**
    Output $\langle \mathcal{S}', c_{\mathcal{C}_1}^{\mathcal{S}'}, \ldots, c_{\mathcal{C}_m}^{\mathcal{S}'} \rangle$

workflow instance that is already in progress. After all workload actions for a time step are collected (and verified to be satisfiable by WSat), they are translated into system actions, their changes are effected in the systems' states, and their costs are accrued. Finally, the set of actors is adjusted according to changes in the state. Once the goal time is reached, the total costs are output.

To address the requirement of *Tractability*, we prove that ACCostEvalSim is *fixed-parameter tractable* (FPT) when workflow constraints are user-independent.[5] We first define FPT; for more details on parameterized complexity, see, e.g., [117].

Definition 20 (Fixed-Parameter Tractable)**.** A *parameterized problem* is a decision problem in which inputs are of the form $\langle x, k \rangle$, where k is the *parameter* of the problem. A parameterized problem is *fixed-parameter tractable* if it can be decided for input $\langle x, k \rangle$ in running time $f(k) \times |x|^{\mathcal{O}(1)}$, where $f(k)$ is some function depending only on k. ◊

Theorem 1. *Assuming that workflow constraints are restricted to user-independent constraints, the simulation procedure* ACCostEvalSim *is* fixed-parameter tractable *in 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** $\mathcal{S} \in \mathfrak{Y}$) handles assignments and initializations. The final loop (**for all** $\mathcal{S}' \in \mathfrak{Y}$) 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 systems ($|\mathfrak{Y}|$), 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 (i.e., WSat solves an instance of the WSP problem).

By previous work [24, 117], 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 $S = |\mathfrak{Y}|$ is the number of systems and $T = T_f/t$ is the number of time steps to simulate. Since T is an input, this means the algorithm is

[5] Prior work has shown that most practically-relevant constraints are user-independent, including separation of duty and all constraints specified in the ANSI RBAC standard [24].

pseudo-polynomial in T and FPT in α. Since FPT is a generalization of pseudo-polynomial time [62], Algorithm 1 is FPT, thus satisfying *Tractability*. □

4.5 CASE STUDY

In this section, we in which we demonstrate the suitability analysis process using our framework. This case study explores the workload based on an online management system for an academic conference first discussed in Section 3.2, including paper submissions, reviews, and discussion. The specification of the workload is based on the group-centric secure information sharing model [71, 73], which we will discuss in greater detail in Chapter 6. Our candidate systems include two variants of role-based access control [36, 103, 104] and traditional UNIX user-group-other permissions [42]. The qualitative phase is conducted using parameterized expressiveness [59], and the quantitative phase is conducted using several cost measures that indicate how naturally the systems can implement the workload.

4.5.1 Workload and Candidate Systems

The workload's operational component (i.e., the abstract system that describes the application's requirements) is based on a group-based program committee workload. Users can join and leave groups, and objects can be added and removed from groups. The log of these events is used to decide whether a user can access an object. Users who perform a *strict join* to a group receive access only to objects added after they join, whereas a *liberal join* grants immediate access to all existing objects. A *strict leave* rescinds all of the user's accesses within the group; a *liberal leave* allows the user to retain access. All objects (i.e., papers, reviews, discussion messages) are added to groups via *liberal add*, and thus whether a user can access an object in a group is determined solely by the relative times the join and add took place and the variants of join/leave that was performed.

Definition 21 (PC Workload Operational Component)**.** The operational component of the *program committee* (PC) workload is defined as follows. States in PC have the following

fields.

- S, the set of subjects
- O, the set of objects
- G, the set of groups
- T, the set of times
- $>_T$, the total order on T
- $Time \in T$, the current time
- $StrictJoin \subseteq S \times G \times T$, the record of strict join events
- $LiberalJoin \subseteq S \times G \times T$, the record of liberal join events
- $StrictLeave \subseteq S \times G \times T$, the record of strict leave events
- $LiberalLeave \subseteq S \times G \times T$, the record of liberal leave events
- $LiberalAdd \subseteq O \times G \times T$, the record of liberal add events

Commands in PC consist of the following forms.

- $addS(s)$: Add $S(s)$
- $delS(s)$: Remove $S(s)$
- $addG(g)$: Add $G(g)$
- $delG(g)$: Remove $G(g)$
- $addO(o)$: Add $O(o)$
- $strictJoin(s, g)$: Remove $Time(t)$, add $StrictJoin(s, g, t)$, $Time(t + 1)$
- $liberalJoin(s, g)$: Remove $Time(t)$, add $LiberalJoin(s, g, t)$, $Time(t + 1)$
- $strictLeave(s, g)$: Remove $Time(t)$, add $StrictLeave(s, g, t)$, $Time(t + 1)$
- $liberalLeave(s, g)$: Remove $Time(t)$, add $LiberalLeave(s, g, t)$, $Time(t + 1)$
- $liberalAdd(o, g)$: Remove $Time(t)$, add $LiberalAdd(o, g, t)$, $Time(t + 1)$

A request in PC is of the form $auth(s, o, g)$, which asks whether subject s has access to object o through group g. Other queries include $Member$, which asks whether a particular subject is currently a member of a particular group, $Assoc$, which asks whether a particular object is currently associated with a particular group. The entailments of these queries are defined as follows. In the case of $auth$, $authForward$ applies in cases where the subject

joined the group before the object was added, and *authBackward* applies when the subject joined after the object was added.

- $Join(s, g, t) \triangleq StrictJoin(s, g, t) \vee LiberalJoin(s, g, t)$
- $Leave(s, g, t) \triangleq StrictLeave(s, g, t) \vee LiberalLeave(s, g, t)$
- $Member(s, g) \triangleq \exists t_1.($
  $Join(s, g, t_1) \wedge$
  $\forall t_2.($
  $Leave(s, g, t_2) \Rightarrow t_1 > t_2$
  $)$
  $)$
- $Assoc(o, g) \triangleq \exists t_1.(LiberalAdd(o, g, t_1))$
- $authForward(s, o, g) \triangleq \exists t_1, t_2.($
  $Join(s, g, t_1) \wedge$
  $LiberalAdd(o, g, t_2) \wedge$
  $t_2 > t_1 \wedge$
  $\forall t_3.($
  $Leave(s, g, t_3) \Rightarrow (t_1 > t_3 \vee t_3 > t_2) \wedge$
  $StrictLeave(s, g, t_3) \Rightarrow t_2 > t_3$
  $)$
  $)$
- $authBackward(s, o, g) \triangleq \exists t_1, t_2.($
  $LiberalJoin(s, g, t_1) \wedge$
  $LiberalAdd(o, g, t_2) \wedge$
  $t_1 > t_2 \wedge$
  $\forall t_3.($
  $StrictLeave(s, g, t_3) \Rightarrow t_1 > t_3$
  $)$
  $)$
- $auth(s, o, g) \triangleq authForward(s, o, g) \vee authBackward(s, o, g)$ $\Diamond$

These capabilities naturally satisfy the requirements of the academic conference. When the program committee is formed, a *discussion group* is created, and each reviewer joins. Each paper is submitted to an *author group*, which holds the objects the author can see (initially, only the submitted paper). During the reviewing period, a *review group* is created for each paper, which the paper's reviewers join. Discussion about papers in contention takes place in the discussion group. When the group discusses a paper with which a reviewer has a conflict of interest, this reviewer will temporarily leave the discussion group (executing liberal leave to retain previous accesses and strict join to return without gaining access to the conflicted discussion).

We consider several role- and group-based candidate systems for implementing the conference workload. As such systems provide a level of indirection between subjects and objects, they are more likely to be effective at implementing the group-based conference workload than systems without this level of indirection (e.g., access control list systems). We choose widely-deployed candidate systems from both the industrial and consumer spaces, making them likely candidates for developing the type of system described by our conference workload. We evaluate RBAC_0 , RBAC_1, and *ugo*.

RBAC_0 is the most basic role-based access control system in the RBAC standard [103,104]. States contain the set of users U, set of roles R, and set of permissions P, as well as relations between them: $UR \subseteq U \times R$ describes users' membership in roles, and $PA \subseteq R \times P$ describes permissions' assignment to roles. A user u is authorized to permission p if and only if $\exists r.(\langle u, r\rangle \in UR \wedge \langle r, p\rangle \in PA)$. Commands allow adding to and removing from all of U, R, P, UR, and PA. RBAC_0 was formally defined in Example 2.

While RBAC_0 grants a level of indirection between users and permissions, RBAC_1 includes a hierarchical structure over roles to further extend this abstraction. RBAC_1 includes all state elements of RBAC_0 as well as the role hierarchy $RH \subseteq R \times R$, a binary relation over R whose transitive closure is the *Senior* partial order (we sometimes designate the transitive, reflexive closure $\geq$). In hierarchical RBAC, a user inherits all permissions from roles junior to roles she is explicitly assigned. That is, a user u is authorized to permission p if $\exists r_1, r_2.(\langle u, r_1\rangle \in UR \wedge \langle r_2, p\rangle \in PA \wedge r_1 \geq r_2)$. Commands allow manipulation of all state elements.

Definition 22 (RBAC_1 System)**.** The hierarchical role-based system, RBAC_1, is based on the system of the same name in the RBAC standard [103, 104]. States in RBAC_1 have the following fields.

- U, the set of users
- R, the set of roles
- P, the set of permissions
- $UR \subseteq U \times R$, the user-role relation
- $PA \subseteq R \times P$, the role-permission relation
- $RH \subseteq R \times R$, a partially ordered role hierarchy (written $\geq$ in infix notation)

Commands in RBAC_1 consist of the following forms.

- $addU(u)$: Add $U(u)$
- $delU(u)$: Remove $U(u)$
- $addR(r)$: Add $R(r)$
- $delR(r)$: Remove $R(r)$
- $addP(p)$: Add $P(p)$
- $delP(p)$: Remove $P(p)$
- $assignUser(u, r)$: Add $UR(u, r)$
- $revokeUser(u, r)$: Remove $UR(u, r)$
- $assignPermission(r, p)$: Add $PA(r, p)$
- $revokePermission(r, p)$: Remove $PA(r, p)$
- $addHierarchy(r_1, r_2)$: Add $RH(r_1, r_2)$
- $removeHierarchy(r_1, r_2)$: Remove $RH(r_1, r_2)$

A request in RBAC_1 is of the form $auth(u, p)$, which asks whether user u is granted permission p. Other queries include UR, which asks whether a particular user is currently a member of a particular role, PA, which asks whether a particular permission is currently assigned to a particular role, R, which asks whether a particular role exists, RH, which asks whether a particular role is the direct senior of another role, and $Senior$, which asks whether a particular role is a (general) senior of another role. The entailments of UR, PA, R, and

RH are defined simply by inspecting the corresponding state elements. The entailments of the remaining queries are defined as follows.

- $Senior(r_1, r_2) \triangleq RH(r_1, r_2) \vee \exists r_3.(Senior(r_1, r_3) \wedge Senior(r_3, r_2))$
- $auth(u, p) \triangleq \exists r_1, r_2.(UR(u, r_1) \wedge PA(r_2, p) \wedge (r_1 = r_2 \vee Senior(r_1, r_2)))$ $\Diamond$

Finally, the *ugo* system [42] is based on the traditional *user, group, other* permission system in UNIX. Thus, if RBAC_0 and RBAC_1 fill the need for a commonly-used industrial standard system, *ugo* fills the role of a common consumer system. In *ugo*, objects can be associated with an owner user and owner group, and permissions are then granted to the user, the group, or everyone else.

Definition 23 (*ugo* System)**.** UNIX's traditional user-group-other discretionary access control system, *ugo*, is defined as follows. States in *ugo* have the following fields.

- S, the set of subjects
- O, the set of objects
- G, the set of groups
- $R = \{read, write, execute\}$, the set of rights
- $Member \subseteq S \times G$, the group-membership relation
- $Owner : O \to S$, the object-ownership record
- $Group : O \to G$, the object-group-membership record
- $OwnerRight \subseteq O \times R$, the granted owner rights for objects
- $GroupRight \subseteq O \times R$, the granted group rights for objects
- $OtherRight \subseteq O \times R$, the granted global rights for objects

Commands in *ugo* consist of the following forms.

- $addS(s)$: Add $S(s)$
- $delS(s)$: Remove $S(s)$
- $addO(o)$: Add $O(o)$
- $delO(o)$: Remove $O(o)$
- $addG(g)$: Add $G(g)$
- $delG(g)$: Remove $G(g)$

- $changeOwner(o, s)$: Set $Owner(o) = s$
- $changeGroup(o, g)$: Set $Group(o) = g$
- $grantOwner(o, r)$: Add $OwnerRight(o, r)$
- $revokeOwner(o, r)$: Remove $OwnerRight(o, r)$
- $grantGroup(o, r)$: Add $GroupRight(o, r)$
- $revokeGroup(o, r)$: Remove $GroupRight(o, r)$
- $grantOther(o, r)$: Add $OtherRight(o, r)$
- $revokeOther(o, r)$: Remove $OtherRight(o, r)$

A request in RBAC_1 is of the form $auth(s, o, r)$, which asks whether subject s has access to object o with right r. Other queries include G, which asks whether a particular group exists, $Member$, which asks whether a particular subject is currently a member of a particular group, and $Group$, which asks whether a particular object is currently assigned to a particular group. The entailments of these queries are defined simply by inspecting the corresponding state elements. The entailment of $auth$ is defined as follows, where $OwnerAccess$ applies if authorization should consider the *owner* rule, $GroupAccess$ applies if authorization should consider the *group* rule, and $OtherAccess$ applies if the authorization should consider the *other* rule.

- $OwnerAccess(s, o) \triangleq Owner(o, s)$
- $GroupAccess(s, o) \triangleq \neg Owner(o, s) \wedge \exists g_1.(Group(o, g_1) \wedge Member(s, g_1))$
- $OtherAccess(s, o) \triangleq \neg Owner(o, s) \wedge \forall g_1.(\neg Group(o, g_1) \vee \neg Member(s, g_1))$
- $auth(s, o, r) \triangleq$

  $OwnerAccess(s, o) \wedge OwnerRight(o, r) \vee$

  $GroupAccess(s, o) \wedge GroupRight(o, r) \vee$

  $OtherAccess(s, o) \wedge OtherRight(o, r)$ $\Diamond$

4.5.2 Qualitative Analysis

In this section, we describe the qualitative analysis we conducted to ensure that each of our candidate systems is capable of satisfying the PC workload. Expressiveness reductions and proofs that are not detailed here are available in [47]. In this case study, we consider a fixed

notion of expressiveness reduction based on parameterized expressiveness (PE) [59] that has been shown to satisfy the implementability requirements [43]. In particular, we require a PE reduction that satisfies the following properties.

Correctness is a bare minimum requirement for an implementation in parameterized expressiveness. Intuitively, correctness says the following: a workload state's image in a system answers mapped queries exactly as the original state answers the original queries; and the same system state is reached by executing a workload action and mapping the result into the system or by mapping the initial state and executing the action's image in the system.

Definition 24 (Correctness)**.** Given a workload, $\mathcal{W} = \langle \mathcal{A}, \mathcal{T} \rangle$, a system $\mathcal{S}$, and an implementation $\langle \sigma_\Gamma, \sigma_\Psi, \sigma_Q \rangle$, the implementation is correct if the following conditions hold.

- σ_Γ preserves σ_Q: For every workload state γ, $Th(\gamma) = \sigma_Q(Th(\sigma_\Gamma(\gamma)))$
- σ_Ψ preserves σ_Γ: For every workload state γ and command ψ, $\sigma_\Gamma(next(\gamma, \psi)) = terminal(\sigma_\Gamma(\gamma), \sigma_\Psi(\psi, \sigma_\Gamma(\gamma)))$ ◊

Next, *AC-preservation* says that σ_Q must map authorization request r from workload state γ to system state $\sigma_\Gamma(\gamma)$ directly, checking whether $\sigma_\Gamma(\gamma) \vdash r$. This forces the workload and system to have identical sets of requests, which is not always the case (e.g., access matrix systems typically use subject-object-right triples, while role-based systems often use user-permission pairs). *Weak AC-preservation* captures the spirit of AC-preservation (ensures the use of the authorization procedure of the system) but allows us to define a request transformation function to map workload requests to system requests.

Definition 25 (Weak AC-Preservation)**.** Given a workload, $\mathcal{W} = \langle \mathcal{A}, \mathcal{T} \rangle$, a system, $\mathcal{S}$, and an implementation, $\langle \sigma_\Gamma, \sigma_\Psi, \sigma_Q \rangle$, σ_Q is *weak AC-preserving* if there exists a request transformation $f : \mathcal{R}^{\mathcal{A}} \to \mathcal{R}^{\mathcal{S}}$ such that for any workload state γ, workload request r, and system request r', the following conditions hold. This allows us to answer a workload authorization using the system's authorization procedure, even if their requests use different formats. Thus, we allow the use of a *request transformation* function f, so we can ask $auth(f(r))$ for some function f.

- $\sigma_Q(r, \sigma_\Gamma(\gamma)) = \text{TRUE} \Rightarrow \sigma_\Gamma(\gamma) \vdash f(r)$
- $\sigma_\Gamma(\gamma) \vdash r' \Rightarrow \exists r.(\sigma_Q(r, \sigma_\Gamma(\gamma)) = \text{TRUE} \wedge f(r) = r')$ ◊

Finally, a *safe* implementation is one in which intermediate states through which the system travels while implementing a single workload command do not add or remove authorized requests except as determined to be necessary by the start and end states.

Definition 26 (Safety)**.** Given a workload, $\mathcal{W} = \langle \mathcal{A}, \mathcal{T} \rangle$, a system, $\mathcal{S}$, and an implementation, $\langle \sigma_\Gamma, \sigma_\Psi, \sigma_Q \rangle$, the implementation is *safe* if, for any workload command ψ, and system state γ_0 if executing $\sigma_\Psi(\psi, \gamma_0$ results in the system state sequence $\langle \gamma_1, \ldots, \gamma_k \rangle$, then for all γ_i in the sequence:

- $Auth(\gamma_i) \setminus Auth(\gamma_0) \subseteq Auth(\gamma_k) \setminus Auth(\gamma_0)$, and
- $Auth(\gamma_0) \setminus Auth(\gamma_i) \subseteq Auth(\gamma_0) \setminus Auth(\gamma_n)$. $\Diamond$

We implement the conference workload in RBAC_1 (role-based access control with role hierarchy) using the following construction techniques. Members who strict join a group are granted a subset of the permissions of older members, a pattern we can mirror naturally using a role hierarchy. In a way, older members "inherit" access to all added objects; new members only receive access to objects added after they joined. We thus create a chain in the role hierarchy for each group. When a group is created, the top of the chain is created in RBAC_1. We name the top role of the chain after the group, and use this to correlate chains to groups.

When an object is added to a group, it is available to all members, and thus the corresponding permission in RBAC_1 is added to the bottom of the chain, where access to it will be inherited upward (to older members). When a user *liberal* joins a group, they gain access to all existing objects, and thus we add this user in RBAC_1 to the top of the role chain, where she will inherit permission corresponding to each object. When a new user *strict* joins a group, they create a new "view" of the group, since they are not authorized to any existing objects. Thus, we create a new RBAC_1 role (named arbitrarily) and link it to the bottom of the group's chain.

When a user *strict* leaves a group, she is removed from any roles in the group's role chain, losing access to all objects in the group. When a user *liberal* leaves a group, on the other hand, she should retain access to the current set of objects. Thus, we create an *orphan role*, which does not inherit any permissions from other roles, and from which no users inherit permissions. Then, the leaving user is added to the orphan role, and the role

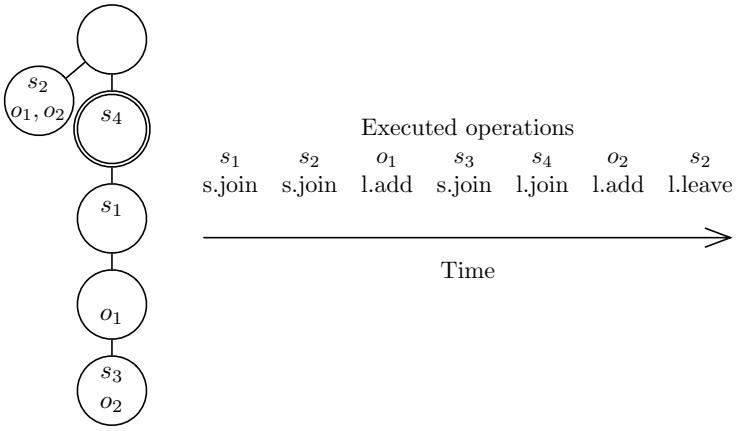


Figure 4: An example role hierarchy implementing the PC workload in RBAC_1

is granted access to the group's objects to which the user currently has permission. Then, when the user is removed from the main role chain, she does not lose accesses. We give a demonstrative example of the role hierarchy structure in Fig. 4. This example displays each of the aforementioned operations, and shows an orphan role created for s_2 after a liberal leave.

Using this technique, we can implement the conference workload in RBAC_1 while preserving correctness, weak AC-preservation, and safety.

Theorem 2. *There exists a correct, weak AC-preserving, and safe implementation of the conference workload in* RBAC_1.

Proof. We present the implementation, $\langle \sigma_\Gamma, \sigma_\Psi, \sigma_Q \rangle$, and show that σ_Ψ preserves σ_Γ and preserves safety, σ_Γ preserves σ_Q, and σ_Q is weak AC-preserving.

First, σ_Γ maps each PC state $\langle S, O, G, T, Time, StrictJoin, LiberalJoin, StrictLeave, LiberalLeave, LiberalAdd \rangle$ to an RBAC_1 state $\langle U, R, P, UR, PA, RH \rangle$. This mapping is described as follows.

```
state_mapping(γ)
   Let WildRoles = {}
   Let UR = {}
```

```
Let PA = {}
Let RH = {}

for each (S(s) ∈ γ)
   output(U(s))
for each (G(g) ∈ γ)
   InitGroup(γ, g, RH, WildRoles)
for each (O(o) ∈ γ)
   output(P(o))

Let Records = sortByTime(StrictJoin ∪ LiberalJoin ∪
                         StrictLeave ∪ LiberalLeave ∪
                         LiberalAdd)
for each (Record ∈ Records)
   If ∃ s, g, t.(Record = <s, g, t> ∧
                 StrictJoin(s, g, t) ∈ γ)
      ProcessSJoin(γ, s, g, UR, RH, WildRoles)
   else If ∃ s, g, t.(Record = <s, g, t> ∧
                      LiberalJoin(s, g, t) ∈ γ)
      ProcessLJoin(γ, s, g)
   else If ∃ s, g, t.(Record = <s, g, t> ∧
                      StrictLeave(s, g, t) ∈ γ)
      ProcessSLeave(γ, s, g, UR, RH)
   else If ∃ s, g, t.(Record = <s, g, t> ∧
                      LiberalLeave(s, g, t) ∈ γ)
      ProcessLLeave(γ, s, g, UR, PA, RH, WildRoles)
   else If ∃ o, g, t.(Record = <o, g, t> ∧
                      LiberalAdd(o, g, t) ∈ γ)
      ProcessLAdd(γ, o, g, PA, RH)
   endif
```

```
   outputSet(UR ∪ PA ∪ RH)

InitGroup(γ, g, RH, WildRoles)
   output(R(g))
   <Top, Bottom> = nFreshConst(2, Consts(γ) ∪ WildRoles,
                               Univ)
   WildRoles = WildRoles ∪ {Top, Bottom}
   output(R(Top))
   output(R(Bottom))
   RH = RH ∪ {<Top, g>, <g, Bottom>}

ProcessSJoin(γ, s, g, UR, RH, WildRoles)
   NewBottom = nFreshConst(1, Consts(γ) ∪ WildRoles,
                           Univ)
   WildRoles = WildRoles ∪ {NewBottom}
   OldBottom = FindBottom(g, RH)
   output(R(NewBottom))
   RH = RH ∪ {<OldBottom, NewBottom>}
   UR = UR ∪ {<s, NewBottom>}

FindBottom(r, RH)
   If ∃ q.(<r, q> ∈ RH)
      return FindBottom(q, RH)
   else
      return r
   endif

ProcessLJoin(γ, s, g)
   UR = UR ∪ {<s, g>}
```

```
ProcessSLeave(γ, s, g, UR, RH)
   LeaveDown(s, g, UR, RH)
   LeaveOrphans(s, g, UR, RH)

LeaveDown(u, r, UR, RH)
   UR = UR \ {<u, r>}
   If ∃ q.(<r, q> ∈ RH)
      LeaveDown(u, q, UR, RH)
   endif

LeaveOrphans(u, g, UR, RH)
   for each (Orphan in {r | ∃ Head.(<Head, g> ∈ RH ∧
                                    <Head, r> ∈ RH)})
      UR = UR \ {<u, Orphan>}

ProcessLLeave(γ, s, g, UR, PA, RH, WildRoles)
   Orphan = nFreshConst(1, Consts(γ) ∪ WildRoles, Univ)
   WildRoles = WildRoles ∪ {Orphan}
   output(R(Orphan))
   Permissions = PermsInChain(s, g, UR, PA, RH)
   Top = FindTop(g, RH)
   for each (Permission ∈ Permissions)
      PA = PA ∪ {<Orphan, Permission>}
   UR = UR ∪ {<s, Orpan>}
   RH = RH ∪ {<Top, Orphan>}
   LeaveDown(s, g, UR, RH)

PermsInChain(u, r, UR, PA, RH)
   If <u, r> ∈ UR
      return PermsBelow(r, PA, RH, {})
```

```
   else If ∃ q.(<r, q> ∈ RH)
      return PermsInChain(u, q, UR, PA, RH)
   else
      return {}
   endif

PermsBelow(r, PA, RH, Perms)
   Perms = Perms ∪ {p | <r, p> ∈ PA}
   If ∃ q.(<r, q> ∈ RH)
      return PermsBelow(q, PA, RH, Perms)
   else
      return Perms
   endif

FindTop(r, RH)
   If ∃ q.(<q, r> ∈ RH)
      return FindTop(q)
   else
      return r
   endif

ProcessLAdd(γ, o, g, PA, RH)
   Bottom = FindBottom(g, RH)
   PA = PA ∪ {<Bottom, o>}
```

The query mapping, σ_Q, is defined as follows.

$$\begin{aligned}
\sigma_Q(Member(s,g),\gamma) &= UR(s,g) \in \gamma \vee \exists r.(UR(s,r) \in \gamma \wedge Senior(g,r) \in \gamma) \\
\sigma_Q(Assoc(o,g),\gamma) &= \exists r.(PA(r,o) \in \gamma \wedge Senior(g,r) \in \gamma) \\
\sigma_Q(auth(s,o,g),\gamma) &= \exists r_1, r_2.(UR(s,r_1) \in \gamma \wedge PA(r_2,o) \in \gamma \wedge \\
&\qquad (r_1 = r_2 \vee Senior(r_1,r_2) \in \gamma) \wedge \\
&\qquad \exists r_3.(Senior(r_3,g) \in \gamma \wedge Senior(r_3,r_2) \in \gamma))
\end{aligned}$$

Let γ be an arbitrary PC state and $v = auth(s,o,g)$ an arbitrary PC request, and let $f(auth(s,o,g)) = auth(s,o)$ be a request transform. Assume $\sigma_Q(v, \sigma_\Gamma(\gamma)) = \text{TRUE}$. Then, by σ_Q, $\exists r_1, r_2.(r_1 \geq r_2 \wedge UR(s,r_1) \in Th(\sigma_\Gamma(\gamma)) \wedge PA(r_2,o) \in Th(\sigma_\Gamma(\gamma)))$. Thus, by RBAC_1's $\vdash$ relation, $\sigma_\Gamma(\gamma) \vdash auth(s,o)$.

Now let γ be an arbitrary PC state, $v' = auth(u,p)$ an arbitrary RBAC_1 request, and f the request transform defined above. Assume $\sigma_\Gamma(\gamma) \vdash v'$. Then, $\exists r_1, r_2.(r_1 \geq r_2 \wedge UR(u, r_1) \in Th(\sigma_\Gamma(\gamma)) \wedge PA(r_2,p) \in Th(\sigma_\Gamma(\gamma)))$. Furthermore, since σ_Γ only assigns permissions to role which correspond to some group, r_2 must exist either in the hierarchy below a role corresponding to a group, or as an orphan node attached to such a role: $\exists r_3.(Senior(r_3, g) \in Th(\sigma_\Gamma(\gamma)) \wedge Senior(r_3,r_2) \in Th(\sigma_\Gamma(\gamma)))$. Finally, $f(auth(u,p,g)) = auth(u,p)$, and $\sigma_Q(auth(u,p,g), \sigma_\Gamma(\gamma)) = \text{TRUE}$. Thus, σ_Q is weak AC-preserving with transform $f(auth(s,o,g)) = auth(s,o)$.

We show that σ_Γ preserves σ_Q (for all PC states γ, $Th(\gamma) = \sigma_Q(Th(\sigma_\Gamma(\gamma))))$ by contradiction. Assume that there is some PC state γ and query q such that the value of q in γ is the opposite of the value of $\sigma_Q(q)$ in $\sigma_\Gamma(\gamma)$. We show that, for each of the query forms of PC, this assumption leads to contradiction.

- **Member** Assume $\gamma \vdash Member(s,g)$ and $\sigma_\Gamma(\gamma) \nvdash \sigma_Q(Member(s,g))$. Then, $\exists t_1.(Join(s, g, t_1) \in Th(\gamma) \wedge \forall t_2.(Leave(s,g,t_2) \in Th(\gamma) \Rightarrow t_1 > t_2))$ (s has joined g and not left). By σ_Γ, if the $Join$ is a $LiberalJoin$, then $UR(s,g) \in Th(\sigma_\Gamma(\gamma))$. If the $Join$ is a $StrictJoin$, then $\exists r_1.(UR(s,r_1) \in Th(\sigma_\Gamma(\gamma)) \wedge Senior(g,r_1) \in Th(\sigma_\Gamma(\gamma)))$. By σ_Q, in either case, $\sigma_Q(Member(s,g), \sigma_\Gamma(\gamma)) = \text{TRUE}$, which is a contradiction on the assumption that $\sigma_\Gamma(\gamma) \nvdash \sigma_Q(Member(s,g))$.

Assume instead that $\gamma \nvdash Member(s,g)$ and $\sigma_\Gamma(\gamma) \vdash \sigma_Q(Member(s,g))$. Then, either $\exists t_1.(Leave(s,g,t_1) \in Th(\gamma) \wedge \forall t_2.(Join(s,g,t_2) \in Th(\gamma) \Rightarrow t_1 > t_2))$ (s has left g and not returned), or $\forall t_1.(Join(s,g,t_1) \notin Th(\gamma))$ (s has not joined g). By σ_Γ, in either case, $UR(s,g) \notin Th(\sigma_\Gamma(\gamma)) \wedge \forall r_1.(Senior(g,r_1) \notin Th(\sigma_\Gamma(\gamma)) \vee UR(s,r_1) \notin Th(\sigma_\Gamma(\gamma)))$. By σ_Q, $\sigma_Q(Member(o,g), \sigma_\Gamma(\gamma)) = \text{FALSE}$, which is a contradiction on the assumption that $\sigma_\Gamma(\gamma) \vdash \sigma_Q(Member(s,g))$.

- **Assoc** Assume $\gamma \vdash Assoc(o,g)$ and $\sigma_\Gamma(\gamma) \nvdash \sigma_Q(Assoc(o,g))$. Then, $\exists t_1.(LiberalAdd(o, g, t_1) \in Th(\gamma))$ (o was added to g). By σ_Γ, $\exists r_1.(Senior(g,r_1) \in Th(\sigma_\Gamma(\gamma)) \wedge PA(r_1, o) \in Th(\sigma_\Gamma(\gamma)))$. By σ_Q, $\sigma_Q(Assoc(o,g), \sigma_\Gamma(\gamma)) = \text{TRUE}$, which is a contradiction on the assumption that $\sigma_\Gamma(\gamma) \nvdash \sigma_Q(Assoc(o,g))$.

  Assume instead that $\gamma \nvdash Assoc(o,g)$ and $\sigma_\Gamma(\gamma) \vdash \sigma_Q(Assoc(o,g))$. Then, $\forall t_1.(LiberalAdd(o,g,t_1) \notin Th(\gamma))$ (o has not added to g). By σ_Γ, $\forall r_1.(Senior(g, r_1) \notin Th(\sigma_\Gamma(\gamma)) \vee PA(r_1,o) \notin Th(\sigma_\Gamma(\gamma)))$. By σ_Q, $\sigma_Q(Assoc(o,g), \sigma_\Gamma(\gamma)) = \text{FALSE}$, which is a contradiction on the assumption that $\sigma_\Gamma(\gamma) \vdash \sigma_Q(Assoc(o,g))$.

- **auth** Assume $\gamma \vdash auth(s,o,g)$ and $\sigma_\Gamma(\gamma) \nvdash \sigma_Q(auth(s,o,g))$. Then, $\exists t_1, t_2.(Join(s,g, t_1) \in Th(\gamma) \wedge LiberalAdd(o,g,t_2) \in Th(\gamma) \wedge \forall t_3.(StrictLeave(s,g,t_3) \in Th(\gamma) \Rightarrow t_1 > t_3))$ (s has joined g and not strict left; o has been added to g). If $t_2 > t_1$ (the join occurred first), then $\forall t_4.(Leave(s,g,t_4) \in Th(\gamma) \Rightarrow t_1 > t_4 \vee t_4 > t_2)$ (s did not leave g between joining and o being added). If $t_1 > t_2$ (the add occurred first), then s's join must be a liberal join. In either case, by σ_Γ, $\exists r_1, r_2.(UR(s,r_1) \in Th(\sigma_\Gamma(\gamma)) \wedge PA(r_2, o) \in Th(\sigma_\Gamma(\gamma)) \wedge r_1 \geq r_2 \in Th(\sigma_\Gamma(\gamma)))$ (s belongs to a role authorized to o or senior to a role authorized to o). Connection to g is preserved by σ_Γ, so $\exists r_3.(Senior(r_3, g) \in Th(\sigma_\Gamma(\gamma)) \wedge Senior(r_3, r_2) \in Th(\sigma_\Gamma(\gamma)))$, either because s and o are in the hierarchy below g or because s and o are in an "orphaned node" due to o's removal from g. Thus, by σ_Q, $\sigma_Q(auth(s,o,g), \sigma_\Gamma(\gamma)) = \text{TRUE}$, which is a contradiction on the assumption that $\sigma_\Gamma(\gamma) \nvdash \sigma_Q(auth(s,o,g))$.

  Assume instead that $\gamma \nvdash auth(s,o,g)$ and $\sigma_\Gamma(\gamma) \vdash \sigma_Q(auth(s,o,g))$. Then, either: $\forall t_1, t_2.(Join(s,g,t_1) \notin Th(\gamma) \vee LiberalAdd(o,g,t_2) \notin Th(\gamma))$ (s has not joined g or o has not been added to g); $\exists t_3.(StrictLeave(s,g,t_3) \in Th(\gamma) \wedge t_3 > t_1)$ (s has since strict left g); or s's and o's membership in g did not overlap in a way that caused the

authorization. In the final case, if $t_2 > t_1$ (the join occurred first), then $\exists t_4.(Leave(s, g, t_4) \in Th(\gamma) \wedge t_2 > t_4 > t_1)$ (s left g before o was added). If $t_1 > t_2$ (the add occurred first), then $Join(s, g, t_1)$ must be $StrictJoin(s, g, t_1)$. Thus, by σ_Γ, if $\exists r_1, r_2.(UR(s, r_1) \in Th(\sigma_\Gamma(\gamma)) \wedge PA(r_2, o) \in Th(\sigma_\Gamma(\gamma)) \wedge r_1 \geq r_2 \in Th(\sigma_\Gamma(\gamma)))$ (s belongs to a role authorized to o or senior to a role authorized to o), then it must be in conjunction with a group other than g: $\forall r_3.(Senior(r_3, g) \notin Th(\sigma_\Gamma(\gamma)) \vee Senior(r_3, r_2) \notin Th(\sigma_\Gamma(\gamma)))$. Thus, by σ_Q, $\sigma_Q(auth(s, o, g), \sigma_\Gamma(\gamma)) =$ FALSE, which is a contradiction on the assumption that $\sigma_\Gamma(\gamma) \vdash \sigma_Q(auth(s, o, g)$.

Thus, by contradiction, σ_Γ preserves σ_Q.

Finally, the command mapping, σ_Ψ, is defined as follows.

```
addS(γ, s)
   output(addU(s))

delS(γ, s)
   output(delU(s))

addG(γ, g)
   output(addR(g))
   <Top, Bottom> = nFreshConst(2, Consts(γ), Univ)
   output(addR(Top))
   output(addR(Bottom))
   output(addHierarchy(Top, g))
   output(addHierarchy(g, Bottom))

delG(γ, g)
   If ∃ Head.(RH(Head, g) ∈ γ)
      DeleteDown(γ, g)
      DeleteOrphans(γ, Head)
      output(delR(Head))
```

```
DeleteDown(γ, r)
   If ∃ q.(RH(r, q) ∈ γ)
      DeleteDown(γ, q)
   endif
   output(delR(r))

DeleteOrphans(γ, Head)
   for each (Orphan ∈ {Role | RH(Head, Role) ∈ γ}
      output(delR(Orphan))

addO(γ, o)
   output(addP(o))

strictJoin(γ, s, g)
   NewBottom = nFreshConst(1, Consts(γ), Univ)
   OldBottom = FindBottom(γ, g)
   output(addR(NewBottom))
   output(addHierarchy(OldBottom, NewBottom))
   output(assignUser(s, NewBottom))

FindBottom(γ, r)
   If ∃ q.(RH(r, q) ∈ γ)
      return FindBottom(γ, q)
   else
      return r
   endif

liberalJoin(γ, s, g)
   output(assignUser(s, g))
```

```
strictLeave(γ, s, g)
   LeaveDown(γ, s, g)
   LeaveOrphans(γ, s, g)

LeaveDown u, r)
   If UR(u, r) ∈ γ
      output(revokeUser(u, r))
   endif
   If ∃ q.(RH(r, q) ∈ γ)
      LeaveDown(γ, u, q)
   endif

LeaveOrphans(γ, u, g)
   for each (Orphan ∈ {r | ∃ Head.(RH(Head, g) ∈ γ ∧
                                   RH(Head, r) ∈ γ)})
      If UR(u, Orphan) ∈ γ
         output(revokeUser(u, Orphan))
      endif

liberalLeave(γ, s, g)
   Orphan = nFreshConst(1, Consts(γ), Univ)
   output(addR(Orphan))
   Top = FindTop(γ, r)
   for each (Permission ∈ PermsInChain(γ, s, g))
      output(assignPermission(Orphan, Permission))
   output(assignUser(s, Orphan))
   output(addHierarchy(Top, Orphan))
   LeaveDown(γ, s, g)
```

```
FindTop(γ, r)
   If ∃ q.(RH(q, r) ∈ γ)
      return FindTop(γ, q)
   else
      return r
   endif

PermsInChain(γ, u, r)
   If UR(u, r) ∈ γ
      return PermsBelow(γ, r, {})
   else If ∃ q.(RH(r, q) ∈ γ)
      return PermsInChain(γ, u, q)
   else
      return {}
   endif

PermsBelow(γ, r, Perms)
   Perms = Perms ∪ {p | PA(r, p) ∈ γ}
   If ∃ q.(RH(r, q) ∈ γ)
      return PermsBelow(γ, q, Perms)
   else
      return Perms
   endif

liberalAdd(γ, o, g)
   Bottom = FindBottom(γ, g)
   output(assignPermission(Bottom, o))
```

We prove that σ_Ψ preserves σ_Γ by showing that, for any PC state γ and command ψ, $\sigma_\Gamma(next(\gamma, \psi)) = terminal(\sigma_\Gamma(\gamma), \sigma_\Psi(\psi, \sigma_\Gamma(\gamma)))$.

Given PC state γ and command ψ, $\gamma' = next(\gamma, \psi)$ is the state resulting from executing command ψ in state γ.

- If ψ is an instance of $addS(s)$, then $\gamma' = \gamma \cup S(s)$. By σ_Γ, this maps in RBAC_1 to state $\sigma_\Gamma(\gamma') = \sigma_\Gamma(\gamma) \cup U(s)$. By σ_Ψ, $\sigma_\Psi(\psi, \sigma_\Gamma(\gamma)) = addU(s)$. By RBAC_1's *next* relation, $next(\sigma_\Gamma(\gamma), addU(s)) = \sigma_\Gamma(\gamma) \cup U(s)$. Thus, if ψ is an instance of $addS(s)$, $\sigma_\Gamma(\gamma') = terminal(\sigma_\Gamma(\gamma), \sigma_\Psi(\psi, \sigma_\Gamma(\gamma)))$.
- If ψ is an instance of $delS(s)$, then $\gamma' = \gamma \setminus (S(s) \cup Entries(\gamma, s))$, where $Entries(\gamma, s)$ denotes the set of all state tuples in γ involving s. By σ_Γ, $\sigma_\Gamma(\gamma') = \sigma_\Gamma(\gamma) \setminus (U(s) \cup Entries(\sigma_\Gamma(\gamma), s))$. By σ_Ψ, $\sigma_\Psi(\psi, \sigma_\Gamma(\gamma)) = delU(s)$. By RBAC_1's *next* relation, $next(\sigma_\Gamma(\gamma), delU(s)) = \sigma_\Gamma(\gamma) \setminus (U(s) \cup Entries(\sigma_\Gamma(\gamma), s))$. Thus, if ψ is an instance of $delS(s)$, $\sigma_\Gamma(\gamma') = terminal(\sigma_\Gamma(\gamma), \sigma_\Psi(\psi, \sigma_\Gamma(\gamma)))$.
- If ψ is an instance of $addG(g)$, then $\gamma' = \gamma \cup G(g)$. By σ_Γ, $\sigma_\Gamma(\gamma') = \sigma_\Gamma(\gamma) \cup R(g) \cup R(r_{top}) \cup R(r_{bottom}) \cup RH(r_{top}, g) \cup RH(g, r_{bottom})$, where r_{top} and r_{bottom} are newly-created roles. By σ_Ψ, $\sigma_\Psi(\psi, \sigma_\Gamma(\gamma)) = addR(g) \circ addR(r_{top}) \circ addR(r_{bottom}) \circ addHierarchy(r_{top}, g) \circ addHierarchy(g, r_{bottom})$. By RBAC_1's *next* relation, $terminal(\sigma_\Gamma(\gamma), addR(g) \circ addR(r_{top}) \circ addR(r_{bottom}) \circ addHierarchy(r_{top}, g) \circ addHierarchy(g, r_{bottom})) = \sigma_\Gamma(\gamma) \cup R(g) \cup R(r_{top}) \cup R(r_{bottom}) \cup RH(r_{top}, g) \cup RH(g, r_{bottom})$. Thus, if ψ is an instance of $addG(g)$, $\sigma_\Gamma(\gamma') = terminal(\sigma_\Gamma(\gamma), \sigma_\Psi(\psi, \sigma_\Gamma(\gamma)))$.
- If ψ is an instance of $delG(g)$, then $\gamma' = \gamma \setminus (G(g) \cup Entries(\gamma, g))$. By σ_Γ, $\sigma_\Gamma(\gamma') = \sigma_\Gamma(\gamma) \setminus (R(g) \cup ConnectedEntries(\sigma_\Gamma(\gamma), g))$, where $ConnectedEntries(\sigma_\Gamma(\gamma), g)$ denotes the set of state tuples in $\sigma_\Gamma(\gamma)$ involving either g or any role connected to g in the role hierarchy of $\sigma_\Gamma(\gamma)$ (i.e., $ConnectedEntries(\gamma, r) \triangleq r \cup Entries(\gamma, r) \cup \{ConnectedEntries(\gamma, q) \mid RH(r, q) \in Th(\gamma) \vee RH(q, r) \in Th(\gamma)\}$). By σ_Ψ, $\sigma_\Psi(\psi, \sigma_\Gamma(\gamma)) = delR(g) \circ delR(r_1) \circ \cdots \circ delR(r_k)$, where $r_1, \ldots, r_k$ is the (finite) set of roles connected to g in the role hierarchy. By RBAC_1's *next* relation, $terminal(\sigma_\Gamma(\gamma), delR(g) \circ delR(r_1) \circ \cdots \circ delR(r_k)) = \sigma_\Gamma(\gamma) \setminus (R(g) \cup ConnectedEntries(\sigma_\Gamma(\gamma), g))$. Thus, if ψ is an instance of $delG(g)$, $\sigma_\Gamma(\gamma') = terminal(\sigma_\Gamma(\gamma), \sigma_\Psi(\sigma_\Gamma(\gamma), \psi))$.
- If ψ is an instance of $addO(o)$, then $\gamma' = \gamma \cup O(o)$. By σ_Γ, $\sigma_\Gamma(\gamma') = \sigma_\Gamma(\gamma) \cup P(o)$. By σ_Ψ, $\sigma_\Psi(\psi, \sigma_\Gamma(\gamma)) = addP(o)$. By RBAC_1's *next* relation, $next(\sigma_\Gamma(\gamma), addP(o)) = \sigma_\Gamma(\gamma) \cup P(o)$. Thus, if ψ is an instance of $addP(o)$, $\sigma_\Gamma(\gamma') = terminal(\sigma_\Gamma(\gamma), \sigma_\Psi(\psi, \sigma_\Gamma(\gamma)))$.

- If ψ is an instance of $strictJoin(s, g)$, then $\gamma' = \gamma \cup StrictJoin(s, g, t) \cup Time(t+1) \setminus Time(t)$. By σ_Γ, $\sigma_\Gamma(\gamma') = \sigma_\Gamma(\gamma) \cup R(r_{new}) \cup RH(r_{bottom}, r_{new}) \cup UR(s, r_{new})$, where r_{bottom} is the current bottom of the hierarchy chain below g and r_{new} is a newly-created role. By σ_Ψ, $\sigma_\Psi(\psi, \sigma_\Gamma(\gamma)) = addR(r_{new}) \circ addHierarchy(r_{bottom}, r_{new}) \circ assignUser(s, r_{new})$. By RBAC_1's *next* relation, $terminal(\sigma_\Gamma(\gamma), addR(r_{new}) \circ addHierarchy(r_{bottom}, r_{new}) \circ assignUser(s, r_{new})) = \sigma_\Gamma(\gamma) \cup R(r_{new}) \cup RH(r_{bottom}, r_{new}) \cup UR(s, r_{new})$. Thus, if ψ is an instance of $strictJoin(s, g)$, $\sigma_\Gamma(\gamma') = terminal(\sigma_\Gamma(\gamma), \sigma_\Psi(\psi, \sigma_\Gamma(\gamma)))$.
- If ψ is an instance of $liberalJoin(s, g)$, then $\gamma' = \gamma \cup LiberalJoin(s, g, t) \cup Time(t+1) \setminus Time(t)$. By σ_Γ, $\sigma_\Gamma(\gamma') = \sigma_\Gamma(\gamma) \cup UR(s, g)$. By σ_Ψ, $\sigma_\Psi(\psi, \sigma_\Gamma(\gamma)) = assignUser(s, g)$. By RBAC_1's *next* relation, $next(\sigma_\Gamma(\gamma), assignUser(s, g)) = \sigma_\Gamma(\gamma) \cup UR(s, g)$. Thus, if ψ is an instance of $liberalJoin(s, g)$, $\sigma_\Gamma(\gamma') = terminal(\sigma_\Gamma(\gamma), \sigma_\Psi(\psi, \sigma_\Gamma(\gamma)))$.
- If ψ is an instance of $strictLeave(s, g)$, then $\gamma' = \gamma \cup StrictLeave(s, g, t) \cup Time(t+1) \setminus Time(t)$. By σ_Γ, $\sigma_\Gamma(\gamma') = \sigma_\Gamma(\gamma) \setminus (UR(s, g) \cup UR(s, r_1) \cup \cdots \cup UR(s, r_k))$, where $r_1, \ldots, r_k$ is the set of roles to which s belongs and which are also connected in the role hierarchy to g. By σ_Ψ, $\sigma_\Psi(\psi, \sigma_\Gamma(\gamma)) = revokeUser(s, g) \circ revokeUser(s, r_1) \circ \cdots \circ revokeUser(s, r_k)$. By RBAC_1's *next* relation, $terminal(\sigma_\Gamma(\gamma), revokeUser(s, g) \circ revokeUser(s, r_1) \circ \cdots \circ revokeUser(s, r_k)) = \sigma_\Gamma(\gamma) \setminus (UR(s, g) \cup UR(s, r_1) \cup \cdots \cup UR(s, r_k))$. Thus, if ψ is an instance of $strictLeave(s, g)$, $\sigma_\Gamma(\gamma') = terminal(\sigma_\Gamma(\gamma), \sigma_\Psi(\psi, \sigma_\Gamma(\gamma)))$.
- If ψ is an instance of $liberalLeave(s, g)$, then $\gamma' = \gamma \cup liberalLeave(s, g, t) \cup Time(t+1) \setminus Time(t)$. By σ_Γ, $\sigma_\Gamma(\gamma') = \sigma_\Gamma(\gamma) \cup R(r_{orphan}) \cup PA(r_{orphan}, p_1) \cup \cdots \cup PA(r_{orphan}, p_k) \cup UR(s, r_{orphan}) \cup RH(r_{head}, r_{orphan}) \setminus (UR(s, r_1) \cup \cdots \cup UR(s, r_l))$, where $p_1, \ldots, p_k$ is the set of permissions to which s is authorized in g, and $r_1, \ldots, r_l$ is the set of roles to which s is authorized in the role hierarchy chain below g. By σ_Ψ, $\sigma_\Psi(\psi, \sigma_\Gamma(\gamma)) = addR(r_{orphan}) \circ assignPermission(r_{orphan}, p_1) \circ \cdots \circ assignPermission(r_{orphan}, p_k) \circ assignUser(s, r_{orphan}) \circ addHierarchy(r_{head}, r_{orphan}) \circ revokeUser(s, r_1) \circ \cdots \circ revokeUser(s, r_l)$. By RBAC_1's *next* relation, $terminal(\sigma_\Gamma(\gamma), addR(r_{orphan}) \circ assignPermission(r_{orphan}, p_1) \circ \cdots \circ assignPermission(r_{orphan}, p_k) \circ assignUser(s, r_{orphan}) \circ addHierarchy(r_{head}, r_{orphan}) \circ revokeUser(s, r_1) \circ \cdots \circ revokeUser(s, r_l)) = \sigma_\Gamma(\gamma) \cup R(r_{orphan}) \cup PA(r_{orphan}, p_1) \cup \cdots \cup PA(r_{orphan}, p_k) \cup UR(s, r_{orphan}) \cup RH(r_{head}, r_{orphan}) \setminus (UR(s, r_1) \cup \cdots \cup UR(s, r_l))$. Thus, if ψ is an instance of $liberalLeave(s, g)$, $\sigma_\Gamma(\gamma') = terminal(\sigma_\Gamma(\gamma), \sigma_\Psi(\psi, \sigma_\Gamma(\gamma)))$.

- If ψ is an instance of $liberalAdd(o, g)$, then $\gamma' = \gamma \cup LiberalAdd(o, g, t) \cup Time(t+1) \setminus Time(t)$. By σ_Γ, $\sigma_\Gamma(\gamma') = \sigma_\Gamma(\gamma) \cup PA(r_{bottom}, o)$, where r_{bottom} is the bottom role of the role hierarchy chain below g. By σ_Ψ, $\sigma_\Psi(\psi, \sigma_\Gamma(\gamma)) = assignPermission(r_{bottom}, o)$. By RBAC_1's *next* relation, $next(\sigma_\Gamma(\gamma), assignPermission(r_{bottom}, o)) = \sigma_\Gamma(\gamma) \cup PA(r_{bottom}, o)$. Thus, if ψ is an instance of $liberalAdd(o, g)$, $\sigma_\Gamma(\gamma') = terminal(\sigma_\Gamma(\gamma), \sigma_\Psi(\psi, \sigma_\Gamma(\gamma)))$.

Thus, for any PC state γ and command ψ, $\sigma_\Gamma(next(\gamma, \psi)) = terminal(\sigma_\Gamma(\gamma), \sigma_\Psi(\psi, \sigma_\Gamma(\gamma)))$. Thus, we have shown that σ_Ψ preserves σ_Γ.

Finally, σ_Ψ is safe by inspection—for any PC state γ and command ψ, the sequence of RBAC_1 commands $\sigma_\Psi(\psi, \sigma_\Gamma(\gamma))$ never revokes or grants authorizations except the images of those that are revoked or granted by ψ.

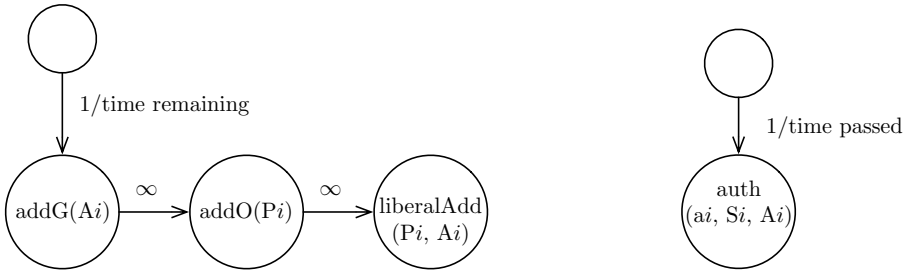
Thus, we have shown that σ_Ψ preserves σ_Γ, and preserves safety; that σ_Γ preserves σ_Q; and that σ_Q is weak AC-preserving.

$\therefore \langle \sigma_\Gamma, \sigma_\Psi, \sigma_Q \rangle$ is an implementation of PC in RBAC_1 which preserves correctness, weak AC-preservation, and safety. □

To implement the workload in RBAC_0 (role-based access control without role hierarchy), we follow the same procedure, but store the role hierarchy encoded in role names. This expands the set of roles to include a role named for every path through the logical hierarchy in the downward direction. Thus, if in RBAC_1 we would store a hierarchy that says $A \geq B$, $B \geq C$, and $A \geq D$, we represent this in RBAC_0 with roles $\{A, B, C, D, AB, ABC, AD, BC\}$. For every role r a user would be assigned to in RBAC_1, she will be assigned to each role starting with r in RBAC_0. In the previous example, if $\langle u, A \rangle \in UR$ in RBAC_1, then in RBAC_0 this maps to $\{\langle u, A \rangle, \langle u, AB \rangle, \langle u, ABC \rangle, \langle u, AD \rangle\} \subset UR$ in RBAC_0. This allows us to implement the conference workload in RBAC_0 while preserving our chosen implementation guarantees.

Theorem 3. *There exists a correct, weak AC-preserving, and safe implementation of the conference workload in* RBAC_0.

Finally, although *ugo* has the inherent disadvantage that each object is owned by only a *single* user and group, it can implement the conference workload by mapping a workload object assigned to multiple groups to an object with a single group owner. This group then



(a) Submit (b) Notify

Figure 5: Program Committee invocation: Author actor machines

represents all groups with authorization and includes as members all users with access. Of course, this incurs a storage penalty; the magnitude of this overhead will be explored in quantitative analysis.

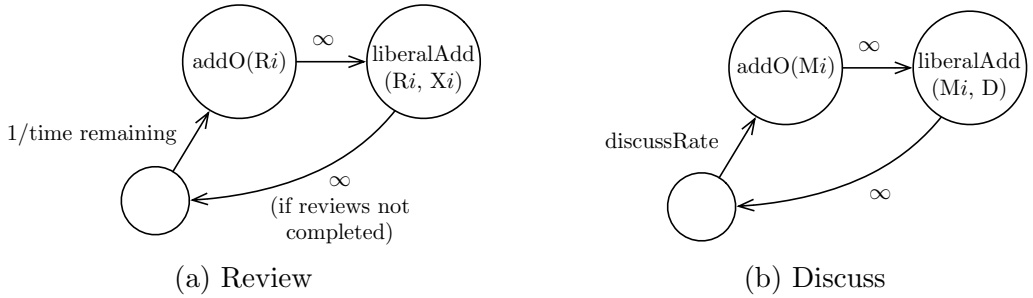
Theorem 4. *There exists a correct, weak AC-preserving, and safe implementation of the conference workload in ugo.*

The proofs of Theorems 3 and 4 can be found in [47].

4.5.3 Quantitative Analysis

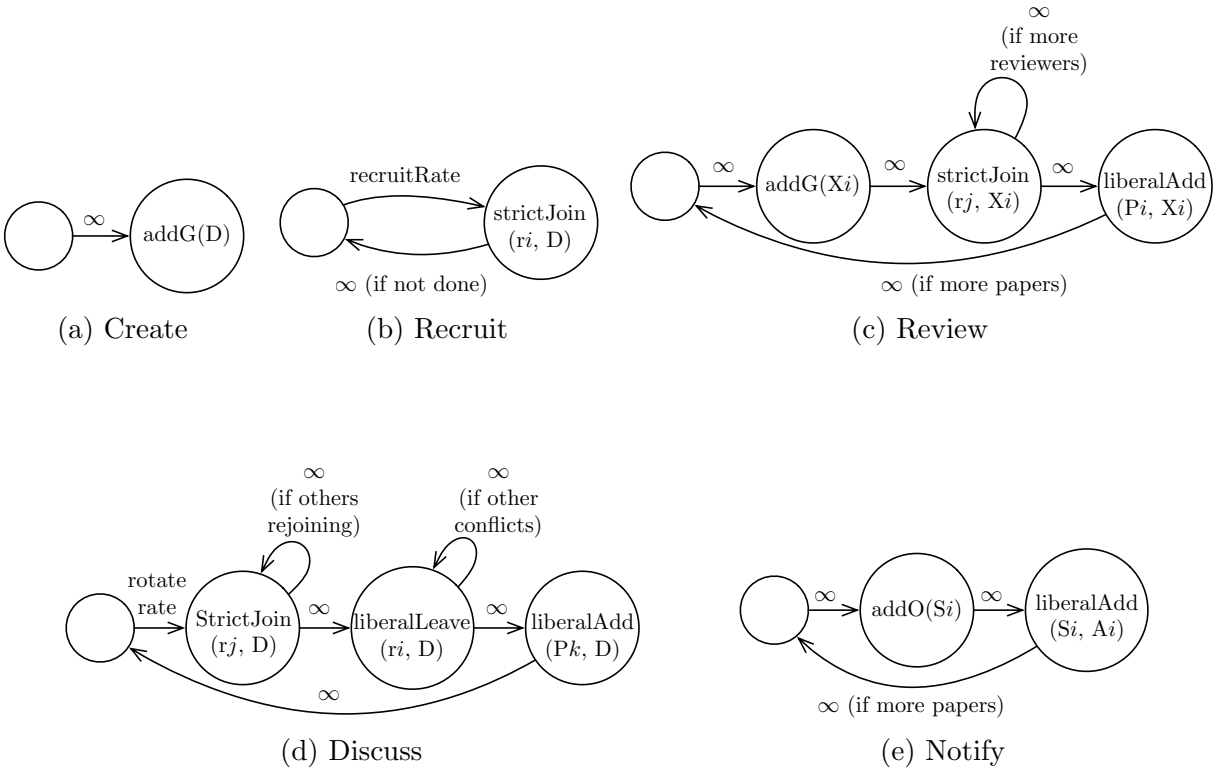
To perform simulation-based cost analysis, we formalized actor machines for the conference program chair, authors, and reviewers, as well as workflows that describe how these actors interact. These instantiations of the structures defined in Section 4.4.1 allow us to describe the usage of the workload system.

The actors in our system are a program chair, a set of reviewers, and a set of authors. Each paper is assigned three reviewers, and each reviewer is assigned nine reviews, and thus we have three times as many authors as reviewers (without loss of generality, we assume that one author is registered to submit each paper). The program chair is responsible for administrative tasks such as creating groups (the discussion group and each paper's review group), assigning reviews to reviewers, copying the submitted papers into their respective



(a) Review (b) Discuss

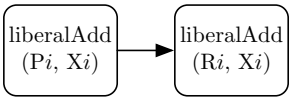
Figure 6: Program Committee invocation: Reviewer actor machines



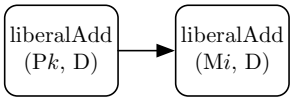
(a) Create (b) Recruit (c) Review

(d) Discuss (e) Notify

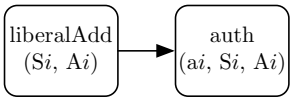
Figure 7: Program Committee invocation: Chair actor machines



(a) Review: Reviews can not be submitted before the corresponding paper is available



(b) Discuss: Discussion can not occur until a paper has been submitted for discussion



(c) Notify: Authors can not read their reviews until they are made available

Figure 8: Program Committee invocation: Workflows

review groups, and rotating through the submitted papers during discussion. The program chair also transitions between the phases of the simulation, which determines the actions that the other actors can execute at any particular time. The phases proceed in the following order:

1. ***Create*** Chair creates the discussion group
2. ***Recruit*** Chair adds the reviewers to the discussion group
3. ***Submit*** Authors create author groups and submit papers
4. ***Review*** Chair creates review groups and adds assigned reviewers; reviewers add reviews to the review groups
5. ***Discuss*** Chair rotates discussion between various papers; reviewers add comments in the discussion group; conflicted reviewers leave during discussion
6. ***Notify*** Chair adds review summaries to author groups; authors read their summaries

Formally, each actor machine includes the actions from all phases, and workflows ensure that only the current phase's actions are enabled. For simplicity, we show them as separate actor machines between which the actor transitions. Figure 5, Fig. 6, and Fig. 7 show the actor machines for each phase for authors, reviewers, and the program chair, respectively. Figure 8 shows the workflows for phases, where they are needed.

To conduct cost analysis, we built a Java implementation of ACCostEvalSim to simulate the conference workload. We repeated the simulation for 200 runs, randomly selecting the

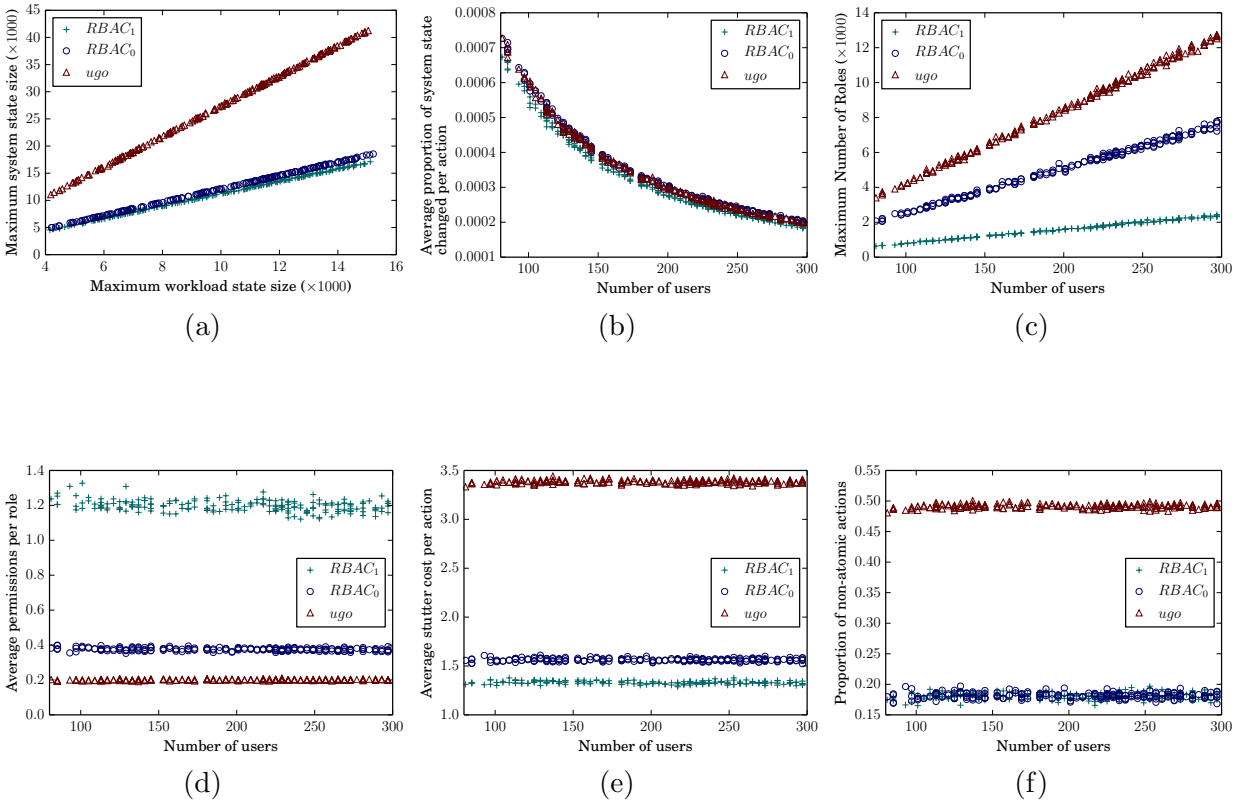


Figure 9: Conference workload cost analysis results using ACCostEvalSim and 200 runs

number of authors and reviewers (preserving the proportion of 3 times as many authors as reviewers).

In Figure 9a, we compare the maximum system state size to the size of the equivalent workload state, demonstrating the storage overhead needed to utilize each system. While the role-based schemes use a small amount of additional state, *ugo* requires several times the storage of the workload. This is due to *ugo*'s restriction that each object is owned by a single group; an object that should be accessed by multiple groups must be owned by a combined group which contains all the members of the originals. Figure 9b shows that a similar *proportion* of each system's state changes on average for each action executed, but given the larger storage required by *ugo*, this system will require much greater I/O as well.

We compare the number of users to the number of roles (groups in *ugo*) created in each system in Figure 9c. RBAC_0 uses many extra roles to simulate hierarchy information, and *ugo* creates even more since each object is owned by only a single group. However, even RBAC_1 uses several times as many roles as there are users in the system, potentially indicating a poor fit from all three systems, as the administrative value of using roles is reduced when the number of roles exceeds the number of users [119]. As indicated in Figure 9d, roles in all three systems are particularly permission-sparse, averaging 1.2, 0.4, and 0.2 permissions per role in RBAC_1, RBAC_0, and *ugo*, respectively. In particular, RBAC_0 and *ugo* utilize many additional roles to store simulated hierarchy information, and many of these roles are never assigned permissions.

In Figure 9e, we investigate the average number of *stutter steps* per action, or the average number of system commands that must be executed to simulate each workload command. RBAC_1, RBAC_0, and *ugo* must execute, on average, 1.3, 1.6, and 3.4 actions (respectively) for each workload action simulated. Furthermore, as shown in Figure 9f, 18% of workload actions incur some stuttering in the role-based systems, and 49% incur stuttering in *ugo*. In scenarios where multiple users will be interacting with the system, this loss of atomicity necessitates the incorporation of an additional locking layer to ensure the system is not accessed in an inconsistent state.

4.5.4 Summary of Findings

The preceding case study shows that, under the lens of several cost measures, RBAC_1 is a better choice than RBAC_0 or *ugo* for implementing the conference workload, due to its native support for role hierarchies, a structure that can mimic the pattern of authorized requests common in the workload. However, even RBAC_1 utilizes a large number of permission-sparse roles, indicating that even it may be a tenuous fit for the workload. More importantly, though, our case study demonstrates that the concepts of our two-phase suitability analysis framework can be applied to a realistic workload and evaluate access control systems that are common in practice with respect to that workload. Our simulation procedure allows us to easily determine the overheads of using each system, and with an average runtime of around

eight minutes per five-month simulation run, does so efficiently. An improved simulator could also execute multiple runs in parallel (in Chapter 5, we discuss an extensible simulation framework for the suitability analysis problem that incorporates this and other features).

4.6 REQUIREMENTS, REDUX

In Section 4.2, we outlined solution 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 by our workload formalism and our simulation procedure: the former allows the analyst to specify a broad range of workloads, while the latter enables cost analysis over many workload instances. *Cooperative Interaction* is met by combining our invocation formalism with the WSP solver (i.e., WSAT) leveraged by ACCOSTEVALSIM: constrained workflows articulate the ways in which cooperation must be carried out, while the use of actor graphs and the WSP solver ensures the generation of compliant traces. Although this chapter makes use of a fixed set of implementation guarantees to define implementation safety, this is not mandatory. Proofs of safety are carried out manually, allowing any notion of safety to be used and providing *Tunable Safety*. Section 4.4.2 demonstrated that our notion of cost measure is capable of representing a wide range of system- and human-centric costs and thus provides *Tunable Costs*. Supporting multi-user workflows is seemingly at odds with the *Tractability* requirement, as the workflow satisfiability problem has been shown to be NP-complete [117]. However, the proof of Theorem 1 makes use of recent results [24, 28] to show that ACCOSTEVALSIM is fixed-parameter tractable in maximum task length (typically a small constant). More concretely, simulating each 5-month period in our case study took, on average, around 8 minutes. In conclusion, the analysis framework developed in this chapter meets each of the desiderata outlined in Section 4.2, and provides a flexible, efficient, and precise mechanism for analyzing instances of the access control suitability analysis problem.

4.7 SUMMARY

In this chapter, we formally defined suitability analysis based on the motivation provided in Chapter 3. We proposed the formal problem statement (which aligns with the analysis goals of this dissertation's thesis statement), then a two-phase framework to solve the problem. We developed a trace generation method based on interleaving the behavior of multiple actors in the system, constrained by workflows that describe how they interact. We also presented an algorithm for conducting cost analysis on traces. Our analysis of the procedure evaluated it formally, and our conference workload case study evaluated it practically.

Moving forward through this dissertation, we will continue to use this framework as the backbone of many of the contributions we make. This will come in the form of presenting more refined or detailed versions of components (e.g., the cost analysis simulation engine in Chapter 5, expressiveness reductions in Chapter 8) as well as using the framework toward additional case studies (e.g., an in-depth case study of the class of group-centric access control models containing the PC workload in Chapter 6, an evaluation of encryption-based workloads for untrusted cloud storage in Chapter 7).

5.0 Portuno: AN ACTOR-BASED SIMULATOR FOR ACCESS CONTROL SUITABILITY ANALYSIS

Having presented the formal problem statement and mathematical framework for suitability analysis, we now focus our attention on the Java-based simulation engine that enables us to conduct the second phase of suitability analysis in practice: Portuno. Through Portuno, we support our thesis by showing that practical cost evaluation of access control is feasible. Further, we discuss the instantiation of the powerful trace generation methods first proposed in Chapter 4, which helps ensure that our evaluation is true to the desired usage of the access control system.

5.1 INTRODUCTION

At its core, Portuno utilizes an implementation of ACCostEvalSim (Algorithm 1). In support of this algorithm, we have developed Java-based representations for the mathematical structures used in the two-phase suitability analysis framework, including access control systems, workloads, implementations, cost measures, and cost functions.

To represent the results of Phase 1 of suitability analysis within Portuno, we present techniques for encoding the state machines representing candidate systems and workload operational components as classes in Java. We also present an API for implementations, to provide Portuno with a recipe for mapping workload usage to candidate system actions.

To enable costs to be evaluated over representative traces of workload usage, we describe how an analyst can encode a constrained, actor-based workload invocation structure in the formalism of Definition 17 within Portuno, representing in an executable way the behavior of

entities that will use the access control system.

To allow Portuno to consider the wide variety of costs incurred by candidate access control systems, we present an extensible, multi-component measurement system. This system is capable of representing the vast range of cost measures described in Section 4.4.2. It also provides tools to assist in encoding cost functions. This measurement system is able to inspect many of the components of the simulation, enabling the analyst to easily represent even complex cost functions.

While ACCostEvalSim conducts cost analysis over a single trace, realistic evaluations should consider many traces. Portuno includes a Monte Carlo driver that repeatedly calls ACCostEvalSim using different traces sampled from the distribution. This driver is useful for exploring trends in costs (for instance, by plotting costs vs. the number of users in the trace). We also discuss a confidence-bounding driver for determining a particular expected cost to within a specific confidence interval.

To demonstrate how Portuno could be used for a realistic suitability analysis, we expand the discussion of the case study from Section 4.5 with details regarding its implementation within Portuno's code framework.

The remainder of this chapter is structured as follows. Section 5.2 presents an overview of how we use simulation to satisfy the remaining components of the Suitability Analysis Problem. We describe the details of our solution in Sections 5.3–5.5. We present further details of the case study from the previous chapter, emphasizing the use of the Portuno simulation engine in Section 5.6 before summarizing in Section 5.7.

5.2 SIMULATING FOR COST ANALYSIS

In this section, we restate the solution requirements for frameworks that solve the suitability analysis problem, describe the key processes for a simulation framework that allows analysts to make the cost decision described in Section 4.4 while satisfying these requirements, and overview the design of Portuno toward carrying out these processes.

5.2.1 Solution Requirements

We now restate the solution requirements for suitability analysis frameworks (Section 4.2).

First, we consider trace generation requirements.

SR1: Domain exploration It must be possible to efficiently explore many initial conditions to examine the effects of various levels of concurrency and resource limitation.

SR2: Cooperative interaction Tasks within large organizations typically require the interaction of many individuals. Suitability analysis frameworks should support operational workflows and constraints on their execution.

Next, we must ensure that the suitability analysis framework can be tuned to meet the specific needs of an application via selection of the metrics used to assess the suitability of an access control system for a given workload.

SR3: Tunable safety There may be many different ways for a system to implement a given workload. It must be possible for an analyst to specify the security guarantees required for implementations of their workload.

SR4: Tunable cost There is no single notion of cost that is sensible for use in every access control analysis [63]. Suitability analysis frameworks must be capable of representing many types of costs and examining multiple costs simultaneously.

Finally, we consider requirements that ensure that the suitability analysis framework remains practical to use (i.e., accurate and efficient), even for large-scale application workloads.

SR5: Tractability The cost analysis simulation procedure should be tractable (e.g., polynomial time or fixed-parameter tractable) to remain feasible to use even for large systems.

SR6: Accuracy Since exploring all possible traces during cost analysis is impractical, it must be possible to approximate the expected error of costs obtained by exploring only a specific subset of these traces.

5.2.2 Key Processes

In cost analysis, we would like to evaluate the costs of traces of expected usage in each candidate system. In order to ensure that the traces considered satisfy the security guarantees as proved by the implementations that were constructed during the expressiveness phase, we generate traces of usage at the workload level, then map them to traces in each candidate system using these implementations. Thus, our first goal of cost analysis is *trace generation*, the process of generating *representative* traces of workload usage (that is, determining a subset of the valid traces from $\mathcal{T}$ that are representative of expected usage and should thus be considered in cost analysis). Given the constrained, actor-based invocation mechanism presented in Definition 17, we will accomplish this task by formalizing actor machines and constrained workflows within Portuno, then executing and interleaving as per the semantics of these structures. This task is detailed in Section 5.3.

The implementability requirements requirements IR1–3 (Section 4.3) enforced on the structure of access control implementation ensure that we are able to map the generated workload traces to corresponding system traces. We must then calculate the costs of each of these mapped system traces. We must therefore formalize within Portuno the cost measures (Definition 18) that describe the quantitative metrics to which the application in question is sensitive. We then label each trace in each candidate system with a cost in each cost measure, which requires us to encode our cost functions (Definition 19) within Portuno as well. Finally, we combine these costs into an overall cost for each candidate access control system. We collectively refer to these processes as measuring the cost of system traces, and address them in Section 5.4.

5.2.3 Simulator Design

We now present an overview of Portuno, the simulation framework that we have developed to conduct the cost analysis phase of suitability analysis while satisfying the requirements recapped in Section 5.2.1. This overview is depicted in Fig. 10.

First, we must represent the structures resulting from Phase 1 of suitability analysis: the *workload*, *candidate systems*, and *implementations* of the former in each of the latter. We will

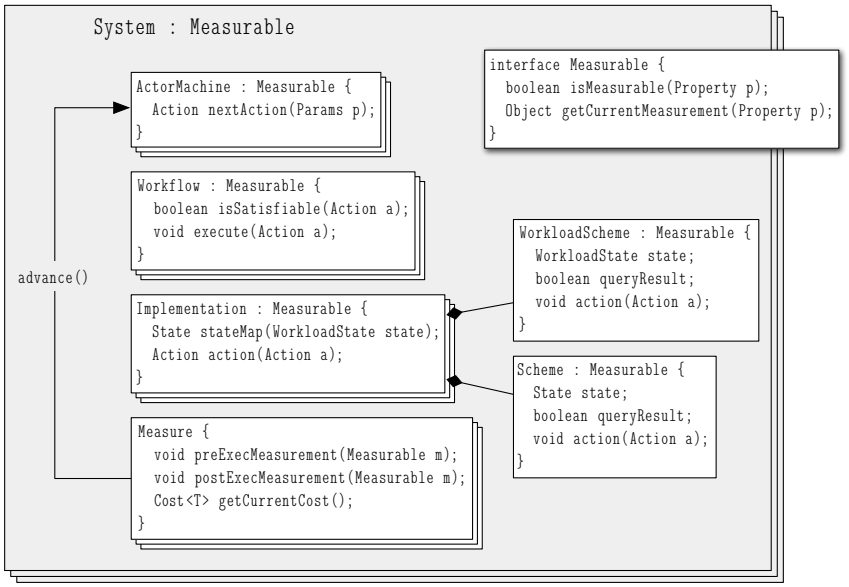


Figure 10: Overview of the architecture of our suitability analysis simulator

address the traces of a workload when discussing trace generation; for now, we only consider the abstract state machine (i.e., operational) component. As workloads are structurally very similar to candidate systems, we represent them in Portuno very similarly as well. A workload system or candidate system consists of sets of states, commands, and queries. A system's states are represented as subclasses of the `State` class (the equivalent for workloads is `WorkloadState`).

Example 3. RBAC_1 states are represented in Portuno using class `Rbac1State : State` with members to store the contents of its state elements: $\langle U, R, P, UR, PA \rangle$. The members are represented as follows.

```
Set<String> U;
Set<String> P;
Set<String> R;
Map<String, Set<String>> UR;
Map<String, Set<String>> PA;
Set<RolePair> RH;
```

Although UR and PA are defined mathematically in Example 1 as $U \times R$ and $R \times P$ (resp.), we implement the corresponding sets, `UR` and `PA`, in Portuno as $U \to \wp(R)$ and $P \to \wp(R)$ (resp.). This design choice is made for efficient indexing of these elements during simulation.

`Rbac1State` also provides low-level methods for altering these state elements; these are primarily for use by the corresponding `Scheme` subclass. ◊

Commands and queries are added in a `Scheme` class (`WorkloadScheme` for workloads). We represent all commands and queries as parameterized objects of the `Action` class, which represents the name and parameters of the command/query. For instance, "Does user u exist?" may be represented as an `Action` with name "userExists" and single parameter u. Each system responds to invocations of the `action(Action acAction)` method by appropriately altering the current state (in the case of a command) or placing the correct TRUE/FALSE value in member variable `Boolean queryResult` (in the case of a query).

Example 4. The RBAC_1 system is represented in Portuno using class `Rbac1 : Scheme`. Its `action(Action acAction)` method contains two cases. If `a` is a query, a private method is called to extract the parameters, inspect the `state` data member, and store the answer to the query in `queryResult`. If `a` is a command, a similar private method is called which determines which `Rbac1State` method should be executed on `state` in order to alter it according to the command.

```
public void action(Action acAction) {
    super.action(acAction);
    if (!acAction.isQuery) {
        command(acAction);
    } else {
        query(acAction);
    }
}
```

The following demonstrates how queries are handled (here we encode only the queries that were needed for our implementation of the PC workload using RBAC_1).

```
private void query(Action acAction) {
```

```
Rbac1State rState = (Rbac1State) state;
String u, r, p;
switch (acAction.name) {
case "UR":
    u = acAction.params[1];
    r = acAction.params[2];
    if (rState.U.contains(u)) {
        if (rState.R.contains(r)) {
            if (rState.UR.containsKey(u)) {
                queryResult = rState.UR.get(u).contains(r);
            } else {
                queryResult = false;
            }
        } else {
            Log.d("Query␣mismatch:␣UR␣with␣non-existent␣role␣" + r);
            queryResult = false;
        }
    } else {
        Log.d("Query␣mismatch:␣UR␣with␣non-existent␣user␣" + u);
        queryResult = false;
    }
    return;
case "auth":
    u = acAction.params[1];
    p = acAction.params[2];
    if (rState.U.contains(u)) {
        if (rState.P.contains(p)) {
            if (rState.UR.containsKey(u) && rState.PA.containsKey(p)) {
                Set<String> uRoles = rState.UR.get(u);
                Set<String> pRoles = rState.PA.get(p);
```

```
                    // Check if one of uRoles is a senior/= to one of pRoles
                    queryResult = SetwiseSeniorEq(uRoles, pRoles);
                } else {
                    queryResult = false;
                }
            } else {
                Log.d("Query␣mismatch:␣auth␣with␣non-existent␣permission␣" + p);
                queryResult = false;
            }
        } else {
            Log.d("Query␣mismatch:␣auth␣with␣non-existent␣user␣" + u);
            queryResult = false;
        }
        return;
    }
}
```

The following demonstrates how commands are handled.

```
private void command(Action acAction) {
    Rbac1State rState = (Rbac1State) state;
    String u, r, p, senior, junior;
    switch (acAction.name) {
        case "addU":
            u = acAction.params[1];
            sys.addActor(u, new ActorUser());
            rState.addUser(u);
            break;
        case "delU":
            u = acAction.params[1];
            sys.removeActor(u);
```

```
        rState.delUser(u);
        break;
    case "addR":
        r = acAction.params[1];
        rState.addRole(r);
        break;
    case "delR":
        r = acAction.params[1];
        rState.delRole(r);
        break;
    case "assignUser":
        u = acAction.params[1];
        r = acAction.params[2];
        rState.assignUser(u, r);
        break;
    case "revokeUser":
        u = acAction.params[1];
        r = acAction.params[2];
        rState.revokeUser(u, r);
        break;
    case "assignPermission":
        p = acAction.params[1];
        r = acAction.params[2];
        rState.assignPermission(p, r);
        break;
    case "revokePermission":
        p = acAction.params[1];
        r = acAction.params[2];
        rState.revokePermission(p, r);
        break;
```

```
            case "addHierarchy":
                senior = acAction.params[1];
                junior = acAction.params[2];
                rState.addHierarchy(acAction.params[1], acAction.params[2]);
                break;
            case "removeHierarchy":
                senior = acAction.params[1];
                junior = acAction.params[2];
                rState.removeHierarchy(acAction.params[1], acAction.params[2]);
                break;
            default:
                throw new RuntimeException("Unrecognized␣command:␣" + acAction);
        }
    }
```

In particular, we note the use of the `addActor()` method in `"addU"` (and corresponding `removeActor()` in `"delU"`). This mechanism allows the actors in the system to be adjusted dynamically during the simulation (most commonly, as seen here, upon adding and removing users). ◊

Finally, an `Implementation` object represents the mapping from a workload to an implementing system. The `stateMap(WorkloadState s)` method returns a state in the implementing system that corresponds to a given workload state (i.e., it encodes σ_Γ). The `action(Action acAction)` method executes a workload action in the current implementing state. Like the `action()` method from the `Scheme` and `WorkloadScheme` classes, it handles both commands and queries (in this case, it encodes both σ_Ψ and σ_Q). If the workload action is a command, the corresponding system actions are executed and the current state updated. If the workload action is a query, the implementing state is inspected and the result of the query is stored in `queryResult`.

Example 5. The implementation of the PC workload in RBAC_1 (as described in the proof of Theorem 2) is represented in Portuno using class `ImplementPcInRbac1 : Implementation`. The `stateMap()` method converts PC states to RBAC_1 states.

```
public State stateMap(WorkloadState ws) {
    Rbac1State rState = new Rbac1State();
    PcState pcState = (PcState) ws;

    List<String> S = pcState.getS().getList();
    for (String s : S) {
        rState.addUser(s);
    }

    List<String> G = pcState.getG().getList();
    for (String g: G) {
        rState.addRole(g);
        String topRole = newRoleName();
        rState.addRole(topRole);
        String bottomRole = newRoleName();
        rState.addRole(bottomRole);
        rState.addHierarchy(topRole, g);
        rState.addHierarchy(g, bottomRole);
    }

    List<String> O = pcState.getO().getList();
    for (String o: O) {
        rState.addPermission(o);
    }

    HashMap<Integer, Object> records = pcState.getSortedRecords();

    for (Object record : records) {
        if (record instanceof LiberalJoinRecord) {
            LiberalJoinRecord lj = (LiberalJoinRecord) record;
```

```
        rState.assignUser(lj.subject, lj.group);
    } else if (record instanceof LiberalLeaveRecord) {
        LiberalLeaveRecord ll = (LiberalLeaveRecord) record;
        String orphan = newRoleName();
        rState.addRole(orphan);
        Set<String> permissions = permsInChain(ll.subject, ll.group, rState);
        String top = findTop(ll.group, rState);
        for (String permission : permissions) {
            rState.assignPermission(permission, orphan);
        }
        rState.assignUser(ll.subject, orphan);
        rState.addHierarchy(top, orphan);
        leaveDown(ll.subject, ll.group, rState);
    } else if (record instanceof StrictJoinRecord) {
        StrictJoinRecord sj = (StrictJoinRecord) record;
        String newBottom = newRoleName();
        String oldBottom = findBottom(sj.group, rState);
        rState.addRole(newBottom);
        rState.addHierarchy(oldBottom, newBottom);
        rState.assignUser(sj.subject, newBottom);
    } else if (record instanceof StrictLeaveRecord) {
        StrictLeaveRecord sl = (StrictLeaveRecord) record;
        leaveDown(sl.subject, sl.group, rState);
        leaveOrphans(sl.subject, sl.group, rState);
    } else if (record instanceof LiberalAddRecord) {
        LiberalAddRecord la = (LiberalAddRecord) record;
        String bottom = findBottom(la.group, rState);
        rState.assignPermission(la.object, bottom);
    } else {
        Log.e("Invalid␣record␣type");
```

```
            throw new InvalidArgumentException();
        }
    }
    return rState;
}
```

The action(Action acAction) method converts PC actions to sequences of RBAC_1 actions and executes the generated sequences.

```
void action(Action acAction) {
    super.action(acAction);
    Rbac1 rbac = (Rbac1)scheme;
    Rbac1State rState = scheme.state;
    String actor = acAction.params[0];
    String s, o, g;
    switch (acAction.name) {
        case "delG":
            g = acAction.params[1];
            Set<String> chain = rState.findRoleChain(g);
            for (String r : chain) {
                rbac.action(new Action("delR", actor, r));
            }
            break;
        case "addO":
            o = acAction.params[1];
            rbac.action(new Action("addP", actor, o));
            break;
        case "delO":
            o = acAction.params[1];
            rbac.action(new Action("delP", actor, o));
            break;
```

```
case "LAdd":
    o = acAction.params[1];
    g = acAction.params[2];
    String bottom = findBottom(g, rState);
    rbac.action(new Action("assignPermission", actor, bottom, o));
    break;
case "SJoin":
    s = acAction.params[1];
    g = acAction.params[2];
    String newBottom = newRoleName();
    String oldBottom = findBottom(g, rState);
    rbac.action(new Action("addR", actor, newBottom));
    rbac.action(new Action("addHierarchy", actor, oldBottom, newBottom));
    rbac.action(new Action("assignUser", actor, s, newBottom));
    break;
case "SLeave":
    s = acAction.params[1];
    g = acAction.params[2];
    leaveDownCmd(actor, s, g);
    leaveOrphansCmd(actor, s, g);
    break;
case "addG":
    g = acAction.params[1];
    rbac.action(new Action("addR", actor, g));
    String topRole = newRoleName();
    rbac.action(new Action("addR", actor, topRole));
    String bottomRole = newRoleName();
    rbac.action(new Action("addR", actor, bottomRole));
    rbac.action(new Action("addHierarchy", actor, topRole, g));
    rbac.action(new Action("addHierarchy", actor, g, bottomRole));
```

```
    break;
case "LLeave":
    s = acAction.params[1];
    g = acAction.params[2];
    String orphan = newRoleName();
    rbac.action(new Action("addR", actor, orphan));
    HashSet<String> permissions = permsInChain(s, g, rState);
    String top = findTop(g, rState);
    for (String permission : permissions) {
        rbac.action(new Action("assignPermission", actor, orphan, permission));
    }
    rbac.action(new Action("assignUser", actor, s, orphan));
    rbac.action(new Action("addHierarchy", actor, top, orphan));
    leaveDownCmd(actor, s, g);
    break;
case "LJoin":
    s = acAction.params[1];
    g = acAction.params[2];
    rbac.action(new Action("assignUser", starter, s, g));
    break;
case "auth":
    s = acAction.params[1];
    o = acAction.params[2];
    g = acAction.params[3];
    rbac.queryResult = false;
    if (rState.U.contains(s) && rState.P.contains(o) &&
            rState.UR.containsKey(s) && rState.PA.containsKey(o)) {
        Set<String> sRoles = rState.UR.get(s);
        Set<String> oRoles = rState.PA.get(o);
        Set<String> gRoles = getSeniors(g, rState);
```

```
                if (!gRoles.isEmpty()) {
                    // Will contain at most one role
                    String gRole = gRoles.get(0);

                    // Eliminate all oRoles except those that gRole is senior to
                    oRoles.retainAll(getJuniors(gRole, rState));

                    // Check if one of sRoles is a senior/= to one of oRoles
                    rbac.queryResult = SetwiseSeniorEq(uRoles, pRoles);
                }
            }
            break;
        default:
            Log.w("Cannot execute action " + acAction);
    }
}
```

We note in particular the process of mapping the authorization query. In answering whether s has access to o via group g as described by the implementation, we manipulate several sets of roles. First, we determine all roles of which s is a member, and all roles that have access to o. Then, we determine the role that is senior to g (if role names are generated properly, then g must be a valid group and thus there must be at most one role senior to g). We eliminate all of o's roles that are not junior to g's senior. Finally, we execute the last check: whether one of s's roles is senior or equal to one of o's remaining roles. This yields the same answer as σ_Q in the proof of Theorem 2, by determining whether s has access to o via a role in the same chain as g. ◊

Having discussed and demonstrated how each of the components from Phase 1 are encoded in the simulation framework, we will now address the key processes from Section 5.2.2. The first is trace generation, which begins by generating an initial state generated from an analyst-supplied distribution. The commands and queries that follow this start state are

generated from the behavior of independent *actors* in the system combined with restrictions on their behavior in the form of *constrained workflows*. Each actor's behavior is represented by an `ActorMachine` object that encodes the relevant instance of Definition 15. Each partially-completed constrained workflow is represented by a `Workflow` object that encodes an instance of Definition 16: that is, it describes which actions depend upon which others and which actions must be completed by which classes of users. At each point in time, each actor is queried for an action (via the `ActorMachine`'s `nextAction()` method). If an action is returned, it is checked against the restrictions formalized in the `Workflow` objects to ensure it can be allowed: if so, it is added to the trace being generated. We discuss these trace-generation components in full detail in Section 5.3.

The second key process is calculating the costs of a trace. In Portuno, we calculate the cost of a trace while simulating its execution. Each cost measure that is part of the simulation is represented as a `Measure` which aggregates objects of a `Cost` class. A `Cost` defines a data type `T` (e.g., `Integer`) and an `aggregate` method (e.g., summation). For each action that is encountered during the execution of a trace, the current value of each measure in the system is calculated and aggregated into the running total for that measure.

In order to support a variety of potential costs, we need to enable the measurement of a wide range of properties of many components in the framework. For instance, measuring storage overhead requires inspecting both the workload state and the implementing system's state; measuring personnel-hours requires inspecting the actor machines; and measuring the proportion of initiated tasks that go uncompleted requires inspecting the workflows. Thus, Portuno provides the `Measurable` interface that is implemented by all of these components and enables each to return measurements (via `getCurrentMeasurement`). To identify which property a `Measurable` should return, a measurement request specifies a `Property` object, which describes the object to do the measuring and the measurement that should be taken (e.g., "the number of users in the workload state").

As we describe Portuno in depth, we will also elaborate on several other details. We typically execute Portuno in a Monte Carlo fashion, generating large numbers of traces and measuring costs across a distribution of start states and parameters that control actor behavior (looking at means for expected costs, and comparing to parameter choices in the

initial states to analyze trends). We can also utilize the framework by picking specific start states, repeating the simulation until we reach a particular confidence about the resulting costs. Finally, as the multiple simulation runs are trivially parallelizable, we have made Portuno a client/server framework, allowing a client running `SimulationManager` to delegate executions to servers running `SimulationWorker`.

5.3 TRACE GENERATION

The first phase of cost analysis is generating traces of workload usage which accurately describe how it is likely to be used in practice. In this section, we describe our solution to trace generation using constrained workflows and actor machines.

The invocation mechanism of an access control workload (Definition 12) describes valid usage of the access control system within the application being described. This is represented as a set of traces through the system's actions (commands and queries). In cost analysis, we need to sample from this set of traces in a way that is representative of the *expected* usage while satisfying the requirements described in Section 5.2.1.

Actions are the basic units of work executed by an actor in the system, a *parameterized* generalization of queries and commands. In Portuno, actions are represented as instances of the `Action` class. Such objects are the primary method of interfacing with both the workload and implementing systems, and can specify some or all of the action's required parameters. An `Action` object thus specifies the action name, whether it is a command action or a query action, and the known parameters (allowing others to be not-yet-specified).

To describe the behavior of actors, we employ state machines that we call *actor machines*. The primary goal of an actor machine is, at each instant in time, to produce a candidate action for the entity that it represents to execute (or to pass on executing an action in this instant). Actor machines are represented as objects of the `ActorMachine` abstract class. Each type of actor is thus represented as a subclass of `ActorMachine`. For instance, an experiment in Portuno may contain a subclass of `ActorMachine` for administrative users and another for regular users. Each actual actor is then represented as an object instantiating the corresponding

ActorMachine subclass.

As the main goal of the actor machine is to return a candidate action for each instant in time, the primary method of any ActorMachine subclass is nextAction(Params p), which returns an Action object based on the current state of the actor machine and the amount of simulated time elapsed since the last execution of nextAction(Params p).

Example 6. The example user actor machine depicted in Figure 3a can be represented in Portuno as follows.

```
Action nextAction(Params p) {
    double createRate = 2.0 / Conversion.Day;
    double declassRate = 0.1 / Conversion.Month;
    double coin = Simulation.rand.nextDouble() / p.delta;

    if (coin <= createRate) {
        return new Action("addO", actor, null);
    }
    coin -= createRate;

    if (coin <= declassRate) {
        return new Action("reqDeclass", actor, null);
    }
    coin -= declassRate;

    return null;
}
```

We use the included Conversion library to specify rates independent of the current simulation timestep (p.delta), though we note that this timestep must still be specified when generating the random "coin." In this example, the first parameter of the resulting action (executing actor) is specified, while the second parameter (the object in question) is left as null, allowing the simulation to automatically select the appropriate object (in this case, an available object

name for creation, or an existing object name for requesting declassification). ◊

Since each executing actor machine represents an entity acting upon the access control system, Portuno must support the adding and removing of actor machines as users are added and removed. Thus, after each action executed, Portuno will re-enumerate the set of actors, spinning up new `ActorMachines` and shutting down others as necessary (see the use of `addActor()` and `removeActor()` in Example 4).

To describe dependencies between actions taken by one or more actors, we make use of the notion of a *constrained access control workflow*, which organizes the execution of sequences of actions. Intuitively, the goal of this structure is to approve or deny a candidate action returned from an `ActorMachine` on the basis of actions taken by other actors in the system. We describe a disjoint subset of a workflow as a *task*.

In Portuno, a constrained workflow is represented as a subclass of the `Workflow` abstract class, and a constrained workflow instance (the workflow along with the actions already executed) is represented as an object of some `Workflow` subclass. The primary method of interest is `isSatisfiable`, which takes an `Action` and determines whether it should be allowed to execute in the current workflow instance, based on the actions already executed. When an `ActorMachine` returns an action, it is checked against each executing `Workflow`: if no existing `Workflow` allows the `Action`, a new, blank `Workflow` is checked. If the `Action` completes a task in the `Workflow`, the `Workflow` is removed from the currently-executing set.

Before we can verify that a candidate action does not violate the constrained workflow, Portuno fills in parameters that the actor did not specify using the `WorkloadScheme`'s `Action addParameters(Action a)` method. For instance, if an actor specifies the action `assignUser(Alice, null)`, which assigns Alice to an unspecified role, the simulation may fill in the missing role name with a random non-privileged role to which Alice is not already assigned.

Once parameters are established, to check an `Action` against a `Workflow`, the action name is found within the constrained workflow system and an instance of the workflow satisfiability problem (WSP) [28] is solved to determine whether the task containing this action will remain satisfiable if this action is allowed to be executed. The simplest failure is one where a prerequisite action has not yet been executed or an already-executed action belongs to a disjoint task; a more subtle failure can occur if allowing this user to execute the current

action will result in no other existing users being capable of executing a later action in the task. Efficient (FPT) algorithms for certain restrictions of WSP have been proposed, and have been shown to cover the vast majority of constraints required in practice [24]. For simple workflows, even more efficient algorithms can be used, e.g., as follows.

Example 7. The example workflow depicted in Figure 3b can be represented in Portuno as follows.

```
boolean isSatisfiable(Action a) {
    switch (a.name) {
        case "addO":
        case "reqDeclass":
            return executed.isEmpty();
        case "approve":
            if (!executed.contains("reqDeclass"))
                return false;
            if (executed.contains("approve"))
                return false;
            return sys.impl.scheme.numAdmins() >= 2;
        case "coApprove":
            if (!executed.contains("approve")) {
                return false;
            }
            return a.params[0] != executed[1].params[0];
    }
}
```

For each action with dependent actions, these dependents are first ensured to exist in executed, the record of executed actions in the current workflow instance. Then, additonal requirements are checked as appropriate, e.g., approve will not be executed unless there exists another administrator to execute coApprove, which in turn will not be allowed unless requested by a different actor than the one who executed approve. ◊

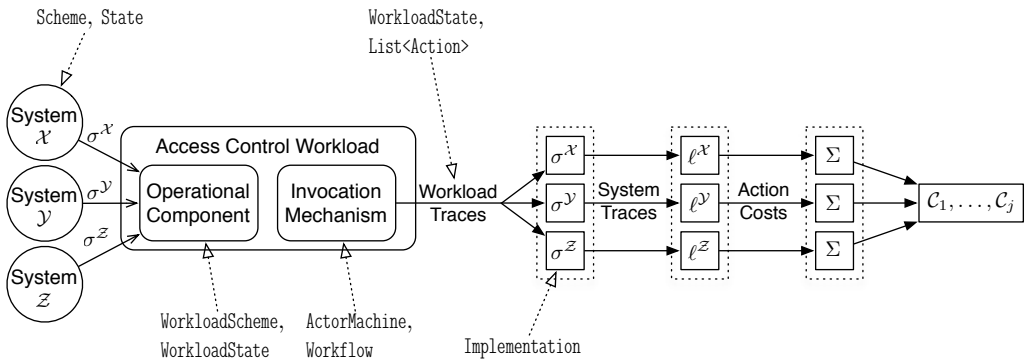


Figure 11: A summary of the Portuno components discussed thus far

5.3.1 Summary

We now summarize the aspects of suitability analysis that we have addressed thus far with the aid of an annotated overview diagram (Fig. 11). We describe in detail the role of expressiveness analysis in Section 4.3. The Portuno structures that represent this phase's structures are discussed in Section 5.2.3: the workload operational component is represented by a `WorkloadScheme`, `WorkloadState` pair; each system is represented by a `Scheme`, `State` pair; and each implementation is represented by an `Implementation`.

Cost analysis requires an expanded workload invocation mechanism in order to sample from the provided valid traces. This sampling should result in a series of traces representing the *expected* usage of the access control system, and as such we introduced the constrained, actor-based invocation mechanism presented in Definition 17. An instance of this definition is represented in Portuno using the `ActorMachine` and `Workflow` classes: one `ActorMachine` subclass per type of actor, and one `Workflow` subclass to manage collaborations between actors.

These structures allow Portuno to generate workload traces. Of course, Portuno needs to execute these traces within each candidate system in order to measure the revelant costs, and thus these workload traces need to be translated into system traces. This will be accomplished using the Portuno components constructed during expressiveness analysis, which yield a procedure for translating traces. Using the `Implementation` corresponding to each `Scheme`, the initial workload state can be converted to a corresponding system state (via `stateMap`), and

each workload action can be converted into an equivalent system action (via `action`).

Having generated representative workload traces and translated such traces into each candidate system, we can then conduct cost analysis by simulating the execution of such system traces and measuring the relevant costs. The next section describes the details of how we conduct this step.

5.4 CALCULATING COST OF TRACES

Once representative workload traces are generated and mapped into candidate system traces, Portuno needs to calculate the costs of these traces.

An important part of cost analysis is choosing relevant cost measures. We do not commit to a particular cost measure, but rather develop our framework to operate on any measure satisfying a number of simple properties (Definition 18). In Portuno, measures are represented as subclasses of the `Measure` class, and each `Measure` aggregates objects of the `Cost` class. A `Cost` defines a data type `T` and an `aggregate` method. Each measure loaded into the simulator will be referenced before and after each action is executed, in order to calculate the cost of the action in question and aggregate it into the running total for the measure. To accomplish this, each `Measure` class contains methods `preExecMeasurement` and `postExecMeasurement`, which are called by the simulator to aggregate the costs of the current action. Some cost measures are more easily calculated before the action's execution (e.g., the number of objects that will be deleted) while others are easier to calculate after (e.g., the number of workflows that have been completed). Still other measures benefit from using both (e.g., the size difference between the initial and final states).

Example 8. We demonstrate the `Measure` class with an example that measures the maximum number of system commands needed to simulate a single workload command.

```
public class MaxStutter extends Measure {
    public MaxStutter() {
        curMeasurement = new IntMaxCost();
```

```
    }

    public String getMeasureName() {
        return "Max_Stutter";
    }

    public String getPrintFriendlyName() {
        return "Maximum␣number␣of␣system␣commands␣to␣simulate␣a␣single␣workload␣command";
    }

    void postExecMeasurement(Measurable m) {
        Property p = new Property(Scheme.class, Scheme.Q_ACTIONSEXECUTED, new String[0]);
        List<Action> actions = (List<Action>)m.getCurMeasure(p);
        curMeasurement.aggregate(actions.size());
    }

    public boolean isMeasurementValid(Measurable w) {
        Property p = new Property(Scheme.class, Scheme.Q_ACTIONSEXECUTED, new String[0]);
        return w.isMeasurable(p);
    }
}
```

This `Measure` uses the `Cost` set of integers with accrual operator max. It measures the `Scheme` property that returns the full set of system actions executed for the last workload action (we discuss returning measurements via the `Property` API in Section 5.4.1). Finally, it aggregates the number of actions that were returned to its current measurement (i.e., takes the max). ◊

5.4.1 Cost Functions

In order to accomplish the aggregation step, the framework needs a way of calculating the cost of executing a particular action within an implementing system. A cost function must

map each (action, parameterization, state) tuple to an element of the relevant cost measure.

Rather than attempt to define and encode all measures and cost functions of interest to analysts, we designed Portuno to allow measures to be loaded at execution-time, and included the `Measurable` interface to facilitate measuring any desired properties about the simulation components that can influence costs; any component that can measure a cost-relevant property must implement this interface. The interface gives measures a defined API for measuring properties about the components of the simulation. Each call to this measurement API must include a `Property` object: the `Property` class defines a naming scheme for measurable objects that ensures that the analyst can precisely identify which components should be polled and which statistics should be measured. The `isMeasurable()` method of the `Measurable` interface allows each measurable entity to specify which measurements it is capable of taking, and the `getCurrentMeasurement()` method allows these measurements to be taken. To take a measurement, Portuno first polls each `Measurable` for whether it is capable of taking the measurement corresponding to the requested `Property`, and then asks the *most general* capable `Measurable` to do so. This allows the analyst to define measurements that are very generic or very specific; "the parameters of the last action executed" is generic and can target any `Scheme`, whereas "the number of roles in the current state" is specific to certain types of schemes and should target, e.g., `RbacState`.

The top level `Measurable`, as seen in Fig. 10, is `System`, the component of Portuno that tracks a single workload-candidate system pair during simulation. This `Measurable` is responsible for passing each request for a `Property` down the hierarchy until it reaches a component that can answer it. A `System` is capable of measuring a top-level `Property`, such as the full set of actors currently in the system, or the action currently being executed. For any other `Property`, it passes it in turn to each of its `Measurable` components. This includes, e.g., each `ActorMachine` and `Workflow`, but the most common is the `Implementation`. This component, in turn, will pass the `Property` to the workload scheme and/or candidate scheme, which will pass it to their corresponding states if needed. Thus, the `Measure` requesting a `Property` should be as specific as needed, otherwise a more general component that can measure the property will take precedence.

Example 9. Here we demonstrate the measurements that `Rbac1State` exposes via the `Property`

API.

```
Object getCurMeasure(Property p) {
    if (super.isMeasurable(p)) {
        return super.getCurMeasure(p);
    }
    Property qTest = new Property(this, "__test", new String[0]);
    if (!qTest.matchQualifier(p)) {
        throw new InvalidMeasureException(p);
    }
    switch (p.name) {
        case "U":
            return U;
        case "P":
            return P;
        case "R":
            return R;
        case "UR":
            return UR;
        case "PA":
            return PA;
        case "RH":
            return RH;
        case "sizeof":
            switch (p.params.get(0)) {
                case "U":
                    return U.size();
                case "P":
                    return P.size();
                case "R":
```

```
                return R.size();
            case "UR":
                return getUrSize();
            case "PA":
                return getPaSize();
            case "RH":
                return 2 * RH.size();
            case "ALL":
                return U.size() + P.size() + R.size() + getUrSize() + getPaSize() +
                        2 * RH.size();
        }
    }
    throw new InvalidMeasureException(p);
}
```

In addition to all the properties exposed by its parent class (`State`), `Rbac1State` exposes its full state elements to measures, as well as methods for determining their size (individually and in total). ◊

Although Portuno allows specific systems, workloads, etc., to expose arbitrary information via measurements, we also provide several commonly-useful measurements in the base classes. By default, measures can request from an implementation the action currently being executed, the actor that requested that action, the set of all actors or workflows, or the current elapsed simulation time. The candidate system and workload system templates include properties that return the current state of the system, the previous state, and the result of the last query. Candidate systems can also return the full list of candidate actions executed to simulate the current workload action (e.g., see Example 8).

While a combination of these generic properties allows a measure to inspect any information about the current state of the simulation, there are situations where it makes sense to define specific properties for specific systems, workflows, or actors for optimization and readability. Consider the analyst who wants to measure the total size of the current

Algorithm 2 ACCostEvalMC: A Monte Carlo driver for ACCostEvalSim

Input: $\mathfrak{Y}$, set of candidate systems
Input: Σ, set of implementations $(\forall \mathcal{S} \in \mathfrak{Y} : \sigma_{\mathcal{S}} \in \Sigma)$
Input: $\mathfrak{C}$, set of cost measures $(\mathcal{C} = \langle \mathbb{R}, +, \leq \rangle \in \mathfrak{C})$
Input: L, set of cost functions $(\forall \mathcal{S} \in \mathfrak{Y}, \mathcal{C} \in \mathfrak{C} : \ell_{\mathcal{C}}^{\mathcal{S}} \in L)$
Input: $I = \langle W, \mathfrak{A}, M \rangle$, 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($\mathfrak{Y}, \Sigma, \mathfrak{C}, L, I, \Pr(\gamma), \chi, T_f, t$)
  for all $[1, \chi]$ **do** ▷ Monte Carlo loop
    $\gamma_0 \leftarrow$ random sample from $\Pr(\gamma)$
    ACCostEvalSim($\mathfrak{Y}, \Sigma, \mathfrak{C}, L, I, \gamma_0, T_f, t$)

candidate system's state. In this case, it probably makes more sense to follow Example 9 in implementing a `sizeof` property that can be measured directly by the state, rather than requesting the state itself and measuring its size within the `Measure` code.

5.5 DRIVERS FOR ACCOSTEVALMC

Once the analyst has defined the trace generation structures, a set of cost measures, and cost functions for each candidate system, she can conduct cost analysis via simulation. Our main simulation procedure, ACCostEvalSim (shown in Algorithm 1), conducts a single, randomized run of the system.

Portuno also includes two drivers for using this simulation procedure. The first, ACCostEvalMC, utilizes the Monte Carlo technique; it calls ACCostEvalSim repeatedly, each time randomly sampling a start state from the given distribution. This allows the analyst to generate a large number of data points across a predefined pattern of start states, which makes it particularly effective in detecting trends across various start states. For example, in the case study presented in Sections 4.5 and 5.6, we randomly choose a number of users in the system for each run, allowing us to see the effect this parameter has on the costs of using each system.

Because the repeated execution in ACCostEvalMC contributes to the complexity of the full

Algorithm 3 ACCostEvalCI: A confidence-bounding driver for ACCostEvalSim

Input: $\mathfrak{Y}$, set of candidate systems
Input: Σ, set of implementations $(\forall \mathcal{S} \in \mathfrak{Y} : \sigma_{\mathcal{S}} \in \Sigma)$
Input: $\mathfrak{C}$, set of cost measures $(\mathcal{C} = \langle \mathbb{R} \times \text{time}, +, \leq \rangle \in \mathfrak{C})$
Input: L, set of cost functions $(\forall \mathcal{S} \in \mathfrak{Y}, \mathcal{C} \in \mathfrak{C} : \ell_{\mathcal{C}}^{\mathcal{S}} \in L)$
Input: $I = \langle W, \mathfrak{A}, M \rangle$, 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 ACCostEvalCI($\mathfrak{Y}, \Sigma, \mathfrak{C}, L, I, \gamma_0, T_f, t, u, v$)
  $n \leftarrow \emptyset$
  while $t_{|n|-1,1-u/2}\sqrt{\frac{S^2(n)}{|n|}} > v \cdot \bar{X}(n)$ **do**
    $n \leftarrow n \cup$ ACCostEvalSim($\mathfrak{Y}, \Sigma, \mathfrak{C}, L, I, \gamma_0, T_f, t$)

analysis by only an additional pseudo-polynomial factor, ACCostEvalMC, like ACCostEvalSim, is in FPT.

Corollary 5. *Under the same assumptions as in Theorem 1, the simulation procedure* ACCostEvalMC *is in* FPT.

Proof. The driver ACCostEvalMC calls ACCostEvalSim χ times. Thus, the runtime complexity of ACCostEvalMC is a factor of χ greater than that of ACCostEvalSim. Since χ is an input, this contributes an additional pseudo-polynomial factor over the runtime complexity of ACCostEvalSim, and thus ACCostEvalMC is in FPT. □

In the interest of satisfying the *Accuracy* requirement, we also present a second driver, which allows the analyst to achieve an intended confidence in the cost value generated for a particular start state. Using this approach, we can decide the number of simulation runs to conduct based on a desired confidence and the assumption of a particular distribution of costs across runs, terminating when a satisfactory confidence is reached. For example, assuming a normal distribution of costs across runs, we can use the fixed-sample-size procedure for point estimate of a mean [74], 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 system. 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 ACCostEvalCI, which uses this approach to execute ACCostEvalSim until a desired confidence is reached, rather than executing for a fixed number of runs.

In Section 4.2 and again in Section 5.2.1, we stated solution requirements to guide the development of our suitability analysis framework: *Domain Exploration*, *Cooperative Interaction*, *Tunable Safety*, *Tunable Costs*, *Tractability*, and *Accuracy*. We now briefly revisit these to discuss the degree to which each was met by Portuno.

The *Domain Exploration* requirement is addressed by the Monte Carlo driver for Portuno in Algorithm 2, which enables cost analysis over many workload instances.

Cooperative Interaction is met by combining the constrained, actor-based invocation formalism of Definition 17 with the WSP solver leveraged by ACCostEvalSim: constrained workflows articulate the ways in which cooperation must be carried out, while the use of actor machines and the WSP solver ensures that all traces generated during cost analysis are compliant with these workflows.

Our framework does not mandate the use of any particular set of safety properties for a notion of safe implementation. Instead, in Section 4.3, we describe the bare minimum requirements that a notion of implementation must satisfy. Proofs of safety are then carried out manually. This allows the use of any notion of safety that meets these minimum requirements (which have been shown to encompass many notions of expressiveness simultaion from the literature [43]), satisfying *Tunable Safety*.

Toward addressing *Tunable Costs*, Section 5.4 describes Portuno's flexible cost measurement system that is capable of aggregating a wide range of system- and human-centric costs.

Supporting multi-user workflows is seemingly at odds with the *Tractability* requirement, as the workflow satisfiability problem has been shown to be NP-complete [117]. However, the proof of Theorem 1 makes use of recent results [24, 28] to show that ACCostEvalSim is fixed-parameter tractable in maximum task length (typically a small constant).

Finally, the *Accuracy* requirement is addressed by ACCostEvalCI, which determines when to halt based on confidence intervals for point estimates of cost.

In conclusion, Portuno meets each of the desiderata outlined in Section 5.2.1, and provides a flexible, efficient, and precise mechanism for analyzing instances of the access control suitability analysis problem.

5.6 CASE STUDY

In this section, we discuss the case study presented in Section 4.5 to demonstrate how it is represented in Portuno.

5.6.1 Workloads Operational Component

The workload's operational component is a group-based program committee workload. The workload states are represented by `PcWorkloadState : WorkloadState` with the following data members.

```
Set<String> S;
Set<String> O;
Set<String> G;
int timestamp;
Map<SGPair, Set<Integer>> liberalJoin;
Map<SGPair, Set<Integer>> liberalLeaves;
Map<OGPair, Set<Integer>> liberalAdd;
Map<SGPair, Set<Integer>> strictJoin;
Map<SGPair, Set<Integer>> strictLeaves;
```

Here, S stores the set of subjects, O the set of objects, and G the set of groups. The set of join/leave/add records is kept in liberalJoin, etc., each stored as a map. These maps are represented with the subject/group or object/group pair as the key and the set of timestamps upon which the relevant operation occured as the value. As with Example 3, this organization is chosen for simulation efficiency; it does not alter the mathematical properties of the state presented in [48].

The workload system, then, is represented by PcWorkload : WorkloadScheme. Like the RBAC_0 example in Example 4, PcWorkload's action(Action acAction) method utilizes methods of its corresponding state class, PcWorkloadState, to carry out the effects of its commands.

```
public void command(Action action) {
    PcWorkloadState pState = (PcWorkloadState) state;
    String g, o, s;
    switch (a.name) {
        case "addG":
            g = action.params[1];
            pState.addGroup(g);
            break;
        case "delG":
            g = action.params[1];
            pState.delGroup(g);
            break;
        case "addO":
            o = action.params[1];
            pState.addObject(o);
            break;
        case "SJoin":
            s = action.params[1];
            g = action.params[2];
            pState.addStrictJoin(s, g);
```

```
            break;
        case "LJoin":
            s = action.params[1];
            g = action.params[2];
            pState.addLiberalJoin(s, g);
            break;
        case "SLeave":
            s = action.params[1];
            g = action.params[2];
            pState.addStrictLeave(s, g);
            break;
        case "LLeave":
            s = action.params[1];
            g = action.params[2];
            pState.addLiberalLeave(s, g);
            break;
        case "LAdd":
            o = action.params[1];
            g = action.params[2];
            pState.addLiberalAdd(o, g);
            break;
    }
}
```

The relevant queries in this workload are the authorization requests, of the form `auth(subject, object, group)` and asking whether `subject` has access to `object` through `group`, as described by the semantics of the strict/liberal join/leave and add operations. However, as the answers to these queries are not relevant to any of the measures of interest in our experiment, we implement these queries in Portuno as simply accessing the required data to accrue the expected I/O costs. This gives us the correct cost information while improving efficiency (e.g.,

not requiring us to develop a functional prototype of the workload's enforcement mechanism).

```
case "auth":
    subject = action.params[0];
    object = action.params[1];
    group = action.params[2];
    SGPair sg = new SGPair(subject, group);
    OGPair og = new OGPair(object, group);

    // Access relevant data
    state.getJoinTimes(sg);
    state.getLeaveTimes(sg);
    state.getLiberalAdds(og);

    // Query result is irrelevant
    queryResult = true;
    return;
}
```

5.6.2 Cost Analysis

To perform simulation-based cost analysis, we formalized actor machines for the conference program chair, authors, and reviewers, as well as workflows that describe how these actors interact. We formalized the relevant behavior by considering six distinct phases of time during which the actors carry out different sets of actions. These structures are presented in Section 4.5.3. Here we describe how they are implemented in Portuno.

To implement the desired actor machines in Portuno, we utilize a centralized class which determines the current phase from the current time in the simulation. Then, the `nextAction(Params params)` method of these actors calls the appropriate method for the phase. The following is this method and its helpers from `ActorPcAuthor : ActorMachine`, the actor machine for paper authors.

```
Action nextAction(Params params) {
    if (params.system.TIME_STOPPED) return null;

    Action a = q.poll();
    if (a != null) return a;

    currentPhase = PcPhases.phase(currentTime);
    switch (currentPhase) {
        case Submit:
            return submitAction(params);
        case Notify:
            return notifyAction(params);
    }
    return null;
}

Action submitAction(Params params) {
    if (submitted) return null;
    double coin = Simulation.rand.nextDouble();
    double timeLeft = PcPhases.phaseTimeLeft(currentTime) / params.delta - 3;
    if (coin <= 1.0/deltasLeft) {
        String id = actor.substring(1);
        submitted = true;
        q.add(new Action("addO", actor, "P" + id));
        q.add(new Action("LAdd", actor, "P" + id, "A" + id));
        return new Action("addG", actor, "A" + id);
    } else {
        return null;
    }
}
```

```
private Action notifyAction(Params params) {
    double coin = Simulation.rand.nextDouble();
    double deltasElapsed = PcPhases.phaseTimeElapsed(currentTime) / params.delta + 2;
    if (coin <= 1.0/deltasElapsed) {
        String id = actor.substring(1);
        return new Action("auth", actor, "S" + id, "A" + id);
    } else {
        return null;
    }
}
```

Using this `ActorMachine`, the authors in our simulation check for the two phases in which they have a role: submit and notify. During the submit phase, authors get exponentially more likely to submit their paper as the deadline approaches. When the submit finally occurs, the paper group is created immediately, and actions are queued to create the object representing the paper and liberal-add it to its group.

We note that this experiment uses a time-step of 1 hour, which balances goals of (1) being fine enough granularity, so that multiple actions are not necessary within a single time-step for the same actor; and (2) being large enough to remain efficient. However, in the PC workload, there *are* certain strings of administrative actions by the chair that always happen in quick succession. For such cases, rather than require the experiment to switch to an infeasibly fine time-step, we provide a switch in Portuno that allows an `ActorMachine` to indicate it needs to stop the simulation clock. This allows that actor to execute a string of actions within the same time-step and without interruption. The above example shows the author ensuring that she does not attempt to execute any action if time is stopped; below we show the chair stopping time while she completes a queue of actions.

```
Action nextAction(Params params) {
    Action a = q.poll();
    if (a == null) {
```

```
        params.system.TIME_STOPPED = false;
    } else {
        return a;
    }

    currentPhase = PcPhases.phase(currentTime);
    switch (currentPhase) {
        case Create:
            return createAction(params);
        case Recruit:
            return recruitAction(params);
        case Review:
            return reviewAction(params);
        case Discuss:
            return discussAction(params);
        case Notify:
            return notifyAction(params);
    }
    return null;
}

Action recruitAction(Params params) {
    if (!pcAdded) {
        params.system.TIME_STOPPED = true;
        pcAdded = true;
        for (int i = 0; i < params.numReviewers; i++) {
            q.add(new Action("SJoin", actor, "r" + i, "D"));
        }
    }
    return null;
```

```
}
```

5.7 SUMMARY

Toward enabling the cost analysis phase of suitability analysis, in Chapter 4 we proposed mathematical structures that enable representative trace sampling, as well as a formal framework for specifying cost measures and cost functions. Furthermore, we presented an algorithm for using these components to conduct simulation-based cost analysis. In this chapter, we describe in detail our Java-based simulation framework, Portuno, that implements all of these components into an extensible package that can be used for suitability analysis for a wide range of access control scenarios. We present a Monte Carlo driver for trend detection and a confidence-bounding driver for bounded-accuracy results. We evaluate our framework both formally, proving additional complexity bounds, and practically, demonstrating within the program committee case study how to represent realistically complex instances of the aforementioned structures within Portuno. As we continue to develop these tools, we will also investigate their use in other security domains, including evaluating cryptographic techniques for enforcing access controls on untrusted infrastructure and proposals for secure web communication. As such, we have shown the impact of suitability analysis on both formal as well as practical research.

6.0 CASE STUDY: DISSEMINATION-CENTRIC SYSTEMS FOR GROUP-CENTRIC SHARING

The Group-centric Secure Information Sharing (g-SIS) family of models has been proposed for modeling environments in which group dynamics dictate information-sharing policies and practices. Examples of such scenarios include secure meeting rooms where a participant only "hears" messages sent while she is a member, online periodicals where a subscriber is granted access to issues that are published during her subscription period (and is possibly allowed to retain such accesses even after her subscription is terminated), and the program committee workload considered in Chapters 4 and 5 in which reviewers and authors are brought together to share and discuss content. These contexts consider bringing subjects and objects together in groups to facilitate sharing, in contrast to traditional, dissemination-centric sharing models, which focus on attaching policies to resources that limit their flow from producer to consumer. The creators of g-SIS speculate that group-centric models may not be strictly more expressive than dissemination-centric models, but that they nevertheless have pragmatic efficiency advantages in group-centric scenarios [72]. In this chapter, we demonstrate the power of suitability analysis to answer such open questions, by formally and systematically testing these characteristics of an access control system's suitability for a scenario—expressiveness and cost—to evaluate the capabilities of dissemination-centric systems within group-centric workloads. We consider several common dissemination-centric systems, and identify security guarantees (most notably, *homomorphism*) that these systems are often unable to satisfy while implementing the wide range of behavior that is characteristic of the g-SIS models, except via impractical, convoluted encodings. Further, even more efficient implementations (admissible under relaxed security requirements) suffer from high storage and computational overheads. These observations support the practical and theoretical significance of the g-SIS

models as well as the suitability analysis techniques we have developed, and provide continued insight into techniques for evaluating and comparing access control systems in terms of both expressiveness and cost.[1]

6.1 INTRODUCTION

Group-centric Security Information Sharing (g-SIS) [71, 72] is a modeling paradigm and class of access control models that has been proposed for sharing environments in which users and resources are brought together in groups to facilitate collaboration and efficient exchange of information. Its creators contrast it with the traditional, *dissemination-centric* modeling paradigms that are currently used in access control and information sharing. In dissemination-centric sharing, emphasis is placed on attaching policies (and/or attributes that determine policy) to resources as they are created or made available. These policies then restrict which consumers can access the resources. In g-SIS, on the other hand, users are granted access to resources based on their temporal membership in groups—e.g., if object o is added to a group that a user u is a member of, u will be granted access to o. The rules for users who join later, objects that are removed, and users who leave are determined by the particular parameterization of g-SIS used.

For instance, consider the program committee workload presented in Definition 21. In this application, a reviewer may be added to a paper's review group so that she can provide her feedback, which necessitates her gaining access to the paper itself as well as older reviews. However, a reviewer may also need to leave a discussion group due to a conflict-of-interest, and upon return should *not* gain access to the conversation that occurred in the interim. Another group-centric scenario considers online magazines. A base subscription to such a periodical may only include access to issues that are published during one's subscription period, and only while one remains a subscriber. However, for an additional fee, the publisher may allow access to back issues, or continued access to existing content even after the subscription runs out.

[1] The material presented in this chapter was first published as [48].

Although g-SIS seems to represent these motivating scenarios rather elegantly, there has, to date, been no fully-functional implementation of the g-SIS models. This may partially be due to the following hypothesis by the creators of the paradigm [72]:

> *It may turn out that at a theoretical level, whatever dissemination-centric [systems] can achieve, group-centric [systems] can also achieve, and vice-versa. But at a pragmatic level, we believe these are significantly different approaches to information sharing.*

We note that this quote indicates a need for precisely the type of evaluation we propose in suitability analysis, considering both expressiveness for formal capabilities, and costs for pragmatic effectiveness. As such, in this chapter, we utilize our two-phase suitability analysis framework to evaluate the above hypothesis regarding the capabilities of dissemination-centric access control systems within the context of group-centric sharing workloads. Specifically, we break the above considerations into the following precise questions:

1. Which systems based on the g-SIS models can be *safely* implemented within, or simulated by, dissemination-centric access control systems?
2. How *strong* are the security properties that can be guaranteed by dissemination-centric systems when implementing workloads based on the g-SIS models?
3. How *efficiently* can dissemination-centric systems implement workloads based on the g-SIS models?
4. What practically-interesting instantiations of the g-SIS models *cannot* be safely and efficiently implemented by dissemination-centric systems?

To investigate the above questions, we formalize several instantiations of the g-SIS models. Some are based on mathematical extrema in the space of g-SIS instantiations and are intended to represent a diverse cross-section of the capabilities of these models overall. Others are based on realistic use cases to which g-SIS seems particularly well-suited. We then employ application-aware parameterized expressiveness [59] to answer questions #1 and #2. To evaluate question #3, we utilize cost analysis via Portuno as described in Chapters 4 and 5. Finally, we address question #4 by interpreting and analyzing the results of questions #1–3, discussing the various failings of the use of dissemination-centric techniques in group-centric environments.

Our analysis provides new insights into the relationship between group-centric and dissemination-centric sharing, and represents the first in-depth analysis into the use of g-SIS. We support the notion that g-SIS is a practically significant proposal by demonstrating the inability of traditional systems to satisfy many of its models safely and efficiently. Our analysis also demonstrates a deep, systematic examination and comparison of access control systems based on suitability. We therefore support our thesis statement, which says that suitability analysis has the potential to answer many realistic questions that arise when examining an application's access control needs.

The rest of this chapter is structured as follows. In Section 6.2, we introduce the g-SIS models. In Section 6.3, we describe the specific g-SIS instantiations that we will be analyzing. In Section 6.4, we present our parameterized expressiveness analysis. In Section 6.5, we discuss our cost analysis using Portuno. Finally, we discuss our results and reason about the drawbacks of utilizing dissemination-centric access control systems for group-centric workloads in Section 6.6 before summarizing in Section 6.7.

6.2 THE G-SIS MODELS

We now discuss background on g-SIS, the group-centric secure information sharing paradigm that inspires our workloads. The g-SIS models encompass a wide range of access control systems and behavior, and, on the surface, appear to subsume other group- and role-based access control and information sharing systems [98, 103, 104, 108]. As we have discussed, the motivating scenarios that inspired g-SIS include periodical subscriptions and secure message rooms. It is conceptually simpler to model such scenarios within g-SIS than by using dissemination-centric access control; that g-SIS is inherently more capable of representing such scenarios is a claim that we make and support in this chapter.

A major distinguishing feature of g-SIS is its preservation of a full membership record for groups. Users can join and leave groups, and objects can be added and removed from groups. The log of these events is used to decide whether a user can access an object. The basic operations each have numerous variants, ranging from *strict* to *fully liberal*. The semantics of

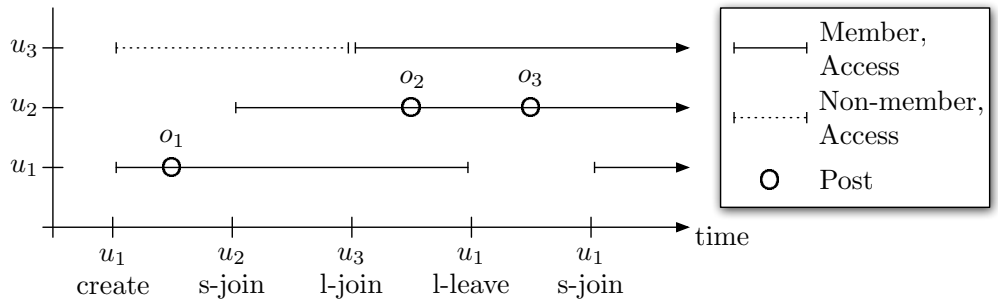


Figure 12: Example accesses in a single group in g-SIS

these variants depends on the type of action.

Users who perform a strict join to a group receive access only to objects added after they join, whereas a fully liberal join grants immediate access to all existing objects. A strict leave rescinds all of the user's accesses within the group; a liberal leave allows the user to retain access. Performing a strict add of a message to a group grants only current members access; a liberal add grants future members access as well. Finally, a strict remove rescinds access to the removed object from all users in the group, while a liberal remove allows users to retain access.

The application of these operations to the motivating scenarios (subscriptions, secure messaging) are, thus, fairly obvious. A subscription service can provide a base level of service with no access to back issues and no continued access after canceling (via strict join and strict leave). For an additional fee, users can access back issues (via liberal join) or maintain access to their issues after canceling (via liberal leave). As we demonstrated in Chapters 4 and 5 through the program committee workload, secure messaging can make various uses of combinations of strict and liberal actions: users can liberal leave before a discussion with which they have a conflict of interest and strict join at its conclusion; objects to which new members should not gain automatic access can be strict added while others are liberal added.

A graphical depiction of a membership/post record is shown in Fig. 12. Here, users u_2 and u_3 both join the group after u_1 posts o_1. However, u_2 performs a strict join whereas u_3 performs a liberal join, meaning that u_3 has access to o_1 while u_2 does not. Furthermore,

o_3 is added after u_1 liberal-leaves the group, and thus u_1 will not be able to access o_3, even after re-joining via strict join. We note, here, that after only a small number of operations in a *single* group, each of the users in the system has access to a different set of objects. If roles are used to implement this scenario, several will thus be needed to represent only this single group. This example shows that even simple group-centric scenarios can be complex to represent using dissemination-centric access control systems.

Although g-SIS also supports variants of actions that lie in between strict and fully liberal, in this chapter we focus on these extremes, since those in between tend to be application-specific. For example, in some systems a liberal (but not *fully* liberal) leave might allow a user to retain only *some* of their accesses when leaving, but the semantics of this action would have to be defined specific to the objects in the system. Thus, in this chapter, liberal actions are assumed to be fully liberal unless otherwise specified. In addition, g-SIS supports variants of join with different behavior for users who have left and re-joined a group. Lossy join may revoke some or all permissions retained from a past liberal leave; lossless join will not revoke any permissions. Restorative join may re-grant some or all permissions revoked by a past leave; non-restorative join will not re-grant any permissions. In this work, we focus on lossless, non-restorative joins unless otherwise specified.

6.3 INSTANTIATIONS OF G-SIS

In this section, we describe the particular g-SIS systems and workloads that form the basis of our analyses in Sections 6.4 and 6.5.

6.3.1 The g-SIS$_0$ Model

The g-SIS$_0$ model defines the state representation and queries for our g-SIS systems. It defines structures for storing the records for all combinations of the basic g-SIS strict and liberal actions (join, leave, add, remove). The authorization request is defined for non-restorative joins. It will thus be the basis for our instantiations of the framework. In the g-SIS$_0$ model,

states are comprised of the following fields.

- Sets S, O, G, and T of subjects, objects, groups, and times
- $>_T$, the total order on T
- $Time \in T$, the current time
- $StrictJoin \subseteq S \times G \times T$, the record of strict joins
- $LiberalJoin \subseteq S \times G \times T$, the record of liberal joins
- $StrictLeave \subseteq S \times G \times T$, the record of strict leaves
- $LiberalLeave \subseteq S \times G \times T$, the record of liberal leaves
- $StrictAdd \subseteq O \times G \times T$, the record of strict adds
- $LiberalAdd \subseteq O \times G \times T$, the record of liberal adds
- $StrictRemove \subseteq O \times G \times T$, the record of strict removes
- $LiberalRemove \subseteq O \times G \times T$, the record of liberal removes

The g-SIS$_0$ authorization requests are of the form $auth(s, o, g)$ for whether subject s has access to object o via group g. Here, like in the PC workload operational component, we define $authForward$, which applies in cases where the user joined the group before the object was added, and $authBackward$, which applies when the user joined after the object was added. Additional queries include $Member(s, g)$ for whether subject s is currently a member of group g and $Assoc(o, g)$ for whether object o is currently associated with group g. These are answered as follows.

- $Join(s, g, t) \triangleq StrictJoin(s, g, t) \vee LiberalJoin(s, g, t)$
- $Leave(s, g, t) \triangleq StrictLeave(s, g, t) \vee LiberalLeave(s, g, t)$
- $Add(o, g, t) \triangleq StrictAdd(o, g, t) \vee LiberalAdd(o, g, t)$
- $Remove(o, g, t) \triangleq StrictRemove(o, g, t) \vee LiberalRemove(o, g, t)$
- $Member(s, g) \triangleq \exists t_1.($
  $Join(s, g, t_1) \wedge$
  $\forall t_2.($
  $Leave(s, g, t_2) \Rightarrow t_1 > t_2$
  $)$
  $)$

- $Assoc(o, g) \triangleq \exists t_1.($
  $Add(o, g, t_1) \wedge$
  $\forall t_2.($
  $Remove(o, g, t_2) \Rightarrow t_1 > t_2$
  $)$
  $)$

- $authForward(s, o, g) \triangleq \exists t_1, t_2.($
  $Join(s, g, t_1) \wedge$
  $Add(o, g, t_2) \wedge$
  $t_2 > t_1 \wedge$
  $\forall t_3.($
  $Leave(s, g, t_3) \Rightarrow (t_1 > t_3 \vee t_3 > t_2) \wedge$
  $StrictLeave(s, g, t_3) \Rightarrow t_2 > t_3 \wedge$
  $StrictRemove(o, g, t_3) \Rightarrow t_2 > t_3$
  $)$
  $)$

- $authBackward(s, o, g) \triangleq \exists t_1, t_2.($
  $LiberalJoin(s, g, t_1) \wedge$
  $LiberalAdd(o, g, t_2) \wedge$
  $t_1 > t_2 \wedge$
  $\forall t_3.($
  $Remove(o, g, t_3) \Rightarrow (t_2 > t_3 \vee t_3 > t_1) \wedge$
  $StrictLeave(s, g, t_3) \Rightarrow t_1 > t_3 \wedge$
  $StrictRemove(o, g, t_3) \Rightarrow t_1 > t_3$
  $)$
  $)$

- $auth(s, o, g) \triangleq authForward(s, o, g) \vee authBackward(s, o, g)$

6.3.2 Extrema Systems

Top, bottom, and role-like g-SIS are systems in the g-SIS$_0$ model. Each of these systems contains a subset of the full set of actions supported in g-SIS$_0$, and represents a different extreme in terms of resulting behavior.

Top g-SIS contains only strict actions. Since all adds and joins are strict, there is no need for *authBackward*. Newer users to a group always have a subset of accesses of older users, users who leave retain no permissions to group-associated objects, and objects that are removed are no longer accessible by group members. In Top g-sis, states are represented as in the g-SIS$_0$ model. Commands are defined as follows.

- $addS(s)$: Add $S(s)$
- $delS(s)$: Remove $S(s)$
- $addG(g)$: Add $G(g)$
- $delG(g)$: Remove $G(g)$
- $addO(o)$: Add $O(o)$
- $delO(o)$: Remove $O(o)$
- $strictJoin(s, g)$: Remove $Time(t)$, add $StrictJoin(s, g, t), Time(t + 1)$
- $strictLeave(s, g)$: Remove $Time(t)$, add $StrictLeave(s, g, t), Time(t + 1)$
- $strictAdd(o, g)$: Remove $Time(t)$, add $StrictAdd(o, g, t), Time(t + 1)$
- $strictRemove(o, g)$: Remove $Time(t)$, add $StrictRemove(o, g, t), Time(t + 1)$

Furthermore, since this system is more restricted than g-SIS$_0$ (it lacks liberal commands, so we never need to inspect the corresponding liberal records), we can use the following authorization formula, which is simpler than that of the full g-SIS$_0$ model.

- $auth(s, o, g) \triangleq \exists t_1, t_2.($
  $StrictJoin(s, g, t_1) \wedge$
  $StrictAdd(o, g, t_2) \wedge$
  $t_2 > t_1 \wedge$
  $\forall t_3.($
  $StrictLeave(s, g, t_3) \Rightarrow t_1 > t_3 \wedge$
  $StrictRemove(o, g, t_3) \Rightarrow t_2 > t_3$

)

)

Bottom g-SIS contains only liberal actions. Thus, a subject is granted access to an object as long as they both belonged to a group at the same time at some point (currently or in the past), however briefly. Access to added objects is granted to current and new users, but once an object is removed no new users are granted access. Thus, new users tend to have fewer accesses than older users. The following commands are defined.

- $addS(s)$: Add $S(s)$
- $delS(s)$: Remove $S(s)$
- $addG(g)$: Add $G(g)$
- $delG(g)$: Remove $G(g)$
- $addO(o)$: Add $O(o)$
- $delO(o)$: Remove $O(o)$
- $liberalJoin(s, g)$: Remove $Time(t)$, add $LiberalJoin(s, g, t), Time(t + 1)$
- $liberalLeave(s, g)$: Remove $Time(t)$, add $LiberalLeave(s, g, t), Time(t + 1)$
- $liberalAdd(o, g)$: Remove $Time(t)$, add $LiberalAdd(o, g, t), Time(t + 1)$
- $liberalRemove(o, g)$: Remove $Time(t)$, add $LiberalRemove(o, g, t), Time(t + 1)$

As before, we can use a simpler authorization procedure that does not inspect the portions of the state that can never be changed.

- $auth(s, o, g) \triangleq \exists t_1, t_2.($
  $LiberalJoin(s, g, t_1) \wedge$
  $LiberalAdd(o, g, t_2) \wedge$
  (
    $t_2 > t_1 \wedge$
    $\forall t_3.(LiberalLeave(s, g, t_3) \Rightarrow t_1 > t_3 \vee t_3 > t_2)$
) $\vee$ (
    $t_1 > t_2 \wedge$
    $\forall t_3.(LiberalRemove(o, g, t_3) \Rightarrow t_2 > t_3 \vee t_3 > t_1)$

)

)

Role-like g-SIS, lastly, is an approximation of a role-based access control system within g-SIS. It allows liberal join and add actions and strict leave and remove. Thus, all current members have access to all current objects, but users who leave lose all access, and objects that are removed are revoked from all users. The commands are as follows.

- $addS(s)$: Add $S(s)$
- $delS(s)$: Remove $S(s)$
- $addG(g)$: Add $G(g)$
- $delG(g)$: Remove $G(g)$
- $addO(o)$: Add $O(o)$
- $delO(o)$: Remove $O(o)$
- $liberalJoin(s, g)$: Remove $Time(t)$, add $LiberalJoin(s, g, t), Time(t + 1)$
- $strictLeave(s, g)$: Remove $Time(t)$, add $StrictLeave(s, g, t), Time(t + 1)$
- $liberalAdd(o, g)$: Remove $Time(t)$, add $LiberalAdd(o, g, t), Time(t + 1)$
- $strictRemove(o, g)$: Remove $Time(t)$, add $StrictRemove(o, g, t), Time(t + 1)$

The authorization procedure is as follows.

- $auth(s, o, g) \triangleq \exists t_1, t_2.($
  $LiberalJoin(s, g, t_1) \wedge$
  $LiberalAdd(o, g, t_2) \wedge$
  $\forall t_3.($
  $StrictLeave(s, g, t_3) \Rightarrow t_1 > t_3 \wedge$
  $StrictRemove(o, g, t_3) \Rightarrow t_2 > t_3$
)
)

6.3.3 Workloads

In addition to the above extrema systems, we also study several more realistic parameterizations that reflect how g-SIS might be used in practice. Conceptually, these lie somewhere in

the g-SIS spectrum between the extrema systems defined above, and represent real-world usages of group-centric techniques. We formalize these as workloads.

The **PC** workload is an instance of the "secure message room" example use case of g-SIS [71, 72] that is defined to model academic program committee discussions. It was discussed at length in Chapters 4 and 5.

The Playstation Plus (PSP) premium gaming service [93] uses temporal constraints to decide accesses, and is thus a natural fit for modeling in g-SIS. **PSP** uses a g-SIS model with an extension over g-SIS$_0$: the *auth* query supports *restorative* joins. Subscribers liberal join, and strict leave when canceling. If a user cancels and later joins again, she is re-granted access to all objects she had before leaving (except those that have been strict removed). Managers liberal add promotions (free games and discounts). When a promotion is complete, free games are liberal removed (users who are members at the time a free game is available may continue to access it as long as they are a member), while discounts are strict removed and thus become inaccessible to all users. Trace restrictions allow users to subscribe in 3-month increments. The managers add and remove several promotions each week, maintaining the same total number for each group.

PCP considers the same state as g-SIS$_0$, with the following commands.

- $addS(s)$: Add $S(s)$
- $delS(s)$: Remove $S(s)$
- $addO(o)$: Add $O(o)$
- $delO(o)$: Remove $O(o)$
- $liberalJoin(s, g)$: Remove $Time(t)$, add $LiberalJoin(s, g, t), Time(t + 1)$
- $strictLeave(s, g)$: Remove $Time(t)$, add $StrictLeave(s, g, t), Time(t + 1)$
- $liberalAdd(o, g)$: Remove $Time(t)$, add $LiberalAdd(o, g, t), Time(t + 1)$
- $strictRemove(o, g)$: Remove $Time(t)$, add $StrictRemove(o, g, t), Time(t + 1)$
- $liberalRemove(o, g)$: Remove $Time(t)$, add $LiberalRemove(o, g, t), Time(t + 1)$

Note that, due to its having restorative rejoin, this system is not a member of the g-SIS$_0$ model and therefore the following *auth* definition is not a simplified or special case of the one used in g-SIS$_0$.

- $auth(s, o, g) \triangleq Member(s, g) \wedge$

$\exists t_1, t_2.($

$LiberalJoin(s, g, t_1) \wedge$

$LiberalAdd(o, g, t_2) \wedge$

$\forall t_3.(StrictRemove(o, g, t_3) \Rightarrow t_2 > t_3) \wedge$

$($

$t_2 > t_1 \wedge$

$\forall t_3.(StrictLeave(s, g, t_3) \Rightarrow (t_1 > t_3 \vee t_3 > t_2))$

$) \vee ($

$t_1 > t_2 \wedge$

$\forall t_3.(Remove(o, g, t_3) \Rightarrow (t_2 > t_3 \vee t_3 > t_1))$

$)$

$)$

6.4 EXPRESSIVENESS ANALYSIS

In this section, we describe the details of our expressiveness analysis using parameterized expressiveness and present a summary of the results. Demonstrative proofs are included here. Remaining proofs have been published in a technical report [47].

We use slightly different techniques for evaluating against g-SIS systems vs. workloads. To evaluate against a workload, we construct an access control implementation as described in previous chapters. When evaluating against a system, on the other hand, we use an access control reduction: a set of mappings allowing us to prove that a system $\mathcal{T}$ is at least as expressive as system $\mathcal{S}$ with respect to a particular set of security guarantees $\mathcal{G}$. This is written $\mathcal{S} \leq^{\mathcal{G}} \mathcal{T}$, and indicates that any workload $\mathcal{W}$ that can be implemented in $\mathcal{S}$ with guarantees $\mathcal{G}$ can also be implemented in $\mathcal{T}$ with $\mathcal{G}$. We use the form of reduction from parameterized expressiveness [59], in which a reduction from $\mathcal{S}$ to $\mathcal{T}$, $\langle \sigma_\Gamma, \sigma_Q \rangle$, defines a state mapping σ_Γ and a query mapping σ_Q where the state mapping preserves the query mapping $(\forall \gamma \in States(\mathcal{S})\ \ Th(\gamma) = \sigma_Q(Th(\sigma_\Gamma(\gamma))))$. The additional conditions put on the reduction

differ based on the set of security guarantees it should preserves.

6.4.1 Security Guarantees

In this work, we consider the following security guarantees.

Correctness Correctness is a bare minimum requirement for any implementation, as defined in Definition 24.

AC-Preservation AC-Preservation says that the native authorization procedure of the candidate system must be used to map workload requests to answers. It is discussed in Section 4.5.2.

Weak AC-Preservation This guarantee is a weaker version of AC-preservation [59], and is defined in Definition 25.

Safety A safe implementation is one that does not grant or revoke unnecessary permissions during the execution of the image of a single workload command. It is defined in Definition 26.

Homomorphism The homomorphic property eliminates implementations that abuse system state by encoding workload state in a way that is fragile to string substitutions. Without this requirement, an implementation can, e.g., store unbounded state in a single user name by encoding whole relations as a single string.

Definition 27 (Homomorphism)**.** A homomorphic mapping f is one in which $f(x)[v] = f(x[v])$ for any constant string substitution $[v]$.

Given a workload, $\mathcal{W} = \langle \mathcal{A}, \mathcal{T} \rangle$, a system, $\mathcal{S}$, and an implementation, $\langle \sigma_\Gamma, \sigma_\Psi, \sigma_Q \rangle$, σ_Q is homomorphic if each of $\sigma\Gamma$, σ_Ψ, and σ_Q is homomorphic. $\Diamond$

Intuitively, homomorphism requires that data elements be opaque, and that the symbol representing any element (e.g., user, object, role) can be substituted for any other without affecting the behavior of the system. Hinrichs et al. [59] define the programming language HPL (Homomorphic Programming Language) such that any mapping expressed in HPL is homomorphic.

6.4.2 Dissemination-Centric Systems

We choose several dissemination-centric access control systems as candidates for implementing the group-centric workloads described in Section 6.3. As in Section 4.5, we focus on role- and group-based models. While these access control models are dissemination-centric, they provide a level of indirection between subjects and objects that enables greater expressiveness than models based on the access matrix or access control lists [89,98]. Comparing to group-enabled dissemination-centric access control systems enables our analysis to more directly compare the effect of the group-centric paradigm, whereas comparing to non-group-enabled systems would be more likely to highlight simply the advantage of the additional level of indirection provided by groups. Thus, we evaluate the following dissemination-centric systems.

RBAC RBAC_0 is the most basic role-based access control system proposed in the RBAC standard [103, 104]. It is defined in Example 2.

Hierarchical RBAC RBAC_1 includes a hierarchical structure over roles, and is defined in Definition 22.

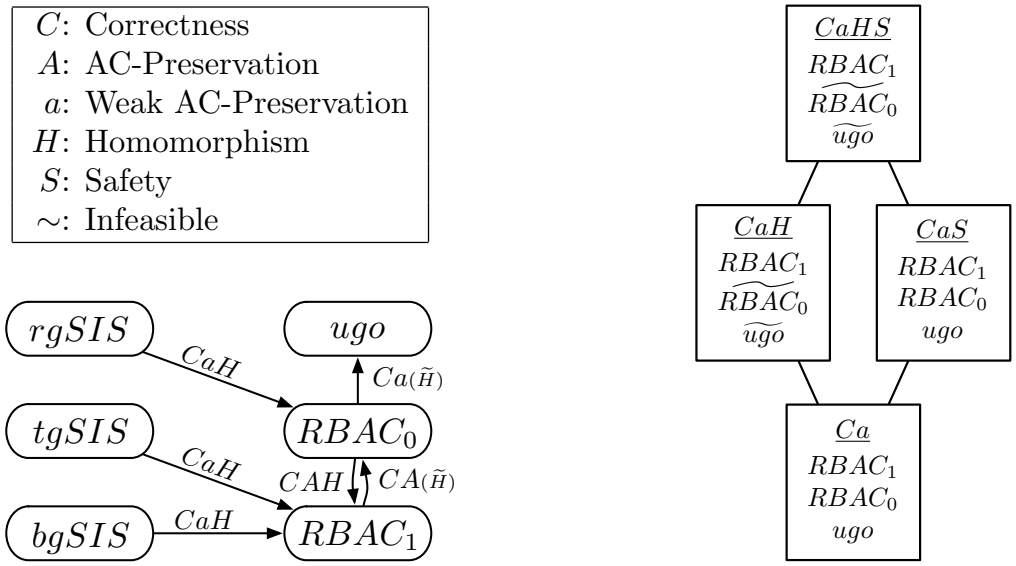
UNIX Permissions Finally, the *ugo* system is based on the *user, group, other* system of access control in UNIX. It is defined in Definition 23.

Thus, we evaluate standard, widely-deployed access control systems, in both the industrial and consumer spaces. These systems are likely candidates for a system administrator who desires to implement a group-centric workload using available and trusted access control mechanisms.

6.4.3 Expressiveness via System Reductions

We now present a summary of the system reductions proving expressiveness statements comparing the chosen dissemination-centric systems and the g-SIS extrema systems. A summary of the expressiveness reductions, including a key to our shorthand for denoting the guarantees a reduction satisfies, is shown in Figure 13a. We now discuss the major results depicted in this figure.

First, we note that it is simple to construct a reduction from role-like g-SIS ($rgSIS$) to



(a) Reductions (b) Implementations

Figure 13: Expressiveness analysis results

RBAC_0 that satisfies all considered security guarantees (correctness, weak AC-preservation, homomorphism). Role-like g-SIS has only liberal add and join operations and strict leave and remove operations, and thus its *auth* only depends on the current group members and associated documents. In this scenario, RBAC_0 can simply simulate groups using roles.

Theorem 6. $rgSIS \leq^{CaH} \mathsf{RBAC}_0$.

Proof. We show that there exists a reduction $\langle \sigma_\Gamma, \sigma_Q \rangle$ from role-like g-SIS to RBAC_0 where:

- σ_Γ preserves σ_Q, is pseudo-injective, preserves reachability, and is homomorphic
- σ_Q is weak AC-preserving and homomorphic

Thus, $rgSIS \leq^{CaH} \mathsf{RBAC}_0$ (RBAC_0 is at least as expressive as role-like g-SIS with respect to correctness, weak AC-preservation, and homomorphism).

We present the reduction, $\langle \sigma_\Gamma, \sigma_Q \rangle$. First, σ_Γ maps the g-SIS state $\langle S, O, G, T, Time, LiberalJoin, StrictLeave, LiberalAdd, StrictRemove \rangle$ to an RBAC_0 state of the form $\langle U,$

$R, P, UR, PA\rangle$. This mapping is described by the following HPL method (and is thus homomorphic).

```
for each (S(s) ∈ γ)
   output(U(s))
for each (G(g) ∈ γ)
   output(R(g))
for each (O(o) ∈ γ)
   output(P(o))

for each (LiberalJoin(s, g, t) ∈ γ)
   Old = {}
   for each (StrictLeave(s, g, x) ∈ γ)
      If >T(x, t) ∈ γ then
         Old = {<s, g>}
      endif
   If <s, g> ∉ Old
      output(UR(s, g))
   endif

for each (LiberalAdd(o, g, t) ∈ γ)
   Old = {}
   for each (StrictRemove(o, g, x) ∈ γ)
      If >T(x, t) ∈ γ then
         Old = {<o, g>}
      endif
   If <o, g> ∉ Old
      output(PA(g, o))
   endif
```

σ_Q is defined as follows.

$$\sigma_Q(Member(s,g),\gamma) = UR(s,g) \in \gamma$$
$$\sigma_Q(Assoc(o,g),\gamma) = PA(g,o) \in \gamma$$
$$\sigma_Q(auth(s,o,g),\gamma) = UR(s,g), PA(g,o) \in \gamma$$

This query mapping clearly contains no string manipulation and is thus homomorphic.

Let γ be an arbitrary $rgSIS$ state and $r = auth(s,o,g)$ an arbitrary $rgSIS$ request, and let $f(auth(s,o,g)) = auth(s,o)$ be a request transform. Assume $\sigma_Q(r, \sigma_\Gamma(\gamma)) = \text{TRUE}$. Then, $UR(s,g) \in Th(\sigma_\Gamma(\gamma)) \wedge PA(g,o) \in Th(\sigma_\Gamma(\gamma))$. Thus, it is clear that $\exists r_1.(UR(s, r_1) \in Th(\sigma_\Gamma(\gamma)) \wedge PA(r_1,o) \in Th(\sigma_\Gamma(\gamma)))$, and therefore $\sigma_\Gamma(\gamma) \vdash f(r)$.

Now let γ be an arbitrary $rgSIS$ state, $r' = auth(u,p)$ an arbitrary RBAC_0 request, and f the request transform defined above. Assume $\sigma_\Gamma(\gamma) \vdash r'$. Then, $\exists r_1.(UR(s,r_1) \in Th(\sigma_\Gamma(\gamma)) \wedge PA(r_1,o) \in Th(\sigma_\Gamma(\gamma)))$. Finally, $f(u,p,r_1) = (u,p)$, and $\sigma_Q(auth(u,p,r_1), \sigma_\Gamma(\gamma)) = \text{TRUE}$. Thus, σ_Q is weak AC-preserving with transform $f(s,o,g) = (s,o)$.

We show that σ_Γ preserves σ_Q (for all $rgSIS$ states γ, $Th(\gamma) = \sigma_Q(Th(\sigma_\Gamma(\gamma))))$ by contradiction. Assume that there is some $rgSIS$ state γ and query q such that the value of q in γ is the opposite of the value of $\sigma_Q(q)$ in $\sigma_\Gamma(\gamma)$. We show that, for each of the query forms of $rgSIS$, this assumption leads to contradiction.

- **Member** Assume $\gamma \vdash Member(s,g)$ and $\sigma_\Gamma(\gamma) \nvdash \sigma_Q(Member(s,g))$. Then, $\exists t_1.(LiberalJoin(s,g,t_1) \in Th(\gamma) \wedge \forall t_2.(LiberalLeave(s,g,t_2) \in Th(\gamma) \Rightarrow t_1 > t_2))$ (s has joined g and not left). By σ_Γ, $UR(s,g) \in Th(\sigma_\Gamma(\gamma))$. Thus, by σ_Q, $\sigma_Q(Member(s, g), Th(\sigma_\Gamma(\gamma))) = \text{TRUE}$, which is a contradiction on the assumption that $\sigma_\Gamma(\gamma) \nvdash \sigma_Q(Member(s,g))$.

  Assume instead that $\gamma \nvdash Member(s,g)$ and $\sigma_\Gamma(\gamma) \vdash \sigma_Q(Member(s,g))$. Then, either $\exists t_1.(LiberalLeave(s,g,t_1) \in Th(\gamma) \wedge \forall t_2.(LiberalJoin(s,g,t_2) \in Th(\gamma) \Rightarrow t_1 > t_2))$ (s has left g and not returned), or $\forall t_1.(LiberalJoin(s,g,t_1) \notin Th(\gamma))$ (s has not joined g). By σ_Γ, in either case, $UR(s,g) \notin Th(\sigma_\Gamma(\gamma))$. Thus, by σ_Q, $\sigma_Q(Member(o,g), \sigma_\Gamma(\gamma)) = \text{FALSE}$, which is a contradiction on the assumption that $\sigma_\Gamma(\gamma) \vdash \sigma_Q(Member(s,g))$.

- **Assoc** Assume $\gamma \vdash Assoc(o, g)$ and $\sigma_\Gamma(\gamma) \nvdash \sigma_Q(Assoc(o, g))$. Then, $\exists t_1.(LiberalAdd(o, g, t_1) \in Th(\gamma) \wedge \forall t_2.(LiberalRemove(o, g, t_2) \in Th(\gamma) \Rightarrow t_1 > t_2))$ (o was added to g and not removed). Thus, by σ_Γ, $PA(g, o) \in Th(\sigma_\Gamma(\gamma)))$. By σ_Q, $\sigma_Q(Assoc(o, g), \sigma_\Gamma(\gamma)) = \text{TRUE}$, which is a contradiction on the assumption that $\sigma_\Gamma(\gamma) \nvdash \sigma_Q(Assoc(o, g))$.
  Assume instead that $\gamma \nvdash Assoc(o, g)$ and $\sigma_\Gamma(\gamma) \vdash \sigma_Q(Assoc(o, g))$. Then, either $\exists t_1.(LiberalRemove(o, g, t_1) \in Th(\gamma) \wedge \forall t_2.(LiberalAdd(o, g, t_2) \in Th(\gamma) \Rightarrow t_1 > t_2))$ (s was removed from g and not re-added), or $\forall t_1.(LiberalAdd(o, g, t_1) \notin Th(\gamma))$ (o has not added to g). By σ_Γ, in either case, $PA(g, o) \notin Th(\sigma_\Gamma(\gamma)))$. Thus, by σ_Q, $\sigma_Q(Assoc(o, g), \sigma_\Gamma(\gamma)) = \text{FALSE}$, which is a contradiction on the assumption that $\sigma_\Gamma(\gamma) \vdash \sigma_Q(Assoc(o, g))$.
- **auth** Assume $\gamma \vdash auth(s, o, g)$ and $\sigma_\Gamma(\gamma) \nvdash \sigma_Q(auth(s, o, g))$. Then, $\exists t_1.(LiberalJoin(s, g, t_1) \in Th(\gamma) \wedge \forall t_2.(LiberalLeave(s, g, t_2) \in Th(\gamma) \Rightarrow t_1 > t_2))$ (s has joined g and not left), and $\exists t_1.(LiberalAdd(o, g, t_1) \in Th(\gamma) \wedge \forall t_2.(LiberalRemove(o, g, t_2) \in Th(\gamma) \Rightarrow t_1 > t_2))$ (o was added to g and not removed). By σ_Γ, $UR(s, g) \in Th(\sigma_\Gamma(\gamma)) \wedge PA(g, o) \in Th(\sigma_\Gamma(\gamma))$. Thus, by σ_Q, $\sigma_Q(auth(s, o, g), \sigma_\Gamma(\gamma)) = \text{TRUE}$, which is a contradiction on the assumption that $\sigma_\Gamma(\gamma) \nvdash \sigma_Q(auth(s, o, g))$
  Assume instead that $\gamma \nvdash auth(s, o, g)$ and $\sigma_\Gamma(\gamma) \vdash \sigma_Q(auth(s, o, g))$. Then, there are four possibilities which we consider in pairs. If $\exists t_1.(LiberalLeave(s, g, t_1) \in Th(\gamma) \wedge \forall t_2.(LiberalJoin(s, g, t_2) \in Th(\gamma) \Rightarrow t_1 > t_2))$ (s has left g and not returned), or $\forall t_1.(LiberalJoin(s, g, t_1) \notin Th(\gamma))$ (s has not joined g), then by σ_Γ, $UR(s, g) \notin Th(\sigma_\Gamma(\gamma))$, and thus by σ_Q, $\sigma_Q(auth(s, o, g), \sigma_\Gamma(\gamma)) = \text{FALSE}$ (a contradiction). If instead $\exists t_1.(LiberalRemove(o, g, t_1) \in Th(\gamma) \wedge \forall t_2.(LiberalAdd(o, g, t_2) \in Th(\gamma) \Rightarrow t_1 > t_2))$ (s was removed from g and not re-added), or $\forall t_1.(LiberalAdd(o, g, t_1) \notin Th(\gamma))$ (o has not added to g), then by σ_Γ, $PA(g, o) \notin Th(\sigma_\Gamma(\gamma)))$, and thus by σ_Q, $\sigma_Q(auth(s, o, g), Th(\sigma_\Gamma(\gamma))) = \text{FALSE}$.

Thus, by contradiction, σ_Γ preserves σ_Q.

For all $rgSIS$ states γ, γ', if γ' is reachable from γ, then there exists a sequence of commands $\langle \psi_1, \psi_2, \ldots, \psi_n \rangle$ such that $terminal(\gamma, \psi_1 \circ \psi_2 \circ \cdots \circ \psi_n) = \gamma'$. We will show that, for any $rgSIS$ state γ and command ψ, $\sigma_\Gamma(next(\gamma, \psi))$ is reachable from $\sigma_\Gamma(\gamma)$ via RBAC_0 commands. By induction, this will show that for each intermediate $rgSIS$ state γ_i between γ and γ', $\sigma_\Gamma(\gamma_i)$ is reachable from $\sigma_\Gamma(\gamma)$ and ultimately that $\sigma_\Gamma(\gamma')$ is reachable from $\sigma_\Gamma(\gamma)$

(i.e., that σ_Γ preserves reachability).

Given $rgSIS$ state γ and command ψ, $\gamma' = next(\gamma, \psi)$ is the state resulting from executing command ψ in state γ.

- If ψ is an instance of $addS(s)$, then $\gamma' = next(\gamma, \psi) = \gamma \cup S(s)$. By σ_Γ, this maps in RBAC_0 to state $\sigma_\Gamma(\gamma') = \sigma_\Gamma(\gamma \cup S(s)) = \sigma_\Gamma(\gamma) \cup U(s)$. By RBAC_0's $next$ relation, $next(\sigma_\Gamma(\gamma), addU(s)) = \sigma_\Gamma(\gamma) \cup U(s)$. Thus, if ψ is an instance of $addS(s)$, $\sigma_\Gamma(\gamma')$ is reachable from $\sigma_\Gamma(\gamma)$ via execution of $addU(s)$. A similar argument holds for instances of $addG(g)$ and $addO(o)$ (with reachability in RBAC_0 via $addR(g)$ and $addP(o)$, respectively).
- If ψ is an instance of $delS(s)$, then $\gamma' = \gamma \setminus (S(s) \cup Entries(\gamma, s))$, where $Entries(\gamma, s)$ denotes the set of all state tuples in γ involving s[2]. By σ_Γ, this maps in RBAC_0 to state $\sigma_\Gamma(\gamma') = \sigma_\Gamma(\gamma \setminus (S(s) \cup Entries(s))) = \sigma_\Gamma(\gamma) \setminus (U(s) \cup Entries(\sigma_\Gamma(\gamma), s))$. By RBAC_0's $next$ relation, $next(\sigma_\Gamma(\gamma), delU(s)) = \sigma_\Gamma(\gamma) \setminus (U(s) \cup Entries(s))$. Thus, if ψ is an instance of $delS(s)$, $\sigma_\Gamma(\gamma')$ is reachable from $\sigma_\Gamma(\gamma)$ via execution of $delU(s)$. A similar argument holds for instances of $delG(g)$ and $delO(o)$ (with reachability in RBAC_0 via $delR(g)$ and $delP(o)$, respectively).
- If ψ is an instance of $liberalJoin(s, g)$, then $\gamma' = \gamma \cup LiberalJoin(s, g, t) \cup Time(t + 1) \setminus Time(t)$. By σ_Γ, this maps in RBAC_0 to state $\sigma_\Gamma(\gamma') = \sigma_\Gamma(\gamma) \cup UR(s, g)$. By RBAC_0's $next$ relation, $next(\sigma_\Gamma(\gamma), assignUser(s, g)) = \sigma_\Gamma(\gamma) \cup UR(s, g)$. Thus, if ψ is an instance of $liberalJoin$, $\sigma_\Gamma(\gamma')$ is reachable from $\sigma_\Gamma(\gamma)$ via execution of $assignUser(s, g)$. A similar argument holds for instances of $liberalAdd(o, g)$ with reachability via $assignPermission(g, o)$.
- If ψ is an instance of $strictLeave(s, g)$, then $\gamma' = \gamma \cup StrictLeave(s, g, t) \cup Time(t + 1) \setminus Time(t)$. By σ_Γ, this maps in RBAC_0 to $\sigma_\Gamma(\gamma') = \sigma_\Gamma(\gamma) \setminus UR(s, g)$. By RBAC_0's $next$ relation, $next(\sigma_\Gamma(\gamma), revokeUser(s, g)) = \sigma_\Gamma(\gamma) \setminus UR(s, g)$. Thus, if ψ is an instance of $strictLeave(s, g)$, $\sigma_\Gamma(\gamma')$ is reachable from $\sigma_\Gamma(\gamma)$ via execution of $revokeUser(s, g)$. A similar argument holds for instances of $strictRemove(o, g)$ with reachability via $revokePermission(g, o)$.

Thus, for any $rgSIS$ state γ and command ψ, $\sigma_\Gamma(next(\gamma, \psi))$ is reachable from $\sigma_\Gamma(\gamma)$ via

[2] In the case of $rgSIS$, $Entries(\gamma, s)$ for subject s is $\{LiberalJoin(s, g, t) \mid LiberalJoin(s, g, t) \in \gamma\} \cup \{StrictLeave(s, g, t) \mid StrictLeave(s, g, t) \in \gamma\}$.

RBAC_0 commands. By induction, for any $rgSIS$ states γ and γ', if γ' is reachable from γ, then $\sigma_\Gamma(\gamma')$ is reachable from $\sigma_\Gamma(\gamma)$. Thus, we have shown that σ_Γ preserves reachability.

Finally, we show that σ_Γ is pseudo-injective. We first inspect σ_Γ to identify what set of differences may exist between γ_1 and γ_2 and still allow $\sigma_\Gamma(\gamma_1) = \sigma_\Gamma(\gamma_2)$. We then show that, if two $rgSIS$ states γ_1 and γ_2 are identical modulo this set of differences, then for any $rgSIS$ command, ψ, $next(\gamma_1, \psi)$ and $next(\gamma_2, \psi)$ are also identical modulo this set.

Assume $\sigma_\Gamma(\gamma_1) = \sigma_\Gamma(\gamma_2)$. The state mapping, σ_Γ, stores S, G, and O directly in U, R, and P, respectively. The set of times, T, and current time, $Time$, are not stored in RBAC_0. However, T is immutable. Thus, states γ_1 and γ_2 must not differ in S, G, and O, and are guaranteed not to differ in T, but may differ in current time $Time$.

Regarding the handling of $LiberalJoin$ and $StrictLeave$ by σ_Γ: These relations of $rgSIS$ are considered in combination. Rather than consider all joins and leaves a particular subject s has performed for a particular group g, σ_Γ only considers the most recent join or leave event. If in γ_1, s has joined g and has not since left, by σ_Γ, $UR(s, g) \in Th(\sigma_\Gamma(\gamma_1))$. If in γ_1, s has left g and has not since re-joined, or if s never joined g, $UR(s, g) \notin Th(\sigma_\Gamma(\gamma_1))$. Since past entries in $LiberalJoin$ and $StrictLeave$ are not considered by σ_Γ, γ_1 and γ_2 can have any contents in these relations, so long as they agree on the most recent event for each s and g (i.e., whether each s is currently a member of each g).

For identical reasons regarding the storage of $\langle g, o\rangle \in PA$ by σ_Γ, γ_1 and γ_2 can have any contents in $LiberalAdd$ and $StrictRemove$ as long as they agree, for each o and g, whether object o is currently in group g.

Finally, $StrictJoin$, $LiberalLeave$, $StrictAdd$, and $LiberalRemove$ are empty and immutable over the commands of $rgSIS$. Thus, γ_1 and γ_2 are guaranteed to be identical in these relations.

Now, we examine each of the differences that may exist between γ_1 and γ_2 to ensure that, after execution of any command in both, the resulting states will also differ only in these ways.

- **Current time** Consider two $rgSIS$ states γ_1 and γ_2 that differ in the current time $Time(t)$. Since the current time is always greater than all times in the join, leave, add, and remove logs, the difference in current time between these states will propogate only to

a difference in the absolute time of future events in the logs; relative order of future events will be preserved. Thus, for any $tgSIS$ command ψ, the differences between $next(\gamma_1, \psi)$ and $next(\gamma_2, \psi)$ will only be in current time and absolute time of events. Thus, $next(\gamma_1, \psi)$ and $next(\gamma_2, \psi)$ will differ only in ways that ensure $\sigma_\Gamma(next(\gamma_1, \psi)) = \sigma_\Gamma(next(\gamma_2, \psi))$.

- **Absolute event times** Consider $rgSIS$ states γ_1 and γ_2 in which some event occured at different absolute times but in the same order relative to other events. Since $rgSIS$ commands can only add events with the most recent timestamp, and do not consider the absolute times of past events, for any $rgSIS$ command ψ, $next(\gamma_1, \psi)$ and $next(\gamma_2, \psi)$ will differ only in this altered timestamp. Thus, $\forall\psi.(\sigma_\Gamma(next(\gamma_1, \psi)) = \sigma_\Gamma(next(\gamma_2, \psi)))$.
- **Relative inter-group event times** Consider $rgSIS$ states γ_1 and γ_2 in which two adjacent events (operating on different groups) swap times. Since γ_1 and γ_2 are adjacent, this swap does not affect the relative times of events within any particular group, and more importantly does not alter the most recent event for a particular s, g or o, g pair. Thus, for any $tgSIS$ command ψ, $next(\gamma_1, \psi)$ and $next(\gamma_2, \psi)$ also differ only in these events' relative times. Therefore, $\forall\psi.(\sigma_\Gamma(next(\gamma_1, \psi)) = \sigma_\Gamma(next(\gamma_2, \psi)))$.
- **Past events** Consider two $rgSIS$ states γ_1 and γ_2 which have different sets of $LiberalJoin$, $StrictLeave$, $LiberalAdd$, and $StrictRemove$, but agree on the most recent event for each s, g (join or leave) and o, g (add or remove). New events can only be added to the state with the most recent time, and thus the different records between γ_2 and γ_1 will never impact a future decision—once events become irrelevant, they cannot become relevant again. Thus, for any ψ, $next(\gamma_1, \psi)$ and $next(\gamma_2, \psi)$ will continue to differ only in these previous events, and thus $\sigma_\Gamma(next(\gamma_1, \psi)) = \sigma_\Gamma(next(\gamma_2, \psi))$.

We have enumerated the ways in which two distinct $rgSIS$ states γ_1 and γ_2 can map to the same RBAC_0 state (i.e., $\gamma_1 \neq \gamma_2$, $\sigma_\Gamma(\gamma_1) = \sigma_\Gamma(\gamma_2)$). In each case, we show that $\forall\psi.(\sigma_\Gamma(next(\gamma_1, \psi)) = \sigma_\Gamma(next(\gamma_2, \psi)))$, that is, that γ_1 and γ_2 are functionally equivalent with respect to σ_Γ. Thus, we have shown that σ_Γ is pseudo-injective.

Thus, we have shown that σ_Γ preserves σ_Q, is pseudo-injective, preserves reachability, and is homomorphic; and that σ_Q is weak AC-preserving and homomorphic.

$\therefore \langle \sigma_\Gamma, \sigma_Q \rangle$ is a reduction from $rgSIS$ to RBAC_0 which shows $rgSIS \leq^{CaH} \mathsf{RBAC}_0$. □

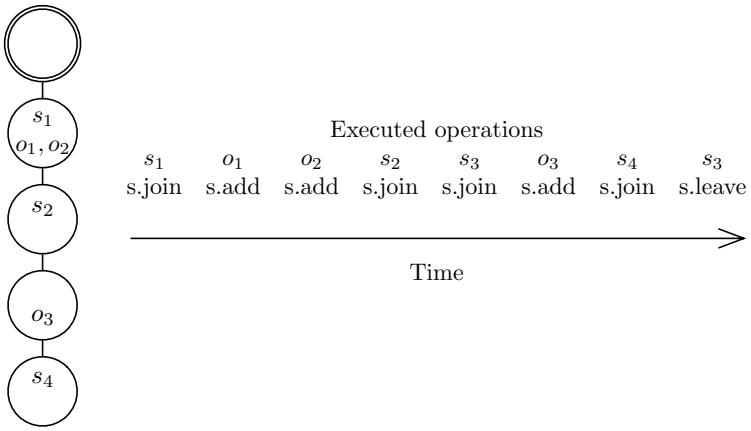


Figure 14: An example role hierarchy implementing top g-SIS in RBAC_1

Furthermore, RBAC_1 can trivially simulate RBAC_0 by ignoring the role hierarchy, yielding the following.

Lemma 7. $\mathsf{RBAC}_0 \leq^{CAH} \mathsf{RBAC}_1$.

Corollary 8. $rgSIS \leq^{CaH} \mathsf{RBAC}_1$.

As we have now shown a full proof of both a reduction (the preceeding proof of Theorem 6) and an implementation (the proof of Theorem 2 in Section 4.5.2), we will state the main ideas behind the remaining reductions and implementations and defer their proofs to the technical report available as [47].

Similarly to our implementation of PC in RBAC_1 (Theorem 2), we are able to construct a reduction from top g-SIS ($tgSIS$) to RBAC_1 by identifying the pseudo-hierarchical structure of the authorization set in $tgSIS$: since all operations are strict, new members of a group will have a *subset* of the permissions of older members. The hierarchy is invoked by the fact that older members thus "inherit" access to all added objects, while new members only receive access to objects added after they joined. We simulate this structure in RBAC_1's role hierarchy by creating a chain in RH for each group. When a g-SIS group g is created, the top of the chain, a role named g, is created in RBAC_1. Objects newly added to the group should be available to all users, and thus the corresponding permission in RBAC_1 is added to

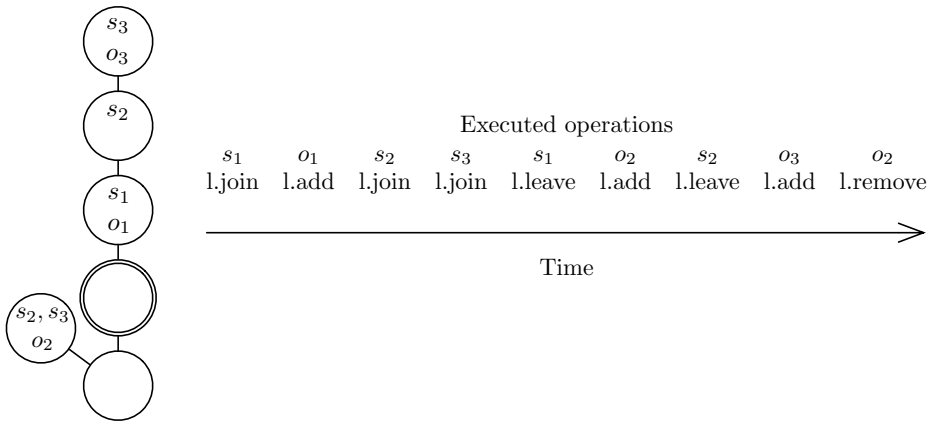


Figure 15: An example role hierarchy implementing bottom g-SIS in RBAC_1

the bottom of the chain, ensuring all users in the chain will be authorized. Finally, when a new user joins group g, they create a new "view" of the g, since they are not authorized to any existing objects (due to strict join). Thus, we create in RBAC_1 a new role (named randomly) and link it to the bottom of g's chain. A demonstrative example of this technique is depicted in Fig. 14. This figure depicts the result of a series of (strict) adds and joins, and finally a leave.

Theorem 9. $tgSIS \leq^{CaH} \mathsf{RBAC}_1$.

We build a reduction from bottom g-SIS ($bgSIS$) to RBAC_1 using a similar hierarchy-chain solution to $tgSIS$ and PC. Since $bgSIS$ contains all liberal actions, we still have a pseudo-hierarchy of $Auth(bgSIS)$. In this case, users who leave a group maintain access to objects from this group, so users who leave earlier have a subset of the authorizations of users who leave later (or are still members). An exception is made for removed objects, since these are not granted to new users after removal. Thus, we again create a hierarchy chain for each group. In this case, the chain grows upward. When a user is removed, a new role is created at the top of the chain and all users remaining in the group are added to this new role. Objects are added to this top role, granting access to all current members. Removed objects, being the exception to the hierarchy rule, are added to orphaned roles, along with all users who should maintain access to them. An example hierarchy chain constructed using this

technique is depicted in Fig. 15, demonstrating how each of the above operations is handled.

Theorem 10. $bgSIS \leq^{CaH} \mathsf{RBAC}_1$.

We utilize a non-homomorphic helper reduction from RBAC_1 to RBAC_0 (and expressiveness transitivity) to prove reductions from $tgSIS$ and $bgSIS$ to RBAC_0. This reduction, in order to store hierarchical accesses in a "flat" roleset, expands the set of roles to include a role named for every path through the hierarchy in the downward direction. Thus, if RBAC_1's hierarchy says $A \geq B$, $B \geq C$, and $A \geq D$, then this is represented in RBAC_0 with roles $\{A, B, C, D, AB, ABC, AD, BC\}$. For every role r a user is assigned to in RBAC_1, she will be assigned to each role starting with r in RBAC_0. In the previous example, if $\langle u, A\rangle \in UR$ in RBAC_1, then in RBAC_0 this maps to $\{\langle u, A\rangle, \langle u, AB\rangle, \langle u, ABC\rangle, \langle u, AD\rangle\} \subset UR$ in RBAC_0.

Theorem 11. $\mathsf{RBAC}_1 \leq^{CA} \mathsf{RBAC}_0$.

Corollary 12. $tgSIS \leq^{Ca} \mathsf{RBAC}_0$; $bgSIS \leq^{Ca} \mathsf{RBAC}_0$.

We were also able to construct a *homomorphic* helper reduction from RBAC_1 to RBAC_0. This reduction makes use of an encoding technique which stores the information in the three binary relations (UR, PA, RH) of RBAC_1 in the two binary relations (UR, PA) of RBAC_0. Using this encoding, each tuple in RBAC_1 is stored using three to four tuples in RBAC_0. Thus, the resulting state encodes the required information in an unnatural, convoluted scheme which requires deeply nested looping to decode. For example, the originally straightforward authorization procedure of finding an r such that $\langle u, r\rangle \in UR$ and $\langle r, p\rangle \in PA$ must be carried out in this reduction by searching for a set of values r, v, x, y, z such that $\{\langle v, x\rangle, \langle u, v\rangle, \langle r, x\rangle\} \subseteq UR$ and $\{\langle y, z\rangle, \langle y, r\rangle, \langle z, p\rangle\} \subseteq PA$. These vast, compounding inefficiencies prevent this reduction from having any practical application. We conjecture that it is impossible to construct an asymptotically more efficient implementation than using this helper reduction while satisfying the given guarantees, which restricts us to storing the required workload state using only tuples in two relations, and using only existing constants (new constants must be information-less). We discuss this reduction further in Appendix A.

Finally, although *ugo* has the inherent disadvantage that each object is owned by only a *single* user and group, we can show $\mathsf{RBAC}_0 \leq^{Ca}$ *ugo* since *ugo* can simulate RBAC_0

implementations by mapping a permission assigned to multiple roles to an object with a single group owner, which represents all roles with authorization and includes as members all users in the RBAC_0 roles. Though the implementation is weakly AC-preserving, it is not homomorphic since it requires the manipulation of strings for group names.

Theorem 13. $\mathsf{RBAC}_0 \leq^{Ca} ugo$.

Corollary 14. $rgSIS \leq^{Ca} ugo$; $tgSIS \leq^{Ca} ugo$; $bgSIS \leq^{Ca} ugo$.

6.4.4 Expressiveness via Implementations

We now present an overview of the implementations of group-centric workloads in dissemination-centric systems. A summary of these implementations and their corresponding strengths is shown in Figure 13b. As in the previous section, we now describe the major results of these implementations.

The PC workload uses liberal join for users joining a program committee group and strict leave for resignation (permanent leave). Liberal leave is used for conflicts-of-interest (temporary leave), and strict join is used to re-join after a COI. We implement this workload in RBAC_1 using techniques from the reductions of both $tgSIS$ and $bgSIS$ in RBAC_1. Like in our reduction from $tgSIS$, each group uses a hierarchy chain building downward, adding a node (and thus a new "view" of the group) each time a user executes a strict join and assigning newly added objects to the bottom role of the chain. Like in the reduction from $bgSIS$, we use the orphan role concept, in this case for users who liberal leave; the departing user and permissions she should continue to be authorized to are added to a new role. We implement u strict leaving g by removing u from all roles connected to g, and u liberal joining g by assigning u directly to g (so she inherits permission to all current objects). This implementation is correct, weakly AC-preserving, homomorphic, and safe (previously stated and proved in Section 4.5.2, depicted in Fig. 4).

Theorem 2 (restated)**.** *There exists a correct, weakly AC-preserving, homomorphic, and safe implementation of PC in* RBAC_1.

The PSP workload supports liberal (restorative) join, strict leave, liberal add, and both strict and liberal remove. Rather than use a role hierarchy chain, this reduction uses a single

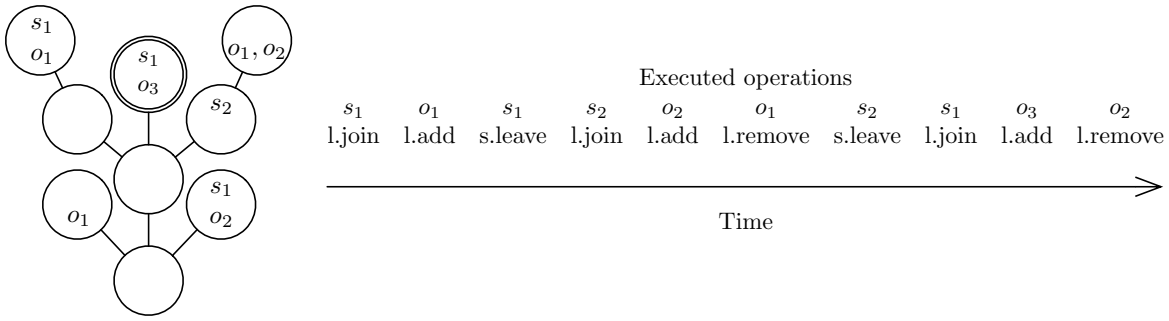


Figure 16: An example role hierarchy implementing the PSP workload in RBAC_1

role for each group that is assigned to all current members and objects in the group. Two types of orphans are used. One type of orphan is in the hierarchy two levels below the group's main role, and is used when an object is liberally removed, to track the users who should remain authorized to the object. The second type of orphan is a *pair* one level below the group's main role, and is used when a user strict leaves, in order to support the restorative join operation. On a strict leave of u from g, we create in RBAC_1 an orphan role pair r, s, where $s \geq r$, and assign u to r and all of u's permissions from g to s. Since u is in a role junior to the permissions, she is no longer authorized to them. On a re-join to the group, we simply assign u to s, re-enabling u's access to these permissions. This is depicted in Fig. 16. This implementation is correct, weakly AC-preserving, homomorphic, and safe.

Theorem 15. *There exists a correct, weakly AC-preserving, homomorphic, and safe implementation of PSP in* RBAC_1.

We use helper reductions from the previous section to establish correct, weakly AC-preserving implementations of PC and PSP in RBAC_0 and *ugo*. We independently prove these implementations are safe, since there is no known meta-theorem for using reductions to prove safety.

Corollary 16. *There exist correct, weakly AC-preserving, and safe implementations of PC & PSP in* RBAC_0 *& ugo.*

6.4.5 Summary of Results

Observing the results of Figures 13a and 13b, it is clear that dissemination-centric systems are sometimes able to meet basic security guarantees when operating within group-centric scenarios. RBAC_1 is the most successful, simulating the extrema systems and workloads with all security guarantees. RBAC_0 was able to implement $rgSIS$ with strong guarantees, but for other g-SIS parameterizations (those with multiple "views" of a single group), RBAC_0 had to sacrifice homomorphism to admit feasible implementations. Finally, *ugo* was also able to satisfy all workloads and systems, but (due to each object being associated with only a single group) did not admit any homomorphic implementations. We note that we also considered the π-system, a g-SIS system defined over the g-SIS$_0$ model with support for all action varieties [71], but were unable to construct a reduction from π-system to (or implementation of $\langle \pi gSIS, \mathcal{T} \rangle$ in) any dissemination-centric system that was AC-preserving.

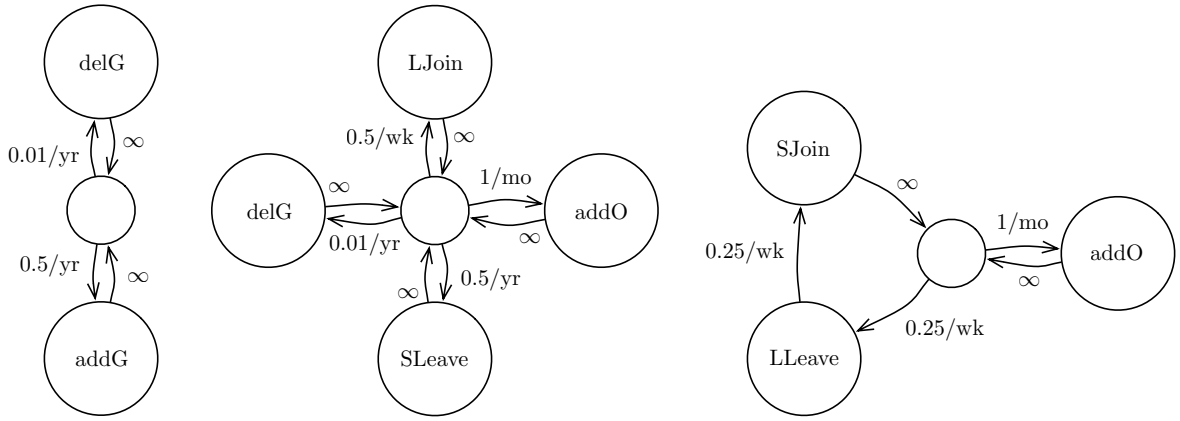
Consider these results in light of the (first half of the) hypothesis stated in Section 6.1 via [72]:

> *It may turn out that at a theoretical level, whatever dissemination-centric [systems] can achieve, group-centric [systems] can also achieve, and vice-versa.*

It is clear that this hypothesis is not entirely true. While there was some success among dissemination-centric systems in implementing *specific* parameterizations of group-centric workloads, these systems do not admit as readily implementations of the *fully expressive* form of g-SIS without the sacrifice of basic security guarantees.

6.5 COST ANALYSIS

Now that we have a clear picture of each dissemination-centric system's expressiveness with respect to group-centric scenarios (which, recall, reflects their theoretical capability), we investigate the second component of these systems' suitability to this set of workloads: efficiency and costs. To consider all of the candidate systems in practical contexts, we evaluate correct, weak AC-preservation implementations, disregarding the homomorphic requirement,



(a) Create (b) Join (c) Discuss

Figure 17: Program Committee actor machines

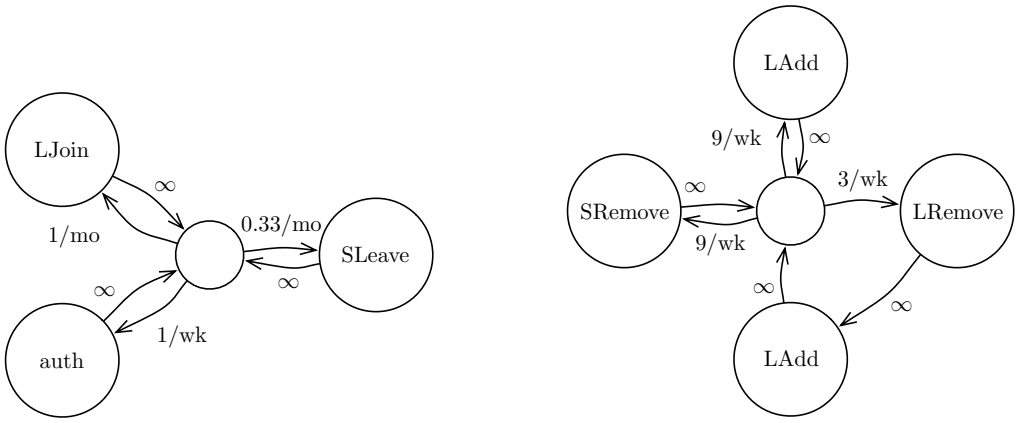
which some systems can not always satisfy feasibly (see Section 6.4.3). We conduct cost analysis via Monte Carlo simulation (using ACCostEvalMC within Portuno—see Chapter 5 for details) driven by the structures built during expressiveness analysis.

We generate initial states and traces to analyze each of our group-centric scenarios as follows.

Program Committee To simulate the PC workload, we select an initial state with 25–75 users. Traces are generated in three phases. First, PC groups are created. Next, PC members join groups. Finally, discussion occurs, and users post objects and execute conflict-of-interest workflows. The actor machines that formalize this behavior are shown in Fig. 17. Traces simulate an eight month cycle, overall.[3]

Playstation Plus Initial states in PSP have 20–100 users and 2–5 regions (subscription groups), with 50–400 objects distributed between them, each representing a current promotion (free game or discount). Traces model users changing membership and administrators adding and removing objects to the regions. The actor machines for this behavior are shown in

[3]While this workload has the same operational component as the case study presented in Sections 4.5 and 5.6, its invocational component is different.



(a) Player (b) Administrators

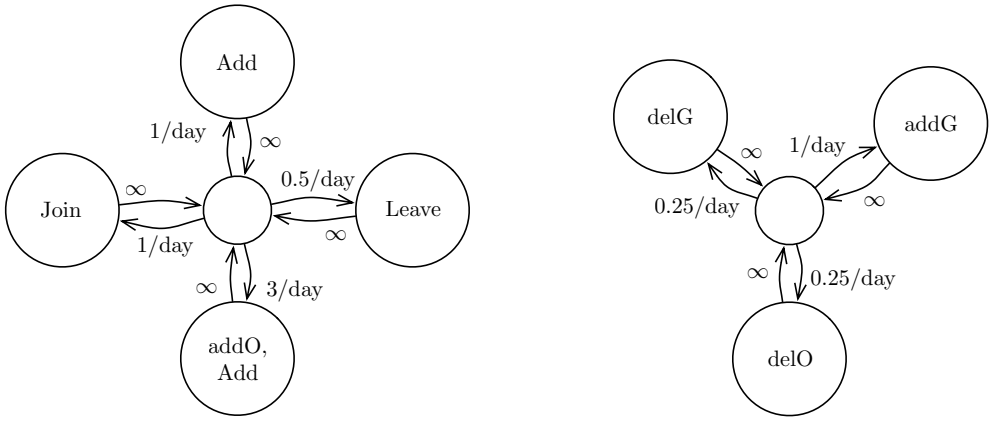
Figure 18: Playstation Plus actor machines

Fig. 18. Each trace models a period of one year.

Extrema Systems To carry out cost analysis of top, bottom, and role-like g-SIS, we must define usage models from scratch, since these systems are not part of workloads. We generate initial states with 25–85 users, and a number of managers between 5 and 1/4 the number of regular users. In traces, managers create groups and sometimes delete posts (e.g., those that violate terms of service). Normal users join groups and share objects, both newly-created and existing (re-shares). The actor machines that formalize this behavior are shown in Fig. 19. Traces model three days to one week of heavy activity, with the average user posting multiple times per day and joining a new group every two days, on average.

6.5.1 Cost Measures

While the type of expressiveness analysis carried out by an analyst is defined by a set of security guarantees that must be upheld, the type of cost analysis is parameterized by the costs to be examined. There are numerous forms of cost measures, from the storage needed to maintain state, to the administrative overhead of executing commands, to the



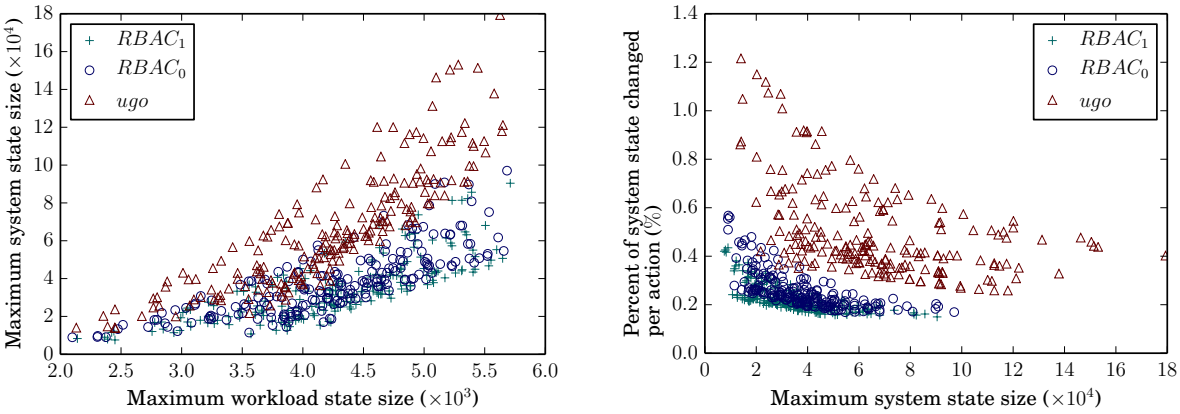
(a) Users (b) Administrators

Figure 19: Extrema system actor machines

computational cost of evaluating queries. In this analysis, we investigate costs representing storage requirements (maximum state size during a run, number of roles); amount of data read/written (average I/O per command, proportion of state changed per command); degree to which atomicity of command execution is violated (number of stutter steps); and other application-specific measures of "misuse" of the implementing systems (average number of permission-assignments per role). To investigate the values of these measures, we plot them against properties of the trace (e.g., number of users, maximum number of objects) and against the workload's own performance within the scenario (e.g., workload I/O, maximum workload state size) for comparison purposes.

6.5.2 Selected Results

We carried out a comprehensive cost analysis of the implementations described in Section 6.4 using the Portuno simulation engine. Our cost analysis uncovered a variety of clear drawbacks to implementing group-centric workloads with dissemination-centric systems; we present several demonstrative examples in Figs. 20–22. Note that each subfigure reports on the result of 200 runs of the workload being simulated.



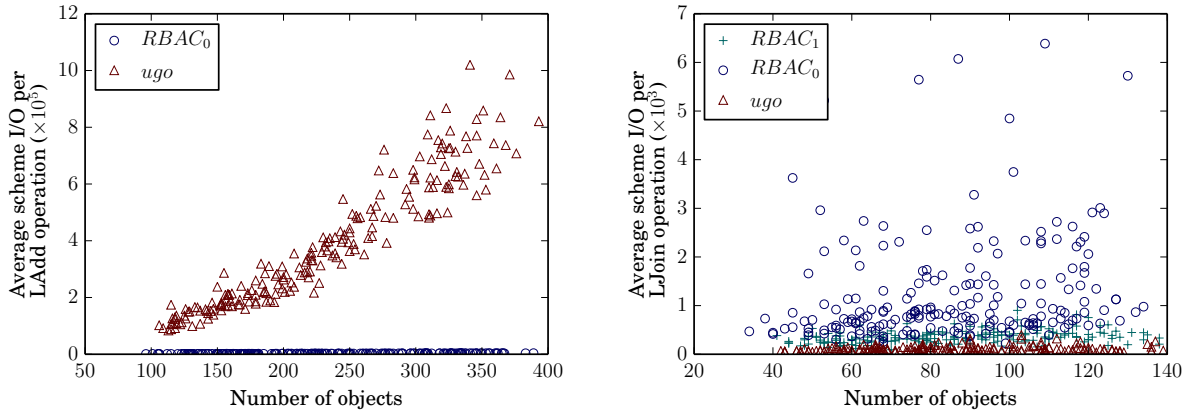
(a) (b)

Figure 20: Group-centric cost analysis results, PlayStation Plus

Storage measures Figure 20a shows that, in the PSP workload, the amount of storage required in each implementing system is superlinear in the size of the workload. This is mostly caused by the blow-up in number of roles/groups required to safely implement the group-centric workloads in systems without built-in temporal abilities. Although PSP is particularly inefficient to implement in dissemination-centric systems, only $rgSIS$ can be implemented using state size comparable to the original workload size even in *ugo*.

Figure 20b shows another aspect of storage, the proportion of the state that is changed on average per simulated (workload) action, again when implementing PSP. This figure shows that our implementation in *ugo* is particularly inefficient. This is largely due to the cached authorization table that must be maintained in *ugo*, making this system a poor choice in scenarios where writes are costly. This pattern is seen across all workloads, and implementations with lower state size generally have the highest proportion of state changing (up to 10% per action), indicating that even those with (relatively) low storage requirements are re-writing large amounts of data to simulate each action.

It is clear that, to support large numbers of users in groups with high object flux, not even hierarchical roles are an efficient replacement for time-aware groups.



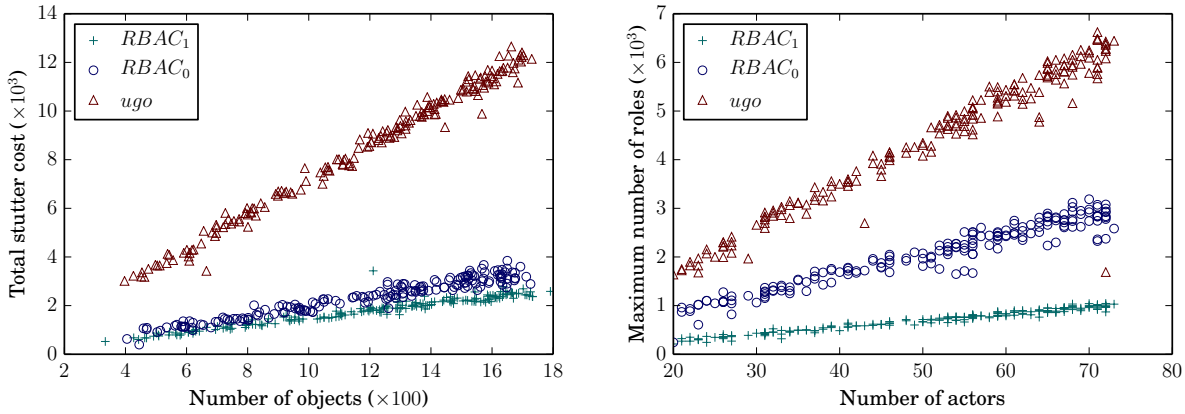
(a) Role-like g-SIS

(b) Bottom g-SIS

Figure 21: Group-centric cost analysis results, $rgSIS$ and $bgSIS$

I/O measures Figure 21a shows the I/O cost (in number of state elements accessed) for simulating liberal add operations in $rgSIS$. Although RBAC_0 is able to simulate role-like g-SIS fairly naturally, *ugo* lacks the ability to natively grant multiple groups access to an object. This missing capability is necessary in group-centric workloads, and thus we must simulate it in *ugo* by assigning each object to a single group and assigning users to these special, semantics-less groups as needed. The extra overhead of iterating over the Member relation to extract the information from RBAC_0's UR and PA and rebuild the cached authorization table is evident in Figure 21a from the superlinear increase in I/O needed to add objects to groups as the number of total objects in the system increases. By comparison to RBAC_0's simple lock-step implementation, *ugo*'s is much more inefficient. The consumer-grade *ugo* system is not practical within even the most simple group-centric workloads.

Figure 21b demonstrates that high I/O is not restricted to implementations in *ugo*. This figure shows I/O for (liberal) joining groups in $bgSIS$. Recall that this is the operation that triggers the role hierarchy chain to expand in RBAC_1's simulation of $bgSIS$, and thus as expected demands high I/O. Specifically, RBAC_1 I/O per join is about 1/4 the total I/O of all commands executed in $bgSIS$ in an average full simulation, and RBAC_0 regularly exceeds the



(a) (b)

Figure 22: Group-centric cost analysis results, Program Committee

workload's full I/O. Although $bgSIS$ in particular has expensive implementations of liberal join, each g-SIS workload (except $rgSIS$ which is efficiently implemented in RBAC_0) has at least one action which causes this characteristically high I/O.

We study the amount of "stuttering" per trace in Figure 22a, which is a description of the number of extra operations that must be executed in implementing systems to simulate single workload actions. We see a drastic increase in stuttering while implementing PC, (especially in *ugo*) as more objects are added to the system. This quantifies the loss of atomicity of operations, and allows us to understand the increasing frequency with which the data structures must be locked to guarantee the desired security properties.

Role abuse Finally, we measure the maximum number of roles created to accommodate the PC workload. It has been said that role-based systems lose their administrative value when the number of roles exceeds the number of users [119]. In Figure 22b, we compare the number of subjects in the PC workload to the number of roles (groups for *ugo*) in the corresponding state of the implementing systems. Roles vastly outnumber users, starting with RBAC_1's role hierarchy chain for each group, and getting worse in both RBAC_0 and *ugo*, each requiring more roles than the previous in order to guarantee weak AC-preservation.

Summary We have shown a number of measures for which dissemination-centric systems RBAC_1, RBAC_0, and *ugo* prove to be very inefficient in implementing group-centric workloads with strong security guarantees. Even the most space-efficient implementations use much more state storage than an equivalent g-SIS parameterization, and require much more I/O to operate. The number of roles created is often many times the number of users and several times the number of objects. With the exception of implementing role-like g-SIS in RBAC, all implementations also cause large amounts of stuttering, or non-atomic sequences of commands to simulate a single workload action. Consider these results through the lens of the second half of the hypothesis stated in Section 6.1 via [72]:

> *[A]t a pragmatic level, we believe these are significantly different approaches to information sharing.*

It seems that in most practical scenarios one must heavily compromise security guarantees or suffer vastly inefficient implementations in order to utilize dissemination-centric systems in the group-centric context. As such, these results confirm this hypothesis.

6.6 DISCUSSION AND FUTURE WORK

6.6.1 Dissemination-centric vs. Group-centric

We set out to evaluate the hypothesis stated by the creators of g-SIS [72]: that the group-centric class of models is equal in collective expressiveness to the dissemination-centric class, but that they are pragmatically different approaches and thus should complement, rather than substitute for, one another. We found in our experiments that these approaches do indeed yield pragmatically dissimilar systems, and that even on a theoretical expressiveness level may not be equivalent.

First, in expressiveness analysis, we displayed the reduction from $rgSIS$ to RBAC_0, the simplest role-based system. It was not surprising that the reduction achieved strong security guarantees ($rgSIS$ was, after all, modeled after role-based systems). However, when we noticed that $tgSIS$ (and, to a lesser extent, $bgSIS$) admitted a hierarchical set of

authorizations within each group, and that this allowed the hierarchical RBAC_1 to implement it just as strongly, we realized that $rgSIS$ is not unique—other parameterizations of g-SIS could be safely implemented using dissemination-centric systems.

This pair of strong implementations shows that in some cases a g-SIS parameterization and a dissemination-centric system can provide the same theoretical capabilities, in part because of structural similarities between how the group- and dissemination-centric counterparts manage internal state. There does not seem to be anything special about these pairs that leads us to believe they are unique. However, in other cases we find the dissemination-centric system unable to fully match the g-SIS system, e.g. RBAC_0 and $tgSIS$. Thus, although we cannot count out the possibility that there is *some* traditional system with the same capabilities as a given g-SIS system, this is not a claim we can confirm based on our investigation of several commonly-used traditional systems and several natural g-SIS parameterizations.

To address the pragmatic differences between dissemination- and group-centric sharing, we carried out cost analyses of the implementations that we developed. Implementations that were bad fits in expressiveness analysis provided continuing evidence of their poor fit in cost analysis. State storage was much higher in these systems than in the ideal systems of the workloads. Executing workload actions often necessitated many stuttering steps in the implementing system, required higher I/O within the implementing system, and changed a high proportion of state for each action. However, these poorly-matched implementations were not alone—even the strongly secure, relatively simple implementation of $tgSIS$ in RBAC_1 had inefficiencies that became evident during cost analysis. Though RBAC_0 could not feasibly satisfy the homomorphic guarantee when implementing $tgSIS$ due to lacking a hierarchy, RBAC_1 required as much of a state space explosion as RBAC_0. Even though it had much lower I/O cost than RBAC_0, RBAC_1 required orders of magnitude greater I/O than in $tgSIS$ to execute its procedure for simulating a strict join.

Thus, we believe we have validated the second point in the hypothesis. Although certain dissemination-centric systems are *able* to implement group-centric workloads, it does not mean they *should*—even when they are theoretically capable, they are not necessarily pragmatically suitable.

6.6.2 In Support of Suitability

Although expressive power analysis has long been the measuring stick for understanding and ranking access control systems in the literature (e.g., [3,11,21,41,59,85,89,101,105,107,114]), the analysis conducted in this chapter supports our central thesis that expressiveness alone does not always tell the whole story. For instance, recall that RBAC_1 was able to implement many interesting g-SIS workloads while maintaining strong security guarantees. However, the complexity required for these implementations to maintain this set of properties resulted in loss of atomicity when executing certain actions, increased state size and state management overheads, and (ultimately) a loss of the elegance of the original workload. From a theoretical perspective, RBAC_1 was expressive enough to encode a variety of group-centric workloads; from a practical perspective, these implementations are less than ideal.

As a large, comprehensive analysis of both expressiveness and cost within the context of a particular access control usage, this case study also firmly supports the feasibility of using the suitability analysis workflow defined in Chapter 3, and the corresponding framework developed in Chapters 4 and 5. Further, the results obtained by this analysis are significant in that they provide a concrete data point indicating the potential dangers of relying too heavily on any one measure of access control suitability when examining the needs of an application. This further supports another central element of our thesis, that suitability analyses must support a wide variety of expressiveness and cost metrics.

6.6.3 Towards an Expressiveness Taxonomy

One goal of parameterized expressiveness [59] is to allow one to choose the notion of expressiveness that best matches the workload in question. Unfortunately, it is not always easy to decide whether to require a particular PE security guarantee, and furthermore PE has not yet enabled the community to break down existing notions of expressiveness into their component properties. For example, it is not known whether there is any combination of parameterized expressiveness properties that yields expressiveness statements equivalent to those made by, e.g., the state matching reduction [114]. We use this observation as motivation for the continued investigation of PE-like techniques, hoping to gain knowledge of both the

properties of and relationships between expressiveness reductions as well as deeper, more fundamental aspects of access control and state machine simulations. Work on this path comes to fruition in Chapter 8.

6.7 SUMMARY

We examined in this chapter the capabilities of popular dissemination-centric access control systems to operate within group-centric workloads. We formalized several group-centric workloads within g-SIS, a family of information sharing models that has been formalized in temporal logic but not yet implemented. We then conducted a large-scale two-phase suitability analysis that we believe to be the first of its kind. We first evaluated whether the dissemination-centric systems are expressive enough to implement the group-centric workloads, assessing the strength of these implementations by examining the security guarantees they preserve. We then conducted a cost analysis, investigating more pragmatic metrics that provide insight into the efficiency of these systems when implementing group-centric workloads.

We found that while RBAC with role hierarchy was able to implement the workloads that we considered with strong security guarantees, a more basic variant of RBAC without role hierarchies could only implement one of our workloads without compromising the guarantees to be upheld. Further, we found that standard UNIX-style user-group-other permissions could not implement any of our group-centric workloads while upholding all required security guarantees. In cost analysis, we found that, with limited exceptions, even those implementations upholding strong security properties suffered from inefficiencies in state size, I/O, and atomicity of operations. These results indicate that g-SIS is a practically significant proposal that elegantly satisfies a class of workloads that existing access control techniques struggle with.

Beyond answering open questions from the literature, this case study supports our thesis by using suitability analysis to prove results that are impossible without it. We also motivate ongoing subjects within suitability, namely the decomposition of expressiveness reductions, which we present in Chapter 8.

7.0 BEYOND POINT STATES: UNDERSTANDING THE COSTS OF DYNAMIC CRYPTOGRAPHIC ACCESS CONTROL IN THE CLOUD

Significant interest has been shown in using (H)IBE, ABE, PE, and FE and related technologies to perform access control for files stored on untrusted cloud providers. Much of this work studies static models, in which the access control policies being enforced do not change over time. However, in most practical cases and in many of the motivations of these works, dynamic access controls are necessary. In this chapter, we explore the application of suitability analysis outside of the realm of classical access control evaluation, and begin to explore the viability and costs of adapting these types of cryptosystems to perform dynamic access control on the cloud under a threat model commonly assumed in the cryptographic literature. To this end, we develop lightweight IBE/IBS- and PKI-based constructions for cryptographically enforcing RBAC_0 access controls over files hosted by a cloud storage provider. In adapting two-phase suitability analysis to this problem, we first prove the correctness of these constructions, and then leverage real-world RBAC datasets and Portuno to experimentally analyze their associated cryptographic costs. Although IBE/IBS and PKI systems are a natural fit for enforcing static RBAC policies, we show that supporting revocation, update, and other state change functionality incurs significant overheads in realistic scenarios. We identify a number of bottlenecks of such systems, and fruitful areas for future work that could lead to more natural and efficient constructions for cryptographic enforcement of dynamic access controls. The majority of our findings also extend to similar attempts to use HIBE, ABE, and PE schemes to enforce dynamic RBAC_1 or ABAC policies.[1]

[1]The material presented in this chapter is currently under review [49].

7.1 INTRODUCTION

With the rise of cloud computing, the economic motivations driving outsourced storage are clear, and major cloud providers such as Google, Microsoft, Apple, and Amazon are providing both industrial large-scale services, and smaller-scale, user-facing services. Similarly, there are a number of user-focused cloud file sharing services, such as Dropbox, Box, and Flickr that use the cloud. However, outsourcing data management raises new questions regarding the maintenance and enforcement of complex—or even simple—access controls that were previously managed in-house. Many times, cloud providers are only partially trusted. This lack of trust is often due to fear of external hacking and data-disclosure (e.g., private photo disclosure [94]), or even state-sponsored attacks against cloud organizations themselves (e.g., Operation Aurora, in which Chinese hackers infiltrated providers like Google, Yahoo, and Rackspace [33, 88]). There has been much discussion about how to achieve access control on the cloud, but in situations where confidentiality or integrity with respect to the cloud provider itself is necessary, cryptographic systems are a natural—if not the only—solution.

With the advent of identity-based encryption (IBE) [16, 97], hierarchical identity-based encryption (HIBE) [15, 50], and forms of attribute-based encryption (ABE) [53] and related signature schemes such as identity-based signatures (IBS) [20], this discussion has further accelerated, as these schemes embed support for many naturally-occurring access control policies. There has also been significant work in the field of cryptography to further expand the expressiveness of these policies. A key question is: *How well-suited are these technologies to solve real-world access control problems?* To the best of our knowledge, there are no cryptographically-enforced access control schemes for the cloud in widespread use. Given the plethora of research in this area, one would expect such systems to be deployed. We believe that this is, in part, due to the fact that much of the writing on using cryptographic schemes as access control mechanisms seems to implicitly assume that the access control policies to be enforced are static (e.g., [12, 53, 84, 97]), or motivate with seemingly dynamic scenarios, but provide static solutions (e.g, [52, 70, 90]).

This static nature is contrary to the majority of traditional access control systems, which support dynamic policies and data. There has been some work in the cryptography

community that provides a level of dynamism for IBE, HIBE, and ABE. This work considers the ability to revoke encryption keys (starting with [14]), with the goal of adding support for dynamic policies and data. When combined with the ability to delegate re-encryption [54, 96], this allows for users' access to be revoked from files. However, this work is not without issues. For example, absent from the discussion of delegated re-encryption (e.g., [54]) is that, in practice, hybrid encryption is used to encrypt files, and that, under reasonable threat models, delegated re-encryption is not compatible with hybrid encryption. Although IBE, HIBE, and ABE primitives seem well-suited for protecting *point states* in many access control paradigms, supporting the *transitions between these states* that are triggered by administrative actions requires addressing very subtle issues involving key management, coordination, and consistency. These issues are not addressed explicitly in the literature, and as the old saying goes, the devil is in the details.

In this chapter, we attempt to use suitability analysis techniques to narrow this gap. We develop two constructions for cryptographically enforcing dynamic role-based access controls (specifically, RBAC_0 [103, 104]) in cloud environments: one based on IBE/IBS techniques, and another based on the standard public-key cryptographic techniques deployed in existing PKIs. We use parameterized expressiveness [59] to prove the correctness of these constructions. To quantify the costs of using these constructions in realistic access control scenarios, we leverage our constrained, actor-based invocation structure and the Portuno simulation environment.

Our simulations, driven by real-world RBAC datasets [34], allow us to explore the costs associated with using these constructions in a variety of environments where the RBAC_0 policy and files in the system are subject to dynamic change. In doing so, we uncover several design considerations that must be addressed, make explicit the complexities of managing transitions that occur as policies or data are modified at runtime (which are often ignored in the literature), and show that the resulting costs of using cryptography to correctly enforce RBAC_0 access controls in a dynamic environment are highly dependent on the mix of administrative operations that occur at runtime. This provides us with a number of insights toward the development of more effective cryptographic access controls. Through our analysis, we make the following contributions:

- Prior work often dismisses the need for an access control reference monitor when using

cryptographically-enforced access controls (e.g., [12,52,53,90]). We discuss the necessity of some minimal reference monitor on the cloud when supporting dynamic, cryptographically-enforced access controls, and we outline other design considerations that must be addressed in dynamic environments.

- We develop constructions that use either the IBE/IBS or public-key cryptographic paradigms to enable dynamic outsourced RBAC_0 access controls, and prove that they correctly implement the RBAC_0 specification. In an effort to lower-bound deployment costs, our constructions sometimes make design choices that emphasize efficiency over the strongest possible security (e.g., using lazy rather than online re-encryption, cf. Section 7.4.3), but our constructions are easily extended to support stronger security guarantees.
- We discuss the deficiencies of using revocation and delegated encryption techniques to update roles/permissions in dynamic RBAC scenarios using hybrid encryption.
- We use real-world RBAC datasets and stochastic models of administrative behavior to quantify the costs of using our constructions. Our findings demonstrate scenarios in which IBE/IBS and public key cryptography are effective means of implementing RBAC access controls, and many situations in which severe overheads are incurred through the use of these techniques. For instance, we show that removing a single user from a role in a moderately-sized organization can require hundreds or thousands of IBE encryptions. Our simulations show that these inflection points in performance are a function of organization size, role density, and administrative operational mix.
- Our experimental findings provide a number of insights into promising future research directions that could lead to better support for cryptographic access controls in dynamic environments. We note that although the use case we explore is based upon RBAC_0, our findings are also relevant to the use of hierarchical RBAC (RBAC_1) or attribute-based access controls (ABAC) implemented using HIBE or ABE techniques, respectively.

The remainder of this chapter is organized as follows. In Section 7.2, we discuss the relevant background in cryptography. Section 7.3 documents our system model and assumptions, and presents the cryptographic primitives used in this chapter. In Section 7.4, we describe our IBE/IBS construction in detail, and overview the key differences between it and our

PKI-based construction. Section 7.5 presents the results of our suitability analysis. In Section 7.6, we identify interesting directions for future work informed by our findings. Section 7.7 summarizes our conclusions.

7.2 BACKGROUND

Starting with the development of practical identity-based encryption (IBE) schemes [16], there has been considerable work put into the development of cryptographic systems that directly support a number of access control functionalities, with examples including hierarchical IBE [50, 61], attribute-based encryption [97], and functional encryption [95]. At a high level, these encryption schemes allow one to encrypt data to a policy, so that only those who have secret keys matching the policy can decrypt. What varies is the expressiveness of the policies that are supported. With IBE and traditional public-key encryption, one can encrypt specifically for a given target individual, and only that individual can decrypt. With attribute-based encryption, a ciphertext can be encrypted to a certain policy, and can be decrypted only by individuals whose secret keys satisfy that policy. With functional encryption, a certain function is embedded in the ciphertext, and when one "decrypts," one does not retrieve the underlying value, but rather a function of the encrypted value and the decryptor's secret key. One underlying motivation in all of the above work is the ability to encrypt data and store it on the cloud while still enforcing access control.

Much of the work on these advanced cryptographic systems allows for data to be stored on the cloud, but it does not address the issue of revocation or dynamic modification of the access control structure being used to store data on the cloud. This can, of course, be done by downloading the data, decrypting it, and then re-encrypting under a new policy, but this is communication intensive, and potentially computationally intensive too. Further, for large files, clients making the changes in the access structure may not be able to support the entire file locally (e.g., smartphones). Therefore, there has been some work done in considering delegated encryption and revocation in these models (e.g., [14, 54, 55, 78, 91, 96, 109]).

There has also been significant work on using cryptography as an access control mechanism,

starting with seminal works such as that by Gudes [56]. This work describes how access controls can be enforced using cryptography, but does not address many practical issues such as key distribution and management, policy updates, and costs. Furthermore, as the motivation is a local file system, the access control system must be trusted with the keys (and trusted to delete them from memory as soon as possible). Work by Akl and Taylor [1] addresses some of the key management issues by proposing a *key assignment scheme*: a system for deriving keys in a hierarchical access control policy, rather than requiring users higher in the hierarchy to store many more keys than those lower in the hierarchy. Again, this work does not consider key distribution or policy updates. Later work in key hierarchies by Atallah et al. [7] proposes a method that allows policy updates, but in the case of revocation, all descendants of the affected node in the access hierarchy must be updated, and the cost of such an operation is not discussed. Continued work in key assignment schemes has improved upon the efficiency of policy updates; see [29] for a survey of such schemes that discusses tradeoffs such as how much private vs. public information must be stored and how much information must be changed for policy updates. Much of this work focuses on the use of symmetric-key cryptography, and so its use for the cloud is potentially limited.

De Capitani di Vimercati et al. [31,32] describe a method for cryptographic access controls on outsourced data using double encryption (one layer by the administrator and one by the service). An extension to this work also enforces write privileges [30]. However, this solution requires a high degree of participation by the cloud provider or third party, and the work does not address the high cost of such operations as deleting users (which can incur cascading updates). Ibraimi's thesis [65] proposes methods for outsourcing data storage using asymmetric encryption. However, the proposed method for supporting revocation requires a trusted mediator and keyshare escrow to verify all reads against a revocation list (and does not address revoked users reusing cached keyshares). Furthermore, policy updates require an active entity to re-encrypt all affected files under the new policy. Similarly, work by Nali et al. [87] enforces role-based access control using public-key cryptography, but requires a series of active security mediators.

Crampton has shown that cryptography is sufficient to enforce RBAC policies [26] and general interval-based access control policies [27], but revocation and policy updates are not

considered (i.e., the constructions are shown only for static policies). Ferrara et al. [38] formally define a cryptographic game for proving the security of cryptographically-enforced RBAC systems and prove that such properties can be satisfied using an ABE-based construction. This construction has since been extended to provide policy privacy and support writes with less trust on the provider [37]. The latter is accomplished by eliminating the reference monitor that checks if a write is allowed and instead accepting each write as a new version; versions must then be verified when downloaded for reading to determine the most recent permitted version (the provider is trusted to provide an accurate version ordering). However, these works do not consider the costs and other practical considerations for using such a system in practice (e.g., lazy vs. active re-encryption, hybrid encryption). In this chapter, we consider exactly these issues in the cryptographic enforcement of access controls.

7.3 THREAT MODELS AND ASSUMPTIONS

Our goal is to understand the practical costs of leveraging public-key cryptographic primitives to implement outsourced dynamic access controls in the cloud. In this section, we define the system and threat models in which we consider this problem, specify the access control model that we propose to enforce, and define the classes of cryptographic primitives that will be used in our constructions.

7.3.1 System and Threat Models

Assumptions regarding trust are central to the specification of any access control approach. Most traditional approaches to access control assume complete mediation of all access requests by a trusted reference monitor, and therefore store resources in plaintext. While this is reasonable when dealing with trusted infrastructure (e.g., a personal machine or corporate server), it becomes less realistic when resources are to be stored on third-party infrastructure. On the other extreme, much of the cryptographic literature on idealized cryptographic access controls tend to assume that the infrastructure housing resources is untrusted and that

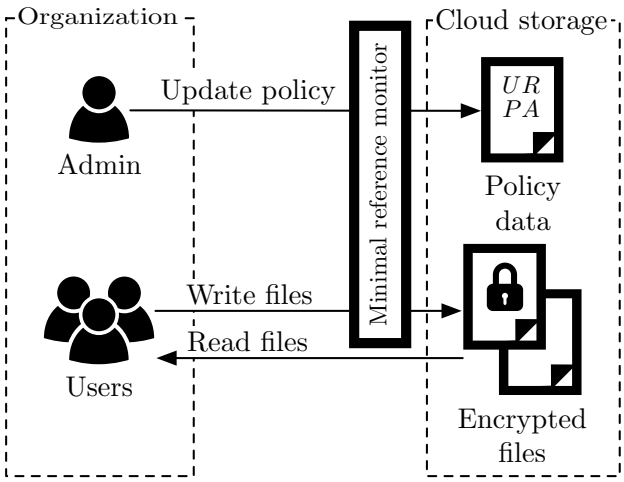


Figure 23: System Diagram

the enforcement of the access control policy is handled entirely cryptographically. These can broken into three categories: (i) those that motivate and provide static access control schemes (e.g., [12, 53, 84]); (ii) those that motivate with dynamic case-studies but provide static solutions(e.g., [52, 70, 90]); and (iii) those that motivate and provide minimal dynamic cryptographic protocols, but do not discuss the security model of reference monitors that would execute the protocols, nor discuss whether the protocols are sufficient to implement a fully featured dynamic access control system (e.g., [96]).

The specific environment that we consider—which is based on the untrusted cloud provider typically assumed in the cryptographic literature—is depicted in Fig. 23. The system consists of three main (classes of) entities: *access control administrators*, *users/clients*, and *cloud storage providers*. Clients and administrators are assumed to belong to the same (perhaps virtual) organization, which outsources some of its data management needs to the cloud. We assume that all parties can communicate via pairwise authenticated and private channels (e.g., SSL/TLS tunnels). *Access control administrators* are tasked with managing the protection state of the storage system. That is, they control the assignment of access permissions, which entails the creation, revocation, and distribution of cryptographic keys used to protect files in a role-based manner. Metadata to facilitate key distribution is stored in a cryptographically

protected manner on the cloud provider. *Users* may download any file stored on the storage provider, but may decrypt, read, and (possibly) modify only the files for which they have been issued the appropriate (role-based) keys. All files are encrypted and signed prior to upload to the cloud storage provider.

The *cloud storage provider* is contracted to manage the storage needs of the users' and administrators' organization. We assume that the cloud is *not* trusted to view the contents of the files that it stores. However, it is trusted to ensure the availability of these files, and to ensure that only authorized individuals update these files. File access is assumed to occur directly though the cloud provider's API, with read access permissions being enforced cryptographically on the client side, and write access permissions being enforced by a minimal reference monitor on the cloud provider that validates client signatures ensuring write privileges prior to file updates. We believe this scenario to encode reasonable assumptions regarding outsourced storage: the storage provider essentially ensures the consistency of this file system by preventing unauthorized updates, yet does not maintain any ability to read or make legitimate modifications to the content of files. In this case study, we explore several cryptographic constructions that meet the above goals, and examine the costs associated with the enforcement of dynamic access controls in this context.

In this chapter, we focus on cryptographic enforcement of RBAC_0 (Example 2), given the prevalence of this type of access control system in both the research literature and commercial systems. Many variants of RBAC exist, but we focus on RBAC_0 as it is conceptually the simplest RBAC variants yet still provides adequate expressive power to be interesting for realistic applications. Generalizing this model to richer RBAC variants (e.g., RBAC_1) and attribute-based access control (ABAC) is discussed in Section 7.6.3.

7.3.2 Cryptographic Primitives

Both of our constructions make use of symmetric-key authenticated encryption ($\mathbf{Gen}^{\mathbf{Sym}}$, $\mathbf{Enc}^{\mathbf{Sym}}$, $\mathbf{Dec}^{\mathbf{Sym}}$). Our PKI scheme uses public-key encryption and digital signatures ($\mathbf{Gen}^{\mathbf{Pub}}$, $\mathbf{Enc}^{\mathbf{Pub}}$, $\mathbf{Dec}^{\mathbf{Pub}}$, $\mathbf{Gen}^{\mathbf{Sig}}$, $\mathbf{Sign}^{\mathbf{Sig}}$, $\mathbf{Ver}^{\mathbf{Sig}}$). While many attribute-based encryption (ABE) schemes are being developed to support policy constructions of varying

expressivity, RBAC_0 does not require this level of sophistication. To this end, we instead use Identity-Based Encryption (IBE) schemes:

- $\mathbf{MSKGen^{IBE}}(1^n)$: Takes security parameter n; generates public parameters (which are implicit parameters to every other IBE algorithm) and master secret key msk.
- $\mathbf{KeyGen^{IBE}}(ID, msk)$: Generates a decryption key k_{ID} for identity ID.
- $\mathbf{Enc}^{\mathbf{IBE}}_{ID}(M)$: Encrypts message M under identity ID.
- $\mathbf{Dec}^{\mathbf{IBE}}_{k_{ID}}(C)$: Decrypts ciphertext C using key k_{ID}; correctness requires that $\forall\ ID$ if $k_{ID} = \mathbf{KeyGen^{IBE}}(ID)$ then $\forall\ M, \mathbf{Dec}^{\mathbf{IBE}}_{k_{ID}}(\mathbf{Enc}^{\mathbf{IBE}}_{ID}(M)) = M$.

  We also use Identity-Based Signature (IBS) schemes:

- $\mathbf{MSKGen^{IBS}}(1^n)$: Takes security parameter n; generates public parameters (which are implicit parameters to every other IBS algorithm) and master secret key msk.
- $\mathbf{KeyGen^{IBS}}(ID, msk)$: Generates a signing key s_{ID} for identity ID.
- $\mathbf{Sign}^{\mathbf{IBS}}_{ID,s_{ID}}(M)$: Generates a signature sig on message M if s_{ID} is a valid signing key for ID.
- $\mathbf{Ver}^{\mathbf{IBS}}_{ID}(M, sig)$: Verifies whether sig is a valid signature on message M for identity ID; requires that $\forall\ ID$
  if $s_{ID} = \mathbf{KeyGen^{IBS}}(ID)$ then
  $\forall\ M \mathbf{Ver}^{\mathbf{IBS}}_{ID}(M, \mathbf{Sign}^{\mathbf{IBS}}_{ID,s_{ID}}(M)) = 1$.

IBE (resp. IBS) schemes build upon traditional public-key schemes by allowing any desired string to act as one's encryption (resp. verification) key. This requires the introduction of a third party who can generate the decryption and signing keys corresponding to these identity strings. This third party, who holds the master keys, is able to produce decryption or signing keys for anyone, and thus the system has inbuilt escrow. In our use of these systems, the RBAC administrator(s) will act as this third party. Since administrators traditionally have the power to access/assign arbitrary permissions, this escrow is not a weakness. In practice, if this is still a concern, threshold/secret splitting schemes can be used to distribute trust amongst several individuals. However, such schemes would increase the cryptographic costs of operations associated with the master key.

7.4 IMPLEMENTATION

While cryptographic access control enforcement has been studied in the past, the focus has been almost entirely on techniques that are best suited for *mostly static* scenarios lacking a trusted reference monitor (e.g., [53, 84]), in which the policies to be enforced and files to be protected change very little over time. As such, the particulars associated with *securely* managing policy change and the associated overheads have been largely under-explored. In this section, we begin with a strawman construction for cryptographic access control enforcement, and use it to highlight a variety of limitations and design considerations that must be addressed. We conclude with a detailed description of our IBE/IBS and PKI constructions, which address these issues.

7.4.1 A Strawman Construction

At first blush, it seems conceptually simple to provision a cryptographically-enforced RBAC_0 system. We now overview such a system, which will allow us to highlight a variety of issues that arise as a result. This strawman construction will make use of IBE/IBS; the use of a more traditional PKI is a straightforward translation. We assume that the administrator holds the master secret keys for the IBE/IBS systems.

- **Registration.** Each user, u, of the system must carry out an initial registration process with the administrator. The result of this process is that the user will obtain identity-based encryption and signing keys $k_u \leftarrow \mathbf{KeyGen}^{\mathbf{IBE}}(u)$ and $s_u \leftarrow \mathbf{KeyGen}^{\mathbf{IBS}}(u)$ from the administrator.
- **Role Administration.** For each role, r, the administrator will generate identity-based encryption and signing keys $k_r \leftarrow \mathbf{KeyGen}^{\mathbf{IBE}}(r)$ and $s_r \leftarrow \mathbf{KeyGen}^{\mathbf{IBS}}(r)$. For each user u that is a member of r (i.e., for each $(u, r) \in UR$ in the RBAC_0 state), the administrator will create and upload a tuple of the form:

$$\langle \mathsf{RK}, u, r, \mathbf{Enc}_u^{\mathbf{IBE}}(k_r, s_r), \mathbf{Sign}_{SU}^{\mathbf{IBS}} \rangle.$$

This tuple provides u with cryptographically-protected access to the encryption and signing keys for r, and is signed by the administrator. Here, $\mathbf{Sign}_{SU}^{\mathbf{IBS}}$ at the end of the tuple represents an IBS signature by identity SU (the administrator), and RK is a sentinel value indicating that this is a role key tuple.

- **File Administration.** For each file f to be shared with a role r (i.e., for each $(r, \langle f, op\rangle) \in PA$ in the RBAC_0 state), the administrator will create and upload a tuple:

$$\langle \mathsf{F}, r, \langle fn, op\rangle, \mathbf{Enc}_r^{\mathbf{IBE}}(f), \mathbf{Sign}_{SU}^{\mathbf{IBS}}\rangle.$$

  This tuple contains a copy of f that is encrypted to members of r. Here, fn represents the name of the file f, while op is the permitted operation—either Read or Write. As before, $\mathbf{Sign}_{SU}^{\mathbf{IBS}}$ is a signature by the administrator, and F is a sentinel value indicating that this is a file tuple.

- **File Access.** If a user u who is authorized to read a file f (i.e., $\exists r : (u, r) \in UR \wedge (r, \langle f, \mathsf{Read}\rangle) \in PA$) wishes to do so, she must (i) download an RK tuple for the role r and an F tuple for f; (ii) validate the signatures on both tuples; (iii) decrypt the role key k_r from the RK tuple using their personal IBE key k_u; and (iv) decrypt the file f from the F tuple using the role key k_r.

  Writes to a file are handled similarly. If u is authorized to write a file f via membership in role r (i.e., $\exists r : (u, r) \in UR \wedge (r, \langle f, \mathsf{Write}\rangle) \in PA$), she can upload a new F tuple $\langle \mathsf{F}, r, \langle fn, \mathsf{Write}\rangle, \mathbf{Enc}_r^{\mathbf{IBE}}(f'), \mathbf{Sign}_r^{\mathbf{IBS}}\rangle$. If the signature authorizing the write ($\mathbf{Sign}_r^{\mathbf{IBS}}$) can be verified by the cloud provider, the existing F tuple for f will be replaced.

This construction describes a cryptographic analog to RBAC_0. The UR relation is encoded in the collection of RK tuples, while the PA relation is encoded in the collection of F tuples. The authorization relation of RBAC_0 is upheld cryptographically: to read a file f, a user u must be able to decrypt a tuple granting her the permissions associated with a role r, which can be used to decrypt a tuple containing a copy of f encrypted to role r.

7.4.2 Design Considerations

While conceptually straightforward, the strawman construction is by no means a complete solution. We now use this construction as a guide to discuss a number of design tradeoffs that must be addressed to support cryptographic enforcement of dynamic RBAC_0 states.

Inefficiency Concerns. The strawman construction exhibits two key issues with respect to efficiency. First, IBE (like public-key cryptography) is not particularly well-suited for the bulk encryption of large amounts of data. As such, the performance of this construction would suffer when large files are shared within the system. Second, this construction requires a duplication of effort when a file, say f, is to be shared with multiple roles, say r_1 and r_2. That is, f must actually be encrypted twice: once with r_1 and once with r_2. We note that this also leads to consistency issues between roles when f is updated. Fortunately, both of these concerns can be mitigated via the use of hybrid cryptography. Rather than storing F tuples of the form:

$$\langle \mathsf{F}, r, \langle fn, op \rangle, \mathbf{Enc}_r^{\mathbf{IBE}}(f), \mathbf{Sign}_{SU}^{\mathbf{IBS}} \rangle$$

We can instead store the following tuples, where $k \leftarrow \mathbf{Gen}^{\mathbf{Sym}}$ is a symmetric key:

$$\langle \mathsf{FK}, r, \langle fn, op \rangle, \mathbf{Enc}_r^{\mathbf{IBE}}(k), \mathbf{Sign}_{SU}^{\mathbf{IBS}} \rangle$$
$$\langle \mathsf{F}, fn, \mathbf{Enc}_k^{\mathbf{Sym}}(f), \mathbf{Sign}_r^{\mathbf{IBS}} \rangle$$

The FK tuples are quite similar to the file encryption tuples in the strawman construction, except that the ciphertext portion of the tuple now includes an IBE-encrypted symmetric key rather than an IBE-encrypted file. The F tuples contain a symmetric-key-encrypted (using an authenticated mode) version of the file f, and are IBS-signed using the role key of the last authorized updater. This adjustment to the metadata improves the efficiency of bulk encryption by making use of symmetric-key cryptography, and greatly reduces the duplication of effort when sharing a file with multiple roles: a single F tuple can be created for the file along with multiple FK tuples (i.e., one per role).

Handling Revocation. The strawman construction can neither revoke a permission from a role, nor remove a user from a role. The former case can be handled by versioning

the F and FK tuples stored within the system, and the latter case handled by adding role versioning to the role key tuples and FK tuples in the system:

$$\langle \mathsf{RK}, u, (r, v_r), \mathbf{Enc}_u^{\mathbf{IBE}}(k_{(r,v_r)}, s_{(r,v_r)}), \mathbf{Sign}_{SU}^{\mathbf{IBS}} \rangle$$
$$\langle \mathsf{FK}, r, \langle fn, op \rangle, v, \mathbf{Enc}_{(r,v_r)}^{\mathbf{IBE}}(k), \mathbf{Sign}_{SU}^{\mathbf{IBS}} \rangle$$
$$\langle \mathsf{F}, fn, v, \mathbf{Enc}_k^{\mathbf{Sym}}(f), \mathbf{Sign}_{(r,v_r)}^{\mathbf{IBS}} \rangle$$

Here, v represents a version number for the symmetric key used to encrypt a file. Role names have been replaced with tuples that include the role name (e.g., r), *as well as a version number (v_r)*. Removing a permission from a role entails re-keying and re-encrypting the file (i.e., creating a new F tuple), and creating new FK tuples for each role whose access to the file have *not* been revoked. The roles increment their previous role number. Similarly, removing a user u from a role r entails deleting u's RK tuple for r, generating new role keys for r (with an incremented version number) and encoding these into new RK tuples for each user remaining in r, and re-versioning all files to which the role r holds some permission. We note that both of these processes must be carried out by an administrator, as only administrators can modify the RBAC_0 state. There is much nuance to these processes, and we defer a full discussion to Section 7.4.3.

Online, Lazy, and Proxy Re-Encryption. Supporting revocation leads to an interesting design choice: should files be re-encrypted immediately upon re-key, or lazily re-encrypted upon their next write? From a confidentiality standpoint, forcing an administrator—or some daemon process running on her behalf—to re-encrypt files immediately upon re-key is preferential, as it ensures that users who have lost the ability to access a file cannot later read its contents. On the other hand, this comes with a potentially severe efficiency penalty in the event that many files are re-keyed due to changes to some role, as access to these files must be locked while they are downloaded, re-encrypted, and uploaded. In our construction, we opt for a *lazy re-encryption* strategy, in which files are re-encrypted by the next user to write to the file (cf., Section 7.4.3). We note that such a scheme is not appropriate for all scenarios, but substantially reduces the computational burden on the cloud when allowing for dynamic updates to the RBAC_0 state (cf., Section 7.5.4). Adapting our construction to instead use online re-encryption is a straightforward extension.

While appealing on the surface, IBE schemes that support proxy re-encryption or revocation (e.g., [14, 54]) are not suitable for use in our scenario. These types of schemes would seemingly allow us to remove our reliance on lazy re-encryption, and have the cloud locally update encryptions when a permission is revoked from a role, or a role from a user. This would be done by creating an updated role name, using proxy re-encryption to move the file from the old role name to the updated one, and then revoking all keys for the old file. The *significant* issue, here, is that such schemes do not address how one would use them with hybrid encryption. We do not believe that a reasonable threat model can assume that even a limited adversary would be unable to cache all the symmetric keys for files she has access to. *Thus, using proxy re-encryption on the* RK *and* FK *tuples and not the* F *tuples would allow users to continue to access files to which their access has been revoked, and so our construction would still require online or lazy re-encryption of the files themselves.*

As a final note, we acknowledge that key-homomorphic PRFs [17] could be combined with revocation and proxy re-encryption schemes, solving the revocation problem completely on the cloud in the hybrid model. However, current technology does not solve the computational effort, as costs of current key-homomorphic PRFs are comparable to or greater than the IBE and PK technologies in consideration.

Multiple Levels of Encryption. We note that our construction has levels of indirection between RK, FK, and F tuples that mirror the indirection between users, roles, and permissions in RBAC_0. This indirection could be flattened to decrease the number of cryptographic operations on the critical path to file access; this would be akin to using an access matrix to encode RBAC_0 states. While this is possible, it has been shown to cause computational inefficiencies when roles' memberships or permissions are altered [48]; in our case this inefficiency would be amplified due to the cryptographic costs associated with these updates.

Other Issues and Considerations. Our constructions are measured without concern for concurrency-related issues that would need to be addressed in practice. We note, however, that features to handle concurrency would be largely independent of the proposed cryptography used to enforce the RBAC_0 policies. As such, we opt for the analysis of the conceptually-simpler schemes presented in this case study. Finally, our analysis is agnostic to the underlying achieved security guarantees and hardness assumptions of the public-key and IBE schemes.

Production implementations would need to consider these issues.

7.4.3 Detailed IBE/IBS Construction

We now flesh out the strawman and previously-discussed enhancements. This produces a full construction for enforcing RBAC_0 protections over an evolving collection managed by a minimally-trusted cloud storage provider.

7.4.3.1 Overview and Preliminaries We reiterate that the administrators act as the Master Secret Key Generator of the IBE/IBS schemes. Users add files to the system by IBE-encrypting these files to the administrators, using hybrid cryptography and F tuples. Administrators assign permissions (i.e., $\langle file, op \rangle$ pairs) to roles by distributing symmetric keys using FK tuples. Role keys are distributed to users using RK tuples. Recall the format of these tuples is as follows:

$$\langle \mathsf{RK}, u, (r, v_r), \mathbf{Enc}_u^{\mathbf{IBE}}(k_{(r,v_r)}, s_{(r,v_r)}), \mathbf{Sign}_{SU}^{\mathbf{IBS}} \rangle$$
$$\langle \mathsf{FK}, r, \langle fn, op \rangle, v, \mathbf{Enc}_{(r,v_r)}^{\mathbf{IBE}}(k), \mathbf{Sign}_{SU}^{\mathbf{IBS}} \rangle$$
$$\langle \mathsf{F}, fn, v, \mathbf{Enc}_k^{\mathbf{Sym}}(f), \mathbf{Sign}_{(r,v_r)}^{\mathbf{IBS}} \rangle$$

Note that symmetric keys and role keys are associated with version information to handle the cases where a user is removed from a role or a permission is revoked from a role.

We assume that files have both read and write permissions associated with them. However, we cannot have write without read, since writing requires decrypting the file's symmetric key, which then can be used to decrypt and read the stored file. Thus we only assign either Read or RW, and only revoke Write (Read is retained) or RW (nothing is retained). When a user wishes to access a file, she determines which of her roles has access to the permission in question. She then decrypts the role's secret key using her identity, and then decrypts the symmetric key for the file using the role's secret key, and finally uses the symmetric key to decrypt the symmetrically-encrypted ciphertext in question.

$addU(u)$
- Generate IBE private key $k_u \leftarrow \mathbf{KeyGen}^{\mathbf{IBE}}(u)$ and IBS private key $s_u \leftarrow \mathbf{KeyGen}^{\mathbf{IBS}}(u)$ for the new user u
- Give k_u and s_u to u over private and authenticated channel

$delU(u)$
- For every role r that u is a member of:
  - $revokeU(u, r)$

$addP_u(fn, f)$
- Generate symmetric key $k \leftarrow \mathbf{Gen}^{\mathbf{Sym}}$
- Send $\langle \mathsf{F}, fn, 1, \mathbf{Enc}_k^{\mathbf{Sym}}(f), \mathbf{Sign}_u^{\mathbf{IBS}} \rangle$ and $\langle \mathsf{FK}, SU, \langle fn, \mathsf{RW} \rangle, 1, \mathbf{Enc}_{SU}^{\mathbf{IBE}}(k), u, \mathbf{Sign}_u^{\mathbf{IBS}} \rangle$ to R.M.
- The R.M. receives $\langle \mathsf{F}, fn, 1, c, sig \rangle$ and $\langle \mathsf{FK}, SU, \langle fn, \mathsf{RW} \rangle, 1, c', u, sig' \rangle$ and verifies that the tuples are well-formed and the signatures are valid, i.e. $\mathbf{Ver}_u^{\mathbf{IBS}}(\langle \mathsf{F}, fn, 1, \mathbf{Enc}_k^{\mathbf{Sym}}(f) \rangle, sig) = 1$ and $\mathbf{Ver}_u^{\mathbf{IBS}}(\langle \mathsf{FK}, SU, \langle fn, \mathsf{RW} \rangle, 1, c', u \rangle, sig') = 1$.
- If verification is successful, the R.M. adds $(fn, 1)$ to FILES and stores $\langle \mathsf{F}, fn, 1, c \rangle$ and $\langle \mathsf{FK}, SU, \langle fn, \mathsf{RW} \rangle, 1, c', u, sig' \rangle$

$delP(fn)$
- Remove (fn, v_{fn}) from FILES
- Delete $\langle \mathsf{F}, fn, -, - \rangle$ and all $\langle \mathsf{FK}, -, \langle fn, - \rangle, -, -, -, - \rangle$

$addR(r)$
- Add $(r, 1)$ to ROLES
- Generate IBE private key $k_{(r,1)} \leftarrow \mathbf{KeyGen}^{\mathbf{IBE}}((r, 1))$ and IBS private key $s_{(r,1)} \leftarrow \mathbf{KeyGen}^{\mathbf{IBS}}((r, 1))$ for role $(r, 1)$
- Send $\langle \mathsf{RK}, SU, (r, 1), \mathbf{Enc}_{SU}^{\mathbf{IBE}}\left(k_{(r,1)}, s_{(r,1)}\right), \mathbf{Sign}_{SU}^{\mathbf{IBS}} \rangle$ to R.M.

$delR(r)$
- Remove (r, v_r) from ROLES
- For all permissions p that r has access to:
  - $revokeP(p, r)$

$assignU(u, r)$
- Find $\langle \mathsf{RK}, SU, (r, v_r), c, sig \rangle$ with $\mathbf{Ver}_{SU}^{\mathbf{IBS}}(\langle \mathsf{RK}, SU, (r, v_r), c \rangle, sig) = 1$
- Decrypt keys $(k_{(r,v_r)}, s_{(r,v_r)}) = \mathbf{Dec}_{k_{SU}}^{\mathbf{IBE}}(c)$
- Send $\langle \mathsf{RK}, u, (r, v_r), \mathbf{Enc}_u^{\mathbf{IBE}}\left(k_{(r,v_r)}, s_{(r,v_r)}\right), \mathbf{Sign}_{SU}^{\mathbf{IBS}} \rangle$ to R.M.

$revokeU(u, r)$
- Delete $\langle \mathsf{RK}, SU, (r, v_r), -, - \rangle$
- Generate new role keys $k_{(r,v_r+1)} \leftarrow \mathbf{KeyGen}^{\mathbf{IBE}}((r, v_r + 1))$, $s_{(r,v_r+1)} \leftarrow \mathbf{KeyGen}^{\mathbf{IBS}}((r, v_r + 1))$
- For all $\langle \mathsf{RK}, u', (r, v_r), c, sig \rangle$ with $u' \neq u$ and $\mathbf{Ver}_{SU}^{\mathbf{IBS}}(\langle \mathsf{RK}, u', (r, v_r), c \rangle, sig) = 1$:
  - Send $\langle \mathsf{RK}, u', (r, v_r + 1), \mathbf{Enc}_{u'}^{\mathbf{IBE}}\left(k_{(r,v_r+1)}, s_{(r,v_r+1)}\right), \mathbf{Sign}_{SU}^{\mathbf{IBS}} \rangle$ to R.M.
- For every p such that there exists $\langle \mathsf{FK}, (r, v_r), p, v_{fn}, c', SU, sig \rangle$ with $\mathbf{Ver}_{SU}^{\mathbf{IBS}}(\langle \mathsf{FK}, (r, v_r), p, v_{fn}, c', SU \rangle, sig) = 1$:
  - For every $\langle \mathsf{FK}, (r, v_r), p, v, c', SU, sig \rangle$ with $\mathbf{Ver}_{SU}^{\mathbf{IBS}}(\langle \mathsf{FK}, (r, v_r), p, v, c', SU \rangle, sig) = 1$:
    - Decrypt key $k = \mathbf{Dec}_{k_{(r,v_r)}}^{\mathbf{IBE}}(c')$
    - Send $\langle \mathsf{FK}, (r, v_r+1), p, v, \mathbf{Enc}_{(r,v_r+1)}^{\mathbf{IBE}}(k), SU, \mathbf{Sign}_{SU}^{\mathbf{IBS}} \rangle$ to R.M.
  - Generate new symmetric key $k' \leftarrow \mathbf{Gen}^{\mathbf{Sym}}$ for p
  - For all $\langle \mathsf{FK}, id, p, v_{fn}, c'', SU, sig \rangle$ with $\mathbf{Ver}_{SU}^{\mathbf{IBS}}(\langle \mathsf{FK}, id, p, v_{fn}, c'', SU \rangle, sig) = 1$:
    - Send $\langle \mathsf{FK}, id, p, v_{fn} + 1, \mathbf{Enc}_{id}^{\mathbf{IBE}}(k'), SU, \mathbf{Sign}_{SU}^{\mathbf{IBS}} \rangle$ to R.M.
  - Increment v_{fn} in FILES, i.e. set $v_{fn} := v_{fn} + 1$
- Delete all $\langle \mathsf{RK}, -, (r, v_r), -, - \rangle$
- Delete all $\langle \mathsf{FK}, (r, v_r), -, -, -, -, - \rangle$
- Increment v_r in ROLES, i.e. set $v_r := v_r + 1$

$assignP(\langle fn, op \rangle, r)$
- For all $\langle \mathsf{FK}, SU, \langle fn, \mathsf{RW} \rangle, v, c, id, sig \rangle$ with $\mathbf{Ver}_{id}^{\mathbf{IBS}}(\langle \mathsf{FK}, SU, \langle fn, \mathsf{RW} \rangle, v, c, id \rangle, sig) = 1$:
  - If this adds Write permission to existing Read permission, i.e. $op = \mathsf{RW}$, $op' = \mathsf{Read}$, and there exists $\langle \mathsf{FK}, (r, v_r), \langle fn, op' \rangle, v, c, SU, sig \rangle$ with $\mathbf{Ver}_{SU}^{\mathbf{IBS}}(\langle \mathsf{FK}, (r, v_r), \langle fn, op' \rangle, v, c, SU \rangle, sig) = 1$:
    - Send $\langle \mathsf{FK}, (r, v_r), \langle fn, \mathsf{RW} \rangle, v, c, SU, \mathbf{Sign}_{SU}^{\mathbf{IBS}} \rangle$ to R.M.
    - Delete $\langle \mathsf{FK}, (r, v_r), \langle fn, \mathsf{Read} \rangle, v, c, SU, sig \rangle$
  - If the role has no existing permission for the file, i.e. there does not exist $\langle \mathsf{FK}, (r, v_r), \langle fn, op' \rangle, v, c, SU, sig \rangle$ with $\mathbf{Ver}_{SU}^{\mathbf{IBS}}(\langle \mathsf{FK}, (r, v_r), \langle fn, op' \rangle, v, c, SU \rangle, sig) = 1$:
    - Decrypt key $k = \mathbf{Dec}_{k_{SU}}^{\mathbf{IBE}}(c)$
    - Send $\langle \mathsf{FK}, (r, v_r), \langle fn, op \rangle, v, \mathbf{Enc}_{(r,v_r)}^{\mathbf{IBE}}(k), SU, \mathbf{Sign}_{SU}^{\mathbf{IBS}} \rangle$ to R.M.

$revokeP(\langle fn, op \rangle, r)$
- If $op = \mathsf{Write}$:
  - For all $\langle \mathsf{FK}, (r, v_r), \langle fn, \mathsf{RW} \rangle, v, c, SU, sig \rangle$ with $\mathbf{Ver}_{SU}^{\mathbf{IBS}}(\langle \mathsf{FK}, (r, v_r), \langle fn, \mathsf{RW} \rangle, v, c, id \rangle, sig) = 1$:
    - Send $\langle \mathsf{FK}, (r, v_r), \langle fn, \mathsf{Read} \rangle, v, c, SU, \mathbf{Sign}_{SU}^{\mathbf{IBS}} \rangle$ to R.M.
    - Delete $\langle \mathsf{FK}, (r, v_r), \langle fn, \mathsf{RW} \rangle, v, c, SU, sig \rangle$
- If $op = \mathsf{RW}$:
  - Delete all $\langle \mathsf{FK}, (r, v_r), \langle fn, - \rangle, -, -, - \rangle$
  - Generate new symmetric key $k' \leftarrow \mathbf{Gen}^{\mathbf{Sym}}$
  - For all $\langle \mathsf{FK}, id, \langle fn, op' \rangle, v_{fn}, c, SU, sig \rangle$ with $\mathbf{Ver}_{SU}^{\mathbf{IBS}}(\langle \mathsf{FK}, id, \langle fn, op' \rangle, v, c, SU \rangle, sig) = 1$:
    - Send $\langle \mathsf{FK}, id, \langle fn, op' \rangle, v_{fn} + 1, \mathbf{Enc}_{id}^{\mathbf{IBE}}(k'), SU, \mathbf{Sign}_{SU}^{\mathbf{IBS}} \rangle$ to R.M.
  - Increment v_{fn} in FILES, i.e. set $v_{fn} := v_{fn} + 1$

$read_u(fn)$
- Find $\langle \mathsf{F}, fn, v, c \rangle$ with valid ciphertext c
- Find a role r such that the following hold:
  - u is in role r, i.e. there exists $\langle \mathsf{RK}, u, (r, v_r), c', sig \rangle$ with $\mathbf{Ver}_{SU}^{\mathbf{IBS}}(\langle \mathsf{RK}, u, (r, v_r), c' \rangle, sig) = 1$
  - r has read access to version v of fn, i.e. there exists $\langle \mathsf{FK}, (r, v_r), \langle fn, op \rangle, v, c'', SU, sig' \rangle$ with $\mathbf{Ver}_{SU}^{\mathbf{IBS}}(\langle \mathsf{FK}, (r, v_r), \langle fn, op \rangle, v, c'', SU \rangle, sig') = 1$
- Decrypt role key $k_{(r,v_r)} = \mathbf{Dec}_{k_u}^{\mathbf{IBE}}(c')$
- Decrypt file key $k = \mathbf{Dec}_{k_{(r,v_r)}}^{\mathbf{IBE}}(c'')$
- Decrypt file $f = \mathbf{Dec}_k^{\mathbf{Sym}}(c)$

$write_u(fn, f)$
- Find a role r such that the following hold:
  - u is in role r, i.e. there exists $\langle \mathsf{RK}, u, (r, v_r), c, sig \rangle$ with $\mathbf{Ver}_{SU}^{\mathbf{IBS}}(\langle \mathsf{RK}, u, (r, v_r), c \rangle, sig) = 1$
  - r has write access to the newest version of fn, i.e. there exists $\langle \mathsf{FK}, (r, v_r), \langle fn, op \rangle, v_{fn}, c', SU, sig' \rangle$ and $\mathbf{Ver}_{SU}^{\mathbf{IBS}}(\langle \mathsf{FK}, (r, v_r), \langle fn, op \rangle, v, c', SU \rangle, sig') = 1$
- Decrypt role key $k_{(r,v_r)} = \mathbf{Dec}_{k_u}^{\mathbf{IBE}}(c)$
- Decrypt file key $k = \mathbf{Dec}_{k_{(r,v_r)}}^{\mathbf{IBE}}(c')$
- Send $\langle \mathsf{F}, fn, v_{fn}, \mathbf{Enc}_k^{\mathbf{Sym}}(f), \mathbf{Sign}_{(r,v_r)}^{\mathbf{IBS}} \rangle$ to R.M.
- The R.M. receives $\langle \mathsf{F}, fn, v, c'', sig'' \rangle$ and verifies the following:
  - The tuple is well-formed with $v = v_{fn}$
  - The signature is valid, i.e. $\mathbf{Ver}_{(r,v_r)}^{\mathbf{IBS}}(\langle \mathsf{F}, fn, v, c'' \rangle, sig'') = 1$
  - r has write access to the newest version of fn, i.e. there exists $\langle \mathsf{FK}, (r, v_r), \langle fn, op \rangle, v_{fn}, c', SU, sig' \rangle$ and $\mathbf{Ver}_{SU}^{\mathbf{IBS}}(\langle \mathsf{FK}, (r, v_r), \langle fn, op \rangle, v_{fn}, c', SU \rangle, sig') = 1$
- If verification is successful, the R.M. replaces $\langle \mathsf{F}, fn, -, - \rangle$ with $\langle \mathsf{F}, fn, v_{fn}, c'' \rangle$

Figure 24: Implementation of RBAC_0 using IBE/IBS

$addU(u)$
- User u generates encryption key pair $(\mathbf{k}_u^{\mathbf{enc}}, \mathbf{k}_u^{\mathbf{dec}}) \leftarrow \mathbf{Gen}^{\mathbf{Pub}}(u)$ and signature key pair $(\mathbf{k}_u^{\mathbf{ver}}, \mathbf{k}_u^{\mathbf{sig}}) \leftarrow \mathbf{Gen}^{\mathbf{Sig}}(u)$.
- Add $(u, \mathbf{k}_u^{\mathbf{enc}}, \mathbf{k}_u^{\mathbf{ver}})$ to USERS

$delU(u)$
- Delete $\mathbf{k}_u^{\mathbf{enc}}$ and $\mathbf{k}_u^{\mathbf{ver}}$
- For every role r that u is a member of:
  - $revokeU(u, r)$

$addP_u(fn, f)$
- Generate symmetric key $k \leftarrow \mathbf{Gen}^{\mathbf{Sym}}$
- Send $\langle \mathsf{F}, fn, 1, \mathbf{Enc}_k^{\mathbf{Sym}}(f), \mathbf{Sign}_{\mathbf{k}_u^{\mathbf{sig}}}^{\mathbf{Sig}} \rangle$ and $\langle \mathsf{FK}, SU, \langle fn, \mathsf{RW} \rangle, 1, \mathbf{Enc}_{SU}^{\mathbf{Pub}}(k), u, \mathbf{Sign}_{\mathbf{k}_u^{\mathbf{sig}}}^{\mathbf{Sig}} \rangle$ to R.M.
- The R.M. receives $\langle \mathsf{F}, fn, 1, c, sig \rangle$ and $\langle \mathsf{FK}, SU, \langle fn, \mathsf{RW} \rangle, 1, c', u, sig' \rangle$ and verifies that the tuples are well-formed and the signatures are valid, i.e. $\mathbf{Ver}_u^{\mathbf{Sig}}(\langle \mathsf{F}, fn, 1, c \rangle, sig) = 1$ and $\mathbf{Ver}_u^{\mathbf{Sig}}(\langle \mathsf{FK}, SU, \langle fn, \mathsf{RW} \rangle, 1, c', u \rangle, sig') = 1$.
- If verification is successful, the R.M. adds $(fn, 1)$ to FILES and stores $\langle \mathsf{F}, fn, 1, c \rangle$ and $\langle \mathsf{FK}, SU, \langle fn, \mathsf{RW} \rangle, 1, c', u, sig' \rangle$

$delP(fn)$
- Remove (fn, v_{fn}) from FILES
- Delete $\langle \mathsf{F}, fn, -, - \rangle$ and all $\langle \mathsf{FK}, -, \langle fn, - \rangle, -, -, -, - \rangle$

$addR(r)$
- Generate encryption key pair $(\mathbf{k}_{(r,1)}^{\mathbf{enc}}, \mathbf{k}_{(r,1)}^{\mathbf{dec}}) \leftarrow \mathbf{Gen}^{\mathbf{Pub}}((r, 1))$ and signature key pair $(\mathbf{k}_{(r,1)}^{\mathbf{ver}}, \mathbf{k}_{(r,1)}^{\mathbf{sig}}) \leftarrow \mathbf{Gen}^{\mathbf{Sig}}((r, 1))$
- Add $(r, 1, \mathbf{k}_{(r,1)}^{\mathbf{enc}}, \mathbf{k}_{(r,1)}^{\mathbf{ver}})$ to ROLES
- Send $\langle \mathsf{RK}, SU, (r, 1), \mathbf{Enc}_{SU}^{\mathbf{Pub}}\left(\mathbf{k}_{(r,1)}^{\mathbf{dec}}, \mathbf{k}_{(r,1)}^{\mathbf{sig}}\right), \mathbf{Sign}_{SU}^{\mathbf{Sig}} \rangle$ to R.M.

$delR(r)$
- Remove $(r, v_r, -, -)$ from ROLES
- Delete all $\langle \mathsf{RK}, -, (r, v_r), -, - \rangle$
- For all permissions $p = \langle fn, op \rangle$ that r has access to:
  - $revokeP(r, \langle fn, \mathsf{RW} \rangle)$

$assignU(u, r)$
- Find $\langle \mathsf{RK}, SU, (r, v_r), c, sig \rangle$ with $\mathbf{Ver}_{SU}^{\mathbf{Sig}}(\langle \mathsf{RK}, SU, (r, v_r), c \rangle, sig) = 1$
- Decrypt keys $(\mathbf{k}_{(r,v_r)}^{\mathbf{dec}}, \mathbf{k}_{(r,v_r)}^{\mathbf{sig}}) = \mathbf{Dec}_{\mathbf{k}_{SU}^{\mathbf{dec}}}^{\mathbf{Pub}}(c)$
- Send $\langle \mathsf{RK}, u, (r, v_r), \mathbf{Enc}_{\mathbf{k}_u^{\mathbf{enc}}}^{\mathbf{Pub}}\left(\mathbf{k}_{(r,v_r)}^{\mathbf{dec}}, \mathbf{k}_{(r,v_r)}^{\mathbf{sig}}\right), \mathbf{Sign}_{SU}^{\mathbf{Sig}} \rangle$ to R.M.

$revokeU(u, r)$
- Generate new role keys $(\mathbf{k}_{(r,v_r+1)}^{\mathbf{enc}}, \mathbf{k}_{(r,v_r+1)}^{\mathbf{dec}}) \leftarrow \mathbf{Gen}^{\mathbf{Pub}}((r, v_r + 1))$, $(\mathbf{k}_{(r,v_r+1)}^{\mathbf{ver}}, \mathbf{k}_{(r,v_r+1)}^{\mathbf{sig}}) \leftarrow \mathbf{Gen}^{\mathbf{Sig}}((r, v_r + 1))$
- For all $\langle \mathsf{RK}, u', (r, v_r), c, sig \rangle$ with $u' \neq u$ and $\mathbf{Ver}_{SU}^{\mathbf{Sig}}(\langle \mathsf{RK}, u', (r, v_r), c \rangle, sig) = 1$:
  - Send $\langle \mathsf{RK}, u', (r, v_r + 1), \mathbf{Enc}_{\mathbf{k}_{u'}^{\mathbf{enc}}}^{\mathbf{Pub}}\left(\mathbf{k}_{(r,v_r+1)}^{\mathbf{dec}}, \mathbf{k}_{(r,v_r+1)}^{\mathbf{sig}}\right), \mathbf{Sign}_{SU}^{\mathbf{Sig}} \rangle$ to R.M.
- For every fn such that there exists $\langle \mathsf{FK}, (r, v_r), \langle fn, op \rangle, v_{fn}, c, SU, sig \rangle$ with $\mathbf{Ver}_{SU}^{\mathbf{Sig}}(\langle \mathsf{FK}, (r, v_r), p, v_{fn}, c, SU \rangle, sig) = 1$:
  - For every $\langle \mathsf{FK}, (r, v_r), \langle fn, op' \rangle, v, c', SU, sig \rangle$ with $\mathbf{Ver}_{\mathbf{k}_{SU}^{\mathbf{ver}}}^{\mathbf{Sig}}(\langle \mathsf{FK}, (r, v_r), \langle fn, op' \rangle, v, c', SU \rangle, sig) = 1$:
    - Decrypt key $k = \mathbf{Dec}_{\mathbf{k}_{(r,v_r)}^{\mathbf{dec}}}^{\mathbf{Pub}}(c')$
    - Send $\langle \mathsf{FK}, (r, v_r + 1), \langle fn, op' \rangle, v, \mathbf{Enc}_{\mathbf{k}_{(r,v_r+1)}^{\mathbf{enc}}}^{\mathbf{Pub}}(k), SU, \mathbf{Sign}_{SU}^{\mathbf{Sig}} \rangle$ to R.M.
  - Generate new symmetric key $k' \leftarrow \mathbf{Gen}^{\mathbf{Sym}}$ for p
  - For all $\langle \mathsf{FK}, id, \langle fn, op'' \rangle, v_{fn}, c'', SU, sig \rangle$ with $\mathbf{Ver}_{\mathbf{k}_{SU}^{\mathbf{ver}}}^{\mathbf{Sig}}(\langle \mathsf{FK}, id, \langle fn, op'' \rangle, v_{fn}, c'', SU \rangle, sig) = 1$:
    - Send $\langle \mathsf{FK}, id, \langle fn, op'' \rangle, v_{fn} + 1, \mathbf{Enc}_{\mathbf{k}_{id}^{\mathbf{enc}}}^{\mathbf{Pub}}(k'), SU, \mathbf{Sign}_{SU}^{\mathbf{Sig}} \rangle$ to R.M.
  - Increment v_{fn} in FILES, i.e. set $v_{fn} := v_{fn} + 1$
- Update r in ROLES, i.e. replace $(r, v_r, \mathbf{k}_{(r,v_r)}^{\mathbf{enc}}, \mathbf{k}_{(r,v_r)}^{\mathbf{ver}})$ with $(r, v_r + 1, \mathbf{k}_{(r,v_r+1)}^{\mathbf{enc}}, \mathbf{k}_{(r,v_r+1)}^{\mathbf{ver}})$
- Delete all $\langle \mathsf{RK}, -, (r, v_r), -, - \rangle$
- Delete all $\langle \mathsf{FK}, (r, v_r), -, -, -, -, - \rangle$

$assignP(r, \langle fn, op \rangle)$
- For all $\langle \mathsf{FK}, SU, \langle fn, \mathsf{RW} \rangle, v, c, id, sig \rangle$ with $\mathbf{Ver}_{id}^{\mathbf{Sig}}(\langle \mathsf{FK}, SU, \langle fn, \mathsf{RW} \rangle, v, c, id \rangle, sig) = 1$:
  - If this adds Write permission to existing Read permission, i.e. $op = \mathsf{RW}$ and there exists $\langle \mathsf{FK}, (r, v_r), \langle fn, \mathsf{Read} \rangle, v, c', SU, sig \rangle$ with $\mathbf{Ver}_{\mathbf{k}_{SU}^{\mathbf{ver}}}^{\mathbf{Sig}}(\langle \mathsf{FK}, (r, v_r), \langle fn, op' \rangle, v, c', SU \rangle, sig) = 1$:
    - Send $\langle \mathsf{FK}, (r, v_r), \langle fn, \mathsf{RW} \rangle, v, c', SU, \mathbf{Sign}_{SU}^{\mathbf{Sig}} \rangle$ to R.M.
    - Delete $\langle \mathsf{FK}, (r, v_r), \langle fn, \mathsf{Read} \rangle, v, c', SU, sig \rangle$
  - If the role has no existing permission for the file, i.e. there does not exist $\langle \mathsf{FK}, (r, v_r), \langle fn, op' \rangle, v, c', SU, sig \rangle$ with $\mathbf{Ver}_{\mathbf{k}_{SU}^{\mathbf{ver}}}^{\mathbf{Sig}}(\langle \mathsf{FK}, (r, v_r), \langle fn, op' \rangle, v, c, SU \rangle, sig) = 1$:
    - Decrypt key $k = \mathbf{Dec}_{\mathbf{k}_{SU}^{\mathbf{dec}}}^{\mathbf{Pub}}(c)$
    - Send $\langle \mathsf{FK}, (r, v_r), \langle fn, op \rangle, v, \mathbf{Enc}_{\mathbf{k}_{(r,v_r)}^{\mathbf{enc}}}^{\mathbf{Pub}}(k), SU, \mathbf{Sign}_{SU}^{\mathbf{Sig}} \rangle$ to R.M.

$revokeP(r, \langle fn, op \rangle)$
- If $op = \mathsf{Write}$:
  - For all $\langle \mathsf{FK}, (r, v_r), \langle fn, \mathsf{RW} \rangle, v, c, SU, sig \rangle$ with $\mathbf{Ver}_{\mathbf{k}_{SU}^{\mathbf{ver}}}^{\mathbf{Sig}}(\langle \mathsf{FK}, (r, v_r), \langle fn, \mathsf{RW} \rangle, v, c, SU \rangle, sig) = 1$:
    - Send $\langle \mathsf{FK}, (r, v_r), \langle fn, \mathsf{Read} \rangle, v, c, SU, \mathbf{Sign}_{SU}^{\mathbf{Sig}} \rangle$ to R.M.
    - Delete $\langle \mathsf{FK}, (r, v_r), \langle fn, \mathsf{RW} \rangle, v, c, SU, sig \rangle$
- If $op = \mathsf{RW}$:
  - Delete all $\langle \mathsf{FK}, (r, v_r), \langle fn, - \rangle, -, -, - \rangle$
  - Generate new symmetric key $k' \leftarrow \mathbf{Gen}^{\mathbf{Sym}}$
  - For all $\langle \mathsf{FK}, id, \langle fn, op' \rangle, v_{fn}, c, SU, sig \rangle$ with $\mathbf{Ver}_{\mathbf{k}_{SU}^{\mathbf{ver}}}^{\mathbf{Sig}}(\langle \mathsf{FK}, id, \langle fn, op' \rangle, v, c, SU \rangle, sig) = 1$:
    - Send $\langle \mathsf{FK}, id, \langle fn, op' \rangle, v_{fn} + 1, \mathbf{Enc}_{id}^{\mathbf{Pub}}(k'), SU, \mathbf{Sign}_{SU}^{\mathbf{Sig}} \rangle$ to R.M.
  - Increment v_{fn} in FILES, i.e. set $v_{fn} := v_{fn} + 1$

$read_u(fn)$
- Find $\langle \mathsf{F}, fn, v, c \rangle$ with valid ciphertext c
- Find a role r such that the following hold:
  - u is in role r, i.e. there exists $\langle \mathsf{RK}, u, (r, v_r), c', sig \rangle$ with $\mathbf{Ver}_{\mathbf{k}_{SU}^{\mathbf{ver}}}^{\mathbf{Sig}}(\langle \mathsf{RK}, u, (r, v_r), c' \rangle, sig) = 1$
  - r has read access to version v of fn, i.e. there exists $\langle \mathsf{FK}, (r, v_r), \langle fn, op \rangle, v, c'', SU, sig' \rangle$ with $\mathbf{Ver}_{\mathbf{k}_{SU}^{\mathbf{ver}}}^{\mathbf{Sig}}(\langle \mathsf{FK}, (r, v_r), \langle fn, op \rangle, v, c'', SU \rangle, sig') = 1$
- Decrypt role key $\mathbf{k}_{(r,v_r)}^{\mathbf{dec}} = \mathbf{Dec}_{\mathbf{k}_u^{\mathbf{dec}}}^{\mathbf{Pub}}(c')$
- Decrypt file key $k = \mathbf{Dec}_{\mathbf{k}_{(r,v_r)}^{\mathbf{dec}}}^{\mathbf{Pub}}(c'')$
- Decrypt file $f = \mathbf{Dec}_k^{\mathbf{Sym}}(c)$

$write_u(fn, f)$
- Find a role r such that the following hold:
  - u is in role r, i.e. there exists $\langle \mathsf{RK}, u, (r, v_r), c, sig \rangle$ with $\mathbf{Ver}_{\mathbf{k}_{SU}^{\mathbf{ver}}}^{\mathbf{Sig}}(\langle \mathsf{RK}, u, (r, v_r), c \rangle, sig) = 1$
  - r has write access to the newest version of fn, i.e. there exists $\langle \mathsf{FK}, (r, v_r), \langle fn, op \rangle, v_{fn}, c', SU, sig' \rangle$ and $\mathbf{Ver}_{\mathbf{k}_{SU}^{\mathbf{ver}}}^{\mathbf{Sig}}(\langle \mathsf{FK}, (r, v_r), \langle fn, op \rangle, v, c', SU \rangle, sig') = 1$
- Decrypt role key $\mathbf{k}_{(r,v_r)}^{\mathbf{dec}} = \mathbf{Dec}_{\mathbf{k}_u^{\mathbf{dec}}}^{\mathbf{Pub}}(c)$
- Decrypt file key $k = \mathbf{Dec}_{\mathbf{k}_{(r,v_r)}^{\mathbf{dec}}}^{\mathbf{Pub}}(c')$
- Send $\langle \mathsf{F}, fn, v_{fn}, \mathbf{Enc}_k^{\mathbf{Sym}}(f), \mathbf{Sign}_{(r,v_r)}^{\mathbf{Sig}} \rangle$ to R.M.
- The R.M. receives $\langle \mathsf{F}, fn, v, c'', sig'' \rangle$ and verifies the following:
  - The tuple is well-formed with $v = v_{fn}$
  - The signature is valid, i.e. $\mathbf{Ver}_{\mathbf{k}_{(r,v_r)}^{\mathbf{ver}}}^{\mathbf{Sig}}(\langle \mathsf{F}, fn, v, c'' \rangle, sig'') = 1$
  - r has write access to the newest version of fn, i.e. there exists $\langle \mathsf{FK}, (r, v_r), \langle fn, op \rangle, v_{fn}, c', SU, sig' \rangle$ and $\mathbf{Ver}_{\mathbf{k}_{SU}^{\mathbf{ver}}}^{\mathbf{Sig}}(\langle \mathsf{FK}, (r, v_r), \langle fn, op \rangle, v_{fn}, c', SU \rangle, sig') = 1$
- If verification is successful, the R.M. replaces $\langle \mathsf{F}, fn, -, - \rangle$ with $\langle \mathsf{F}, fn, v_{fn}, c'' \rangle$

Figure 25: Implementation of RBAC_0 using PKI

7.4.3.2 Full Construction Figure 24 lists every RBAC_0 operation and shows how each can be implemented using IBE, IBS, and the metadata structures described previously. This figure uses the following notation: u is a user, r and q are roles, p is a permission, fn is a file name, f is a file, c is a ciphertext (either IBE or symmetric), sig is an IBS signature, and v is a version number. The identity corresponding to a role r is (r, v), where v is a positive integer representing the version number. We use v_r to denote the latest version number for role r. Roles and versions are stored as (r, v_r) pairs in a file called ROLES, which is publicly viewable and can only be changed by the admin. Similarly, we use v_{fn} to denote the latest version number for the file with name fn. Filenames and versions are stored as (fn, v_{fn}) pairs in a file called FILES, which is publicly viewable and can only be changed by the admin or reference monitor (R.M.). SU is the superuser identity possessed by the administrators. We use "$-$" to represent a wildcard. $\mathbf{Sign}^{\mathbf{IBS}}_{id}$ at the end of a tuple represents an IBS signature by identity id over the rest of the tuple. The subscript after an operation name identifies who performs the operation if it is not performed by an administrator.

Many operations described in Fig. 24 are straightforward given the discussion earlier in this section. To demonstrate some of the more complicated aspects of this construction, we now describe the procedure to revoke a role from a user, which demonstrates several types of re-keys as well as our notion of lazy re-encryption. The procedure for removing a user u from a role r consists of three steps: (i) re-keying r, (ii) re-encrypting existing file keys stored in FK tuples to the new role key, and (iii) re-keying all files accessible by r.

To re-key a role r, we must transition from (r, v_r) to $(r, v_r + 1)$, generating new IBE keys for this new role version. The old RK tuples for r are deleted, and each remaining member u' of role r is given the new RK tuples of the form of $\langle \mathsf{RK}, u', (r, v_r + 1), c, \mathbf{Sign}^{\mathbf{IBS}}_{SU} \rangle$, where c contains the new IBE/IBS keys encrypted to u''s identity key. Next, all (symmetric) file keys encrypted to (r, v_r) in FK tuples are replaced with file keys encrypted to $(r, v_r + 1)$. This allows the remaining members of r to retain access to existing files, while preventing the revoked user u from accessing any file keys that he has not already decrypted and cached.

Finally, each file to which r has access must be re-keyed to prevent u from accessing future updates to this file *using cached symmetric keys*. For each file f, a new symmetric key is generated via $\mathbf{Gen}^{\mathbf{Sym}}$. This key is then encrypted for each role r' that has access to

f (including r), and new FK tuples $\langle \mathsf{FK}, r', \langle f, op \rangle, v+1, c', \mathbf{Sign}^{\mathbf{IBS}}_{SU} \rangle$ are uploaded *alongside* existing $\langle \mathsf{FK}, r', \langle f, op \rangle, v, c, \mathbf{Sign}^{\mathbf{IBS}}_{SU} \rangle$ tuples. Here, $v+1$ is the new file key version, c is the existing encrypted file key, and c' is the new file key IBE-encrypted to identity r'. The next time f is read, the key contained in c will be used for decryption; the next time f is written, the key contained in c' will be used for encryption. This process obviates the need for a daemon to re-encrypt all files at revocation time, but prevents the revoked user u from accessing any future modifications to these files using cached symmetric file keys.

7.4.4 PKI Construction Overview

In our PKI construction presented in Fig. 25, public-key encryption and signatures take the place of IBE and IBS. Each role is assigned a public/private key pair rather than IBE/IBS keys. The primary difference between the IBE and PKI constructions is that IBE/IBS clients are given *escrowed* IBE/IBS identity private keys by the role administrator, while PKI clients generate their own public/private key pairs and upload their public keys. Note that in both systems, the administrators have access to all of the roles' private keys.

7.5 ANALYSIS

In this section, we describe our evaluation of the *suitability* of IBE/IBS and PKI constructions for enforcing RBAC_0 access controls. We utilize a workflow similar to the two-phase suitability analysis framework proposed in Chapter 4, in which we first evaluate the candidates' *expressive power*, then evaluate the *cost* of using each candidate using Monte Carlo simulation based on initial states obtained from real-world datasets.

7.5.1 Qualitative Analysis

As with our previous case studies (Section 4.5 and Chapter 6), we analyze the correctness and security guarantees of our implementations using parameterized expressiveness. In particular, we ensure that the implementation properties of correctness, AC-preservation, and safety are

preserved by these constructions. For formal definitions of these properties, see Section 4.5.2. Using parameterized expressiveness, we prove the following results (full proofs are available in the technical report of [49]).

Theorem 17. *The implementation of* RBAC_0 *using IBE and IBS detailed in Fig. 24 is correct, AC-preserving, and safe.*

Theorem 18. *The implementation of* RBAC_0 *using public key cryptographic techniques detailed in Fig. 25 is correct, AC-preserving, and safe.*

7.5.2 Algebraic Costs

Table 1 lists the costs for each RBAC operation based on the system state. All costs are incurred by the user or administrator running the operation unless otherwise noted. In order to simplify the formulas, we employ a slight abuse of notation: we use the operation itself to represent its cost (e.g., $\mathbf{Enc}^{\mathbf{IBE}}$ is used to represent the cost of one $\mathbf{Enc}^{\mathbf{IBE}}$ operation). We use the following notation:

- $roles(u)$ is the set of roles to which user u is assigned
- $perms(r)$ is the set of permissions to which role r is assigned
- $users(r)$ is the set of users to which role r is assigned
- $roles(p)$ is the set of roles to which permission p is assigned
- $versions(p)$ is the number of versions of permission p

7.5.3 Experimental Setup

To experimentally evaluate the costs of using our IBE construction to enforce RBAC_0, we utilize the Portuno framework proposed in Chapter 5. Within Portuno, we encode RBAC_0 as a workload, with implementations in IBE/IBS and PKI as described in Sections 7.4.3 and 7.4.4. Finally, we add to this framework an adapter to start simulating from start states extracted from real-world RBAC datasets. This adapter allows us to read RBAC datasets in `.ur` and `.pa` format as commonly represented in the role mining community [34], and use these as initial states for our simulation.

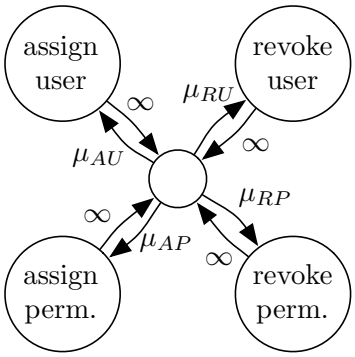
$addU(u)$:	$\mathbf{KeyGen^{IBE}} + \mathbf{KeyGen^{IBS}}$
$delU(u)$:	$\sum_{r \in roles(u)} revokeU(u, r)$
$addP(p)$:	$\mathbf{Enc^{IBE}} + 2 \cdot \mathbf{Sign^{IBS}}$ and $2 \cdot \mathbf{Ver^{IBS}}$ by R.M.
$delP(p)$:	None
$addR(r)$:	$\mathbf{KeyGen^{IBE}} + \mathbf{Enc^{IBE}} + \mathbf{KeyGen^{IBS}} + \mathbf{Sign^{IBS}}$
$delR(r)$:	$\sum_{p \in perms(r)} revokeP(p, r)$
$assignU(u, r)$:	$\mathbf{Enc^{IBE}} + \mathbf{Dec^{IBE}} + \mathbf{Sign^{IBS}} + \mathbf{Ver^{IBS}}$
$revokeU(u, r)$:	$\mathbf{KeyGen^{IBE}} + \mathbf{KeyGen^{IBS}} + \Big(\lvert users(r)\rvert + \sum_{p \in perms(r)} (versions(p) + \lvert roles(p)\rvert)\Big) \cdot (\mathbf{Enc^{IBE}} + \mathbf{Sign^{IBS}} + \mathbf{Ver^{IBS}}) + \Big(\mathbf{Dec^{IBE}} \cdot \sum_{p \in perms(r)} versions(p)\Big)$
$assignP(p, r)$:	$versions(p) \cdot \Big(\mathbf{Sign^{IBS}} + \mathbf{Ver^{IBS}}\Big)$; if r has no permissions for the file then also $versions(p) \cdot \Big(\mathbf{Enc^{IBE}} + \mathbf{Dec^{IBE}}\Big)$
$revokeP(p, r)$:	Revokes all access: $\lvert roles(p)\rvert \cdot \Big(\mathbf{Enc^{IBE}} + \mathbf{Sign^{IBS}} + \mathbf{Ver^{IBS}}\Big)$; Revokes only write access: $\lvert versions(p)\rvert \cdot \Big(\mathbf{Sign^{IBS}} + \mathbf{Ver^{IBS}}\Big)$
$read(fn)$:	$2 \cdot \Big(\mathbf{Dec^{IBE}} + \mathbf{Ver^{IBS}}\Big)$
$write(fn, f)$:	$\mathbf{Sign^{IBS}} + 2 \cdot \Big(\mathbf{Dec^{IBE}} + \mathbf{Ver^{IBS}}\Big)$ and $2 \cdot \mathbf{Ver^{IBS}}$ by R.M.

Table 1: Algebraic costs of RBAC operations using IBE

We simulate one-month periods in which the administrator of the system behaves as described in the actor machine depicted in Fig. 26. The administrative workload increases with the number of users in the system, and we randomly sample an *add bias* parameter that describes the relative proportion of assignment vs. revocation operations. We do not include administrative actions that add or remove users or roles due to the unlikely occurrence of these actions on such short timescales (one-month simulations).

While this administrative behavior model may not be the *only* reasonable one, it describes a range of realistic scenarios and allows us to investigate the interactions in which we are interested. The overall administrative rate is approximately $\sqrt{|U|}$ (with $|U|$ the number of users), ranging from about 0.6 administrative actions per day on our smallest dataset to 2.2 on the largest. Up to 30% of the administrative load is in revocations, since in realistic scenarios permissions tend to be assigned at a greater rate than they are revoked [110].

To quantify the costs associated with our cryptographic constructions, we record the

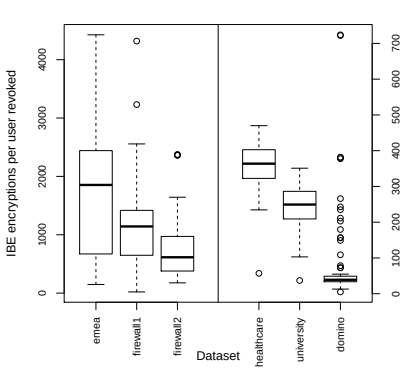


var	semantics	value
R	administrative rate	$0.1 \times \sqrt{\lvert U \rvert}$/day
μ_A	add bias	$[0.7, 1.0]$
μ_U	UR bias	$[0.3, 0.7]$
μ_{AU}	Rate of assignUser	$\mu_A \times \mu_U \times R$
μ_{RU}	Rate of revokeUser	$(1 - \mu_A) \times \mu_U \times R$
μ_{AP}	Rate of assignPermission	$\mu_A \times (1 - \mu_U) \times R$
μ_{RP}	Rate of revokePermission	$(1 - \mu_A) \times (1 - \mu_U) \times R$

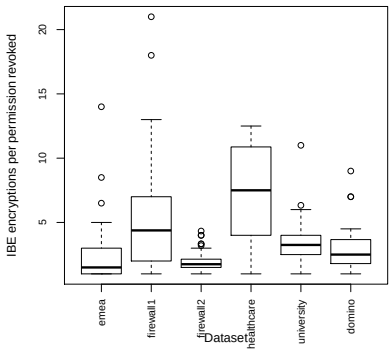
Figure 26: Administrative mix of actions

number of instances of each cryptographic operation executed, including counts or averages for traces of related operations (e.g., the average number of IBE encryptions needed to revoke a role from a user).

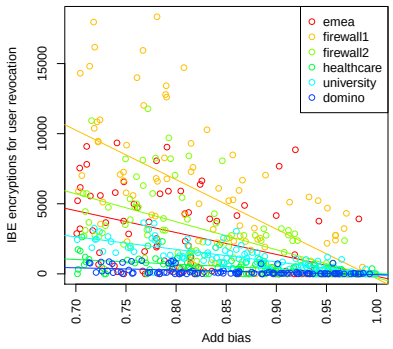
As mentioned above, we initialize our simulation from start states extracted from real-world RBAC datasets. We give a summary of the initial states used in our experiment in Table 2. All of these datasets, aside from university, were originally provided by HP [34]. The domino dataset is from a Lotus Domino server, emea is from a set of Cisco firewalls, firewall1 and firewall2 are generated from network reachability analysis, and healthcare is a list of healthcare permissions from the US Veteran's Administration. The university dataset describes a university's access control system, and was developed by IBM [83, 112].



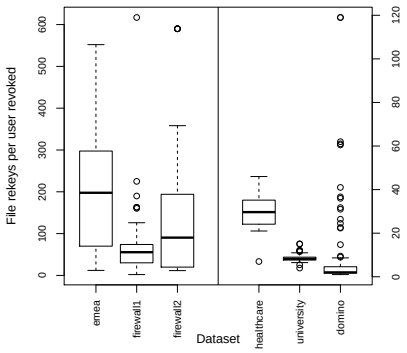
(a) IBE encs. per user revoked



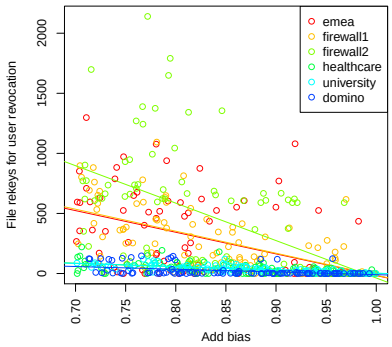
(b) IBE encs. per permission revoked



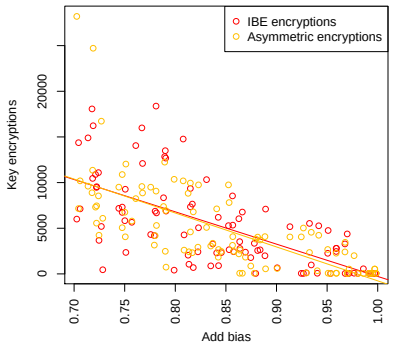
(c) IBE encs. for user revocation vs. add bias



(d) File rekeys per user revoked



(e) File rekeys for user revoc. vs. add bias



(f) Key encryptions vs. add bias (firewall1)

Figure 27: Results

7.5.4 Experimental Results

Figure 27 presents a sampling of our results. First, we consider the cost of performing revocations in our implementation of RBAC_0 using IBE/IBS. Figure 27a shows the average number of IBE encryptions needed for a single user revocation (i.e., removing a user from a role), and Figure 27b shows the same for permission revocation (i.e., revoking a permission from a role). This shows that revoking a permission can cost several IBE encryptions, while user revocation incurs hundreds or thousands of IBE encryptions, on average. We note that, by inspection of the code in Fig. 24, a user revocation also requires an equal number of IBS

						roles/user		users/role		perm./role		roles/perm.	
set	users	|P|	|R|	|UR|	|PA|	max	min	max	min	max	min	max	min
domino	79	231	20	75	629	3	0	30	1	209	1	10	1
emea	35	3046	34	35	7211	1	1	2	1	554	9	31	1
firewall1	365	709	60	1130	3455	14	0	174	1	617	1	25	1
firewall2	325	590	10	325	1136	1	1	222	1	590	6	8	1
healthcare	46	46	13	55	359	5	1	17	1	45	7	12	1
university	493	56	16	495	202	2	1	288	1	40	2	12	1

Table 2: Overview of datasets

signatures and verifications, a smaller number of IBE decryptions, and the generation of new IBE and IBS keys for the role.

For our chosen distribution of administrative actions, Figure 27c shows the total number of IBE encryptions performed over a month for *all* user revocations. As the add bias approaches 1, the number of revocations (and thus the total number of IBE encryptions for user revocation) approaches 0. However, even when only 5–10% of administrative actions are revocation, the number of monthly IBE encryptions under this parameterization is often in the thousands.

In Figure 27d, we show the number of files that must be re-keyed for a single user revocation. This highlights the benefit of utilizing lazy re-encryption; if we had instead utilized *active* re-encryption, each of these files would need to be locked, downloaded, decrypted, re-encrypted, and re-uploaded *immediately* following revocation. In certain scenarios, active re-encryption may not be computationally infeasible. For instance, in university, only ≈ 10 files must be re-encrypted for the average user revocation, adding less than 1% to the total number of file encryptions executed over the entire simulation, even at the highest rate of revocations that we consider. However, in most other scenarios, a user revocation triggers the re-key of tens or hundreds of files, such as in emea or firewall2, where active re-encryption increases the total number of file encryptions by 63% and 12%, respectively (at 20–30% revocation rate). Thus, in most scenarios, active re-encryption is likely to be infeasible, as discussed in Section 7.4.2.

Given the administrative behavior model depicted in Fig. 26, Figure 27e shows the total number of file re-keys that take place over a month for the purpose of user revocation. For

scenarios with very user- and permission-dense roles (e.g., firewall1 and firewall2), we see several times as many re-keys as *total files*, indicating that, on average, each file is re-keyed multiple times per month for the purposes of user revocation. This further enforces that inefficiencies that active re-encryption would bring, as each file (on average) would be locked and re-encrypted by the administrator multiple times per month.

Finally, we note that the costs for our IBE/IBS- and PKI-based constructions for RBAC_0 are not notably different. For instance, Figure 27f compares, for scenario firewall1, the number of IBE encryptions with the number of asymmetric encryptions executed over each simulated month and reveals the same distribution in both IBE/IBS- and PKI-based constructions. Given the similarity in the cost of these classes of operations, we can conclude that these constructions are similarly expensive from a computational standpoint.

7.6 DISCUSSION

There is no doubt that IBE and ABE can enable various forms of cryptographic access control for data in the cloud. In fact, the results presented in Figures 27c, 27e and 27f show that in situations in which the system grows in a monotonic manner (i.e., users and files are *added* to the system and roles are provisioned with *new* permissions), there is no need for revocation, re-keying, or complicated metadata management: IBE alone can enforce RBAC access controls on the cloud. In fact, there are even implications or direct claims in the literature that, in the static setting, the reference monitor can be removed entirely (e.g., [52, 53, 84]). However, this does not imply that IBE or ABE alone can entirely replace the use of a reference monitor when implementing outsourced access controls: *it is not the case when dynamic controls are required.*

Specifically, this chapter shows that IBE and PKI systems are well-suited for implementing *point states* of an RBAC_0 system. However, managing *transitions* between these states—specifically, supporting the removal of a user from a role, the revocation of a permission from a role, and efficient updates to files shared with multiple roles—requires non-trivial metadata management and a small, minimally-trusted reference monitor that verifies signatures prior

to file deletion and replacement. In some of the datasets that we analyzed, this could lead to thousands of IBE encryptions (Figure 27a) and over one hundred file re-keys/re-encryptions (Figure 27d) when a single user is removed from a role.

The above considerations lead to a tradeoff between confidentiality and efficiency that must be weighed by both cryptographers and system designers. There are two obvious ways that this can be accomplished: by altering the threat model assumed, or developing cryptographic approaches that are more amenable to the dynamic setting. We now discuss both of these approaches, and comment on lessons learned during our analysis that can be applied to richer cryptographic access control, such as using HIBE to support RBAC_1, or ABE to support ABAC.

7.6.1 Alternate Threat Models

Many of the overheads that we report on in the previous section result from working within the threat model often implied by the cryptographic literature (i.e., untrusted storage server, minimal client-side infrastructure). Altering this model can reduce the cryptographic costs of enforcing dynamic access controls on the cloud. Here we consider two such alternate models.

Encryption/Decryption Proxy. A large amount of overhead comes from relying the cloud storage provider to act as a (cryptographic) metadata broker, as well as a file store. An alternative approach might make use of an encryption/decryption proxy server situated within an organization, using the cloud provider solely as a backing store for encrypted files. This proxy would act as a traditional reference monitor, mediating all file access requests, downloading and decrypting files for authorized readers[2], and returning plaintext to the user. This would obviate the need for any cryptography beyond authenticated symmetric key encryption, and could make use of tried-and-true access control reference monitors. However, this approach carries an extra infrastructure overhead (the proxy server, itself) that could make it unappealing to *individuals* hoping to enforce access controls over cloud hosted files. Large *organizations* may also have to deal with synchronizing access control policies and key material across multiple proxies in the event that file I/O demands outpace the abilities of a

[2]Writes could be handled symmetrically.

single server.

Trusted Hardware. A more extreme approach to simplifying the cryptographic overheads of access control enforcement would be to use, e.g., an SGX enclave [66, 81] to carry out the work of the encryption/decryption proxy discussed above. In this scenario, files could be stored encrypted on the cloud server, while file encryption keys and the access control policy to be enforced would be managed by a process running within an SGX enclave. To access a file, a user would negotiate an authenticated channel (e.g., using public key cryptography) with this trusted process/reference monitor. The reference monitor could then check the user's permission to access the file, and transmit the encrypted file and its associated key to the user using a session key that is unknown to any process outside of the SGX enclave. This approach frees organizations from the overheads of running their own encryption/decryption proxies, but is not without its limitations. For instance, this approach will not work on commonly-used, storage-only services (e.g., Dropbox). Further, this approach may be subject to architectural compromises or flaws (e.g., memory integrity vulnerabilities) that cryptography-only solutions are not.

While these and other alterations to the threat model that we consider can lead to decreased cryptographic overheads, each incurs other costs or tradeoffs. We now consider future research directions that may decrease the costs associated with cryptography-only solutions to the problem of outsourcing dynamic access controls.

7.6.2 Future Directions

The insight that our suitability analysis framework has enabled us to gain into cryptographically-enforced access controls for cloud storage have led to a number of interesting directions for future work:

Revocation It is unclear how to use IBE to enforce even RBAC_0 without incurring high costs associated with revocation-based state changes. Given our use of hybrid cryptography for efficiency reasons, existing schemes for revocation or proxy re-encryption (e.g., [14, 54]) cannot solve the problem. Developing techniques to better facilitate these forms of revocation and efficient use of hybrid encryption is an important area of future work.

Trust Minimization Our construction makes use of a reference monitor on the cloud to validate signatures prior to file replacement or metadata update. Moving to file versioning (e.g., based on trusted timestamping or block-chaining) rather than file replacement may result in a minimization of the trust placed in this reference monitor, but at the cost of potential confidentiality loss, since old key material may remain accessible to former role members. It is important to better explore this tradeoff between reference monitor trust and confidentiality guarantees.

"Wrapper" Minimization Our construction required the management and use of three types of metadata structures to correctly implement RBAC_0 using IBE or PKI technologies. It would be worth exploring whether the core cryptography used to support outsourced access controls could be enhanced to reduce the use of trusted management code needed to maintain these sorts of structures.

Deployability/Usability Costs We did not consider issues related to the *use* of the cryptographic tools underlying our constructions. Further, our simulations do not separate our IBE- and PKI-based constructions on the basis of RBAC_0 implementation complexity. However, it may be the case that the maturity of tools to support the use of PKIs or the conceptual simplicity of IBE techniques tips the scales in one direction or the other. Developing reasonable approaches for considering these types of tradeoffs would greatly inform future analyses.

While we focused on the use of IBE/IBS and PKI schemes to enforce RBAC_0 access controls, our findings translate in a straightforward manner to the use of other cryptographic tools (e.g., HIBE or ABE/ABS) to implement more complex access control policies (e.g., RBAC_1 or ABAC). We now discuss some lessons learned when considering these richer access control models.

7.6.3 Lessons Learned for More Expressive Systems

RBAC_0 and IBE were natural choices for our initial exploration of the costs associated with using cryptography to implement dynamic access control: RBAC_0 is a simple, but widely used, access control system; roles in RBAC_0 have a natural correspondence to identities in

IBE; and the use of hybrid encryption allows us to easily share resources between roles. Further, it seemed like an implementation of RBAC_0 using IBE would be a jumping off point for exploring the use of hierarchical roles in RBAC_1 via an analogous use of HIBE. However, many of the costs that we see with our IBE implementation of RBAC_0 have analogues (or worse) in any reasonable RBAC_1 or ABAC implementation that we foresee based on respective cryptographic operations.

We first note that we assume that any reasonable cryptographic access control system must make use of hybrid encryption. Without hybrid encryption, we would need to continuously apply expensive asymmetric operations to small "blocks" of a file that is to be encrypted. Given the complexity of IBE/ABE encryption operations, the associated overheads of this approach would be prohibitive, even for moderately-sized files. Additionally, depending on the security requirements of the application (e.g., Chosen Ciphertext Attack security), even more complicated constructions than this simple blocking will be required. The following observations may not apply to an access control scheme where all files are small enough to do away with the need of hybrid-encryption. However, the use cases for such schemes seem limited.

A seemingly natural extension of our IBE based RBAC_0 scheme to a HIBE based RBAC_1 scheme exploits the fact that the HIBE can be used to encode hierarchical relationships, such as those that exist between roles in a RBAC_1 role hierarchy. However, the costs of this implementation proved to be considerable. A large initial problem is that an RBAC_1 role hierarchy can be an arbitrary DAG structure, while HIBE only supports trees. Yet, even limiting RBAC_1 to role hierarchies that form a tree structure comes with serious costs. For example, removing non-leaf roles in the hierarchy cascades re-encryption down to all files at descendant leaves of the role, the creation of new roles for each descendant node, and associated rekeying. Similarly, practical operations like moving sub-trees in the access structure can only be achieved by breaking the operation down into addition and deletion of roles, which comes with the associated costs of these primitive operations. We note that we have developed a full RBAC_1 implementation using HIBE, which attempts to minimize costs. Unfortunately, a simple inspection of this implementation shows that it would incur significantly more computational expense than the RBAC_0 scheme discussed herein.

Similarly, one might hope that the expressiveness of the ABE encryption schemes would allow us to naturally implement ABAC access control schemes. Further, there has been some initial work [96] supporting dynamic (restrictive) credentials and revocations. However, there is still significant work associated with making a practical ABE implementation of ABAC, and such schemes will still have significant costs and meta-data to manage (as in our IBE/RBAC_0 implementation). For example, revoking a secret-key in an KP-ABE/ABAC setting requires the dynamic re-encryption of every ciphertext whose attributes satisfy the policy in the revoked user's key. Each attribute in each ciphertext that is re-encrypted must given a new version, and then finally all users whose keys have policies affected by the re-versioning of the attributes must be re-issued. Further, there are ABAC design decisions that must be informed by the ABE scheme being implemented. For example, suppose a single file is to be accessed by multiple policies in a CP-ABE scheme. One can support multiple policies $p_1, \ldots, p_n$ as individual public-key encryptions all encrypting the same hybrid key, or as a single encryption supporting the disjunction of all previous policies, $p_1 \vee p_2 \vee \cdots \vee p_n$. The cost trade-offs are completely dependent on the ABE scheme used for the implementation, as the cost of ABE encryption is highly dependent on the policy encoded into the ciphertext.

7.7 SUMMARY

IBE and ABE are promising approaches for cryptographically enforcing RBAC and ABAC access controls in the cloud. While prior work has focused the *types* of policies that can be represented by these approaches, little attention has been given to how these policies will *evolve* over time. In this chapter, we move beyond the consideration of point states in an access control system and develop an IBE-based construction that uses hybrid cryptography to enforce RBAC_0 access controls over files hosted on a third-party cloud storage provider. In addition to proving the correctness of our construction, we use real-word RBAC datasets to experimentally analyze its associated cryptographic costs. Our findings indicate that IBE and ABE are a natural fit to this problem in instances where users, roles, and permissions increase monotonically, but incur very high overheads when updates and revocation must

be supported—sometimes exceeding thousands of encryption operations to support a single revocation. In doing so, we believe that we have identified a number of fruitful areas for future work that could lead to more natural constructions for cryptographic enforcement of access control policies in cloud environments.

More pointed toward the goals we set forth in our thesis statement, this case study shows that suitability analysis can be applied to a wider range of scenarios than classic access control. As alluded to in Section 3.4.4, there are many other security problems which share this general structure—potential solutions should first be evaluated with respect to their *capabilities*, to ensure they are able to satisfy the application in question, and then with respect to their *costs*, to determine which is most efficient at satisfying that application. In this chapter, we have shown that our workflow, mathematical framework, and simulation engine Portuno have adapted well to a different problem domain, and in future work we will continue to investigate other case studies that fit this general structure.

8.0 DECOMPOSING, COMPARING, AND SYNTHESIZING ACCESS CONTROL EXPRESSIVENESS SIMULATIONS

The most common technique for analysis of access control, *relative expressiveness analysis*, uses formal mappings called *reductions* to explore whether one access control system is capable of emulating another, thereby comparing the expressive power of these systems. Unfortunately, the notions of expressiveness reduction that have been explored vary widely, which makes it difficult to compare results in the literature, and even leads to apparent contradictions between results. Furthermore, some notions of expressiveness reduction make use of non-determinism, and thus cannot be used to define mappings between access control systems that are useful in practical scenarios. In this chapter, we define the minimum set of properties for an *implementable* access control reduction; i.e., a deterministic "recipe" for using one system in place of another. We then define a wide range of properties spread across several dimensions that can be enforced on top of this minimum definition. These properties define a taxonomy that can be used to separate and compare existing notions of access control reduction, many of which were previously thought to be incomparable. We position existing notions of reduction within our properties lattice by formally proving each reduction's equivalence to a corresponding set of properties. Lastly, we take steps towards bridging the gap between theory and practice by exploring the systems implications of points within our properties lattice. This shows that relative expressive analysis is more than just a theoretical tool, and can also guide the choice of the most suitable access control system for a specific application or scenario.[1]

[1]The material presented in this chapter was first published as [43].

8.1 INTRODUCTION

As noted in Section 2.2, the formal definitions of the various access control expressiveness reductions used in the literature vary widely. Different reductions have been used to prove various types of results, ranging from very specific properties about whole ranges of models (e.g., monotonic access control models with multi-parent creation cannot be emulated by monotonic models with only single-parent creation [3]) to the ability to replace certain specific models with others in practice (e.g., role-based access control can be configured to enforce mandatory and discretionary policies [89]). This disparity in the goals of these works has led to many different definitions of access control reduction, often tailored to the particular result sought. It has been shown that these different reductions prove wildly different notions of expressiveness, often not preserving any particular security properties (see Chapter 2).

Furthermore, not all of these notions of reduction are practically useful. For instance, some make use of non-determinism, manipulating the policy differently depending on which future queries will be asked. While this may allow a theorist to show that system $\mathcal{T}$ is capable of doing all the things $\mathcal{S}$ is, if a practitioner wants to use system $\mathcal{T}$ in place of system $\mathcal{S}$, she needs a deterministic procedure for doing so.

In this work, we build a taxonomy for expressiveness reductions based on the properties that they satisfy. We determine the minimum requirements for a mapping to be *implementable*, or applicable toward using one system in place of another in practice. We use these requirements to construct a general definition of implementable reduction, and provide a taxonomy of additional restrictions on this definition for reductions that enforce more stringent properties. We then position existing reductions from the literature within this lattice, providing the first such comparison in the literature.

To this end, we make the following contributions.

- We propose a general definition of an *implementable* access control mapping that is broad enough to encompass much of the wide range of existing access control reductions, yet precise enough to guarantee implementability. Intuitively, an *implementable* reduction from $\mathcal{S}$ to $\mathcal{T}$ shows that $\mathcal{T}$ can accomplish everything $\mathcal{S}$ can, and deterministically shows *how* (Section 8.3).

- We decompose and expand upon the properties enforced by various access control reductions from the literature, forming a lattice relating the range of access control reductions to one another. This lattice allows us to formally compare the guarantees offered by existing notions of access control reduction (many of which were not formerly known to be comparable) and points to unexplored combinations of properties that can yield different expressiveness results (Section 8.4).
- We construct formal proofs positioning existing notions of access control reduction within our lattice of reduction properties, including a comparative discussion of reductions that previously seemed incomparable. We thus systematize the formal relationships between previously-published reductions, allowing reconciliation of previously disparate expressiveness knowledge (Section 8.5).
- We observe that many of the dimensions upon which our reduction property lattice is built have implications for the use of reductions for satisfying real-world requirements using existing access control systems (e.g., required storage, whether data structures must be locked for concurrent usage). Thus, in addition to positioning existing notions of reduction within our lattice of properties, we assist in creating new notions of reduction by selecting the properties that should be enforced in an expressiveness analysis based upon the scenario in which an eventual access control deployment will occur. To this end, we discuss in detail various interactions between reduction properties, the results of enforcing different properties, and how a specific deployment scenario dictates which properties are relevant (Section 8.6).

  We then summarize our conclusions and future work in Section 8.7.

8.2 MOTIVATING EXAMPLES

An access control system's *expressiveness* (or expressive power) is a measure of the range of policies that it can represent and the transformations it can make to those policies. Statements of *relative* expressiveness state that one system is capable of replacing another (that is, it can represent all the same policies and transform them in equivalent ways). Assume, for instance,

that an organization is considering transitioning from one access control solution to another, in order to accommodate evolving requirements. The organization may have specific desired features for this new access control system, but it certainly must be able to represent all of the policies that the existing system can, or it would not be a suitable replacement. Thus, this organization is searching for a new system that is *at least as expressive as* its old system.

We have made extensive use of expressiveness analysis in this dissertation toward realizing suitability analysis. While we have shown that expressive power alone is insufficient for evaluating an access control system, expressiveness is a fundamentally important component of the more general suitability analysis framework: one cannot determine which access control system is *best* for a particular use case without first determining which are *capable of satisfying* that use case.

Unfortunately, there are several indications that research on expressiveness analysis is being held back by the inability to reconcile the vastly different notions of expressiveness reductions and the disconnect between the properties preserved by a reduction and those that are important to a practical deployment. Several works have demonstrated scenarios in which static notions of expressiveness indicate two systems are equally capable of satisfying a set of operational requirements, but in practice they are better-suited to very different deployment scenarios [48, 86]. Bourdier et al. point out the existence of several competing techniques for expressiveness analysis, none of which consider the deployment. They approach one facet of this problem by proposing a formalism for access control systems that can more easily be transformed into implementations using rewrite-based tools [19]. Several others simply express a desire to use expressiveness analysis, but never do so, presumably due to the complexities of selecting and using the right notion of reduction [73, 92].

A group at the National Institute of Standards and Technology has developed Policy Machine, an attempt at a universal access control system (one that can represent any policy via only configuration changes) [64]. However, in evaluating Policy Machine's success, they avoid formally proving its expressiveness and instead show informal mappings that demonstrate how one might use Policy Machine to represent several existing access control systems' policies [35]. Soon after, the group published a report bemoaning the lack of quality metrics for evaluating access control systems, noting that, in access control, "one size does

not fit all," and thus said metrics must consider the deployment scenario [63].

This overview illustrates that while expressiveness is an important metric in evaluation of access control, the body of knowledge is troublesome to interpret and utilize due to the wide variation in the properties required by each reduction. Through the contributions of this chapter, we fill this void in the literature.

8.3 IMPLEMENTABLE EXPRESSIVENESS REDUCTIONS

In this section, we revisit the *implementability requirements* first presented in Section 4.3 as requirements for an expressiveness mapping to be usable as an implementation of a workload in an access control system. Here, these same properties serve as requirements for a reduction to be *implementable*, and motivate our general formulation of relative expressiveness analysis through the lens of implementability.

8.3.1 Implementability Requirements

In this work, we aim to consider expressiveness reductions that are *implementable*: i.e., practically useful for making decisions about which system is most suitable for a particular deployment. Implementability enforces the following intuition: if a system $\mathcal{T}$ is as expressive as $\mathcal{S}$, then one should be able to determine a general way to use $\mathcal{T}$ in place of $\mathcal{S}$. Thus, we rephrase the minimal set of properties for an expressiveness mapping to be considered implementable.

IR4: State mapping (restated) In order to use $\mathcal{T}$ in place of $\mathcal{S}$, it must be possible to (uniquely) determine which $\mathcal{T}$ state to use in place of a particular $\mathcal{S}$ state. Thus, the state mapping must be a function from the emulated system states to the emulating system states.[2]

IR5: Command mapping (restated) To use $\mathcal{T}$ in place of $\mathcal{S}$, it must be possible to execute commands in $\mathcal{T}$ that are equivalent to the commands in $\mathcal{S}$. It is not necessarily

[2]It is possible that multiple states in $\mathcal{S}$ can be represented using the same state in $\mathcal{T}$. Thus, we do not require the state mapping to be an injection. Furthermore, there may be states in $\mathcal{T}$ that are not used to emulate $\mathcal{S}$, and thus the state mapping need not be an surjection.

the case that each $\mathcal{S}$ command can be emulated using a single $\mathcal{T}$ command, so we require a function from $\mathcal{S}$ commands to *sequences of* $\mathcal{T}$ commands.[3] Finally, it may be necessary to map an $\mathcal{S}$ command differently depending upon the state in which it is intended to be executed. Since using $\mathcal{T}$ in place of $\mathcal{S}$ means we only have a $\mathcal{T}$ state to inspect during execution, this function should map an $\mathcal{S}$ command and a $\mathcal{T}$ state to a sequence of $\mathcal{T}$ commands.

IR6: Query decider (restated) For some reductions of $\mathcal{S}$ in $\mathcal{T}$, we may only care that $\mathcal{T}$ allows the same set of accesses that $\mathcal{S}$ would. However some types of reductions may allow the overriding of $\mathcal{T}$'s default method of deciding granted permissions (e.g., adding the additional requirement that the requesting user is a member of the REAL_USERS group, to distinguish from other data stored in the user-set). While some types of reductions do not allow this, to remain general we simply require a function that maps each $\mathcal{S}$ query and $\mathcal{T}$ state to either true or false. In some formalisms, this only includes the queries requesting access, while in other cases other types of queries are allowed (e.g., "Is user u a member of role r?").

We use these requirements to motivate our definition of the general case of implementable relative expressiveness.

8.3.2 Expressiveness Mappings

Before we define relative expressiveness mappings, we first restate the definitions of the state machines that represent access control systems.

An access control system is formalized as a state machine belonging to a particular access control *model* that formalizes the way in which the access control system will store and interpret information to make access control decisions.

Definition 1 (Access Control Model, restated)**.** An *access control model* is defined as $\mathcal{M} = \langle \Gamma, \mathcal{R} \rangle$, where Γ is the set of states and $\mathcal{R}$ is the set of authorization requests, where each request $r \in \mathcal{R}$ is a function $\Gamma \to \{\text{TRUE}, \text{FALSE}\}$. The entailment ($\vdash$) of a request is

[3] Not *sets* of $\mathcal{T}$ commands, as commands may appear multiple times; and not *bags* of $\mathcal{T}$ commands, as order matters.

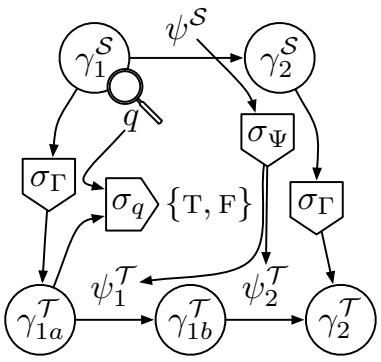


Figure 28: The general form of an implementable expressiveness mapping.

defined as $\gamma \vdash r \triangleq r(\gamma) = \textsc{true}$. ◊

When we refer to the *size* of a state, we are referring to the size of its decomposition into primitive objects (e.g., users and roles) and tuples (e.g., entries in a user assignment relation).

Definition 28. Given an access control model $\mathcal{M} = \langle \Gamma, \mathcal{R} \rangle$ and a state $\gamma \in \Gamma$, the *set decomposition* of γ is denoted $[\gamma]$, and refers to the "set of sets" forming γ, in which γ is represented as being comprised of primitive sets and relations. ◊

Thus, the size of an access control state γ is defined as $|\gamma| = \sum_{S \in [\gamma]} |S|$. For example, if $[\gamma] = \{U = \{u_1\}, R = \{r_1, r_2\}, UR = \{\langle u_1, r_1 \rangle, \langle u_1, r_2 \rangle\}\}$, then $|\gamma| = |U| + |R| + |UR| = 5$.

An access control *system* expands on a model by providing methods of transforming the current state and additional methods of querying the states.

Definition 2 (Access Control System, restated)**.** Given access control model $\mathcal{M} = \langle \Gamma, \mathcal{R} \rangle$, an *access control system* within $\mathcal{M}$ is a state transition system, $\mathcal{S} = \langle \Gamma, \Psi, Q \rangle$, where Ψ is the set of commands, where each command $\psi \in \Psi$ is a function $\Gamma \to \Gamma$, and $Q \supseteq \mathcal{R}$ is the set of queries, where each query $q \in Q$ is a function $\Gamma \to \{\textsc{true}, \textsc{false}\}$. ◊

Next, we define an access control mapping, which maps one system to another but does not enforce any reduction properties. We define a mapping as motivated in Section 8.3.1 so that it can represent any implementable expressiveness reduction.

Definition 29 (Implementable Access Control Mapping)**.** Given two access control systems, $\mathcal{S} = \langle \Gamma^{\mathcal{S}}, \Psi^{\mathcal{S}}, Q^{\mathcal{S}} \rangle$ and $\mathcal{T} = \langle \Gamma^{\mathcal{T}}, \Psi^{\mathcal{T}}, Q^{\mathcal{T}} \rangle$, a *mapping* from $\mathcal{S}$ to $\mathcal{T}$ is a triple of functions $\sigma = \langle \sigma_\Gamma, \sigma_\Psi, \sigma_Q \rangle$, where:

- $\sigma_\Gamma : \Gamma^{\mathcal{S}} \to \Gamma^{\mathcal{T}}$ is the state mapping
- $\sigma_\Psi : \Psi^{\mathcal{S}} \times \Gamma^{\mathcal{T}} \to \left(\Psi^{\mathcal{T}}\right)^*$ is the command mapping
- $\sigma_Q = Q^{\mathcal{S}} \times \Gamma^{\mathcal{T}} \to \{\text{TRUE}, \text{FALSE}\}$ is the query decider

$\Diamond$

This definition is demonstrated in Fig. 28. Each function takes its most general form that satisfies the requirements from Section 8.3.1. Thus, the definition remains general (it does not enforce any specific security requirements yet), while ensuring that any such mappings can generate implementable procedures for using the emulating system in place of the emulated system.

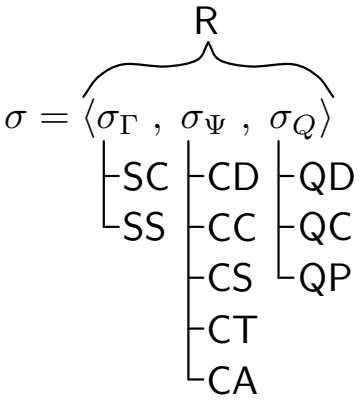
8.4 EXPRESSIVENESS REDUCTION PROPERTIES

In this section, we describe the lattice of properties that we use to taxonomize access control expressiveness reductions.

8.4.1 Overview of dimensions of properties

In order for a mapping to be considered a reduction, it must enforce additional properties over Definition 29. There are naturally three categories of restrictions to consider for reductions, given their structure (a set of three functions): i.e., refinements to each of the state correspondence, command mapping, and query decider. We also consider restrictions to the reachability constraints required (a cross-cutting dimension describing how these functions must relate to one another). A summary of these dimensions is depicted in Fig. 29.

Our state correspondence σ_Γ can be based on any of a handful of structural definitions, defined by SC (i.e., what elements do we inspect to determine whether two states correspond?).



Symbol	Description
SC	State correspondence
SS	State storage
CD	Command mapping dependence
CC	Command mapping complexity
CS	Command mapping stuttering
CT	Trace structure
CA	Actor preservation
QD	Query decider dependence
QC	Query decider complexity
QP	Query preservation
R	Reachability

Figure 29: An overview of the dimensions of expressiveness reduction properties

Further, SS can limit the amount of storage the state correspondence uses (e.g., $\mathcal{T}$ must emulate $\mathcal{S}$ using only a linear amount of additional storage).

The command mapping σ_Ψ can be restricted by CD in what state elements it can use to map commands (e.g., whether it can inspect arbitrary state elements or only those that are exposed via queries). CC considers limiting the time-complexity of the command mapping routine. Since the command mapping returns a *sequence* of commands, CS can limit the number of commands it can return (e.g., only one, or constant in the size of the state). We identify CT, a dimension of concurrency-related trace structure restrictions, as well as CA, requiring the reduction to map $\mathcal{S}$ commands executed by certain types of users only to $\mathcal{T}$ commands executed by that category of users.

The query decider σ_Q can also be restricted in a number of ways. Like the command mapping, we may limit what elements of the state the decider can inspect when deciding how to answer queries within a specific state (QD), or the time-complexity of the routine (QC). In some cases a reduction from $\mathcal{S}$ to $\mathcal{T}$ may be required to map certain $\mathcal{S}$ queries to specific related queries in $\mathcal{T}$, most notably authorization requests (e.g., to answer whether user u should have permission p in $\mathcal{S}$, $\mathcal{T}$ should simply check whether user u has permission p in its current $\mathcal{T}$ state); this type of restriction is handled in QP.

Finally, our reachability restrictions R define how these three functions relate, by allowing us to parameterize whether we require one-way reachability ($\mathcal{T}$ must be able to transition to states corresponding to all reachable $\mathcal{S}$ states) or bidirectional reachability ($\mathcal{T}$ also cannot transition to states that do not correspond to reachable $\mathcal{S}$ states).

The bare minimum set of properties that must be enforced on top of Definition 29 for a mapping to be considered a reduction is (i) a notion of state correspondence, and (ii) a reachability relation. We present the definition of implementable expressiveness reduction, which refines the mapping by enforcing these properties.

Definition 30. Given two access control systems, $\mathcal{S} = \langle \Gamma^{\mathcal{S}}, \Psi^{\mathcal{S}}, Q^{\mathcal{S}} \rangle$ and $\mathcal{T} = \langle \Gamma^{\mathcal{T}}, \Psi^{\mathcal{T}}, Q^{\mathcal{T}} \rangle$ and a mapping $\sigma = \langle \sigma_\Gamma, \sigma_\Psi, \sigma_Q \rangle$ from $\mathcal{S}$ to $\mathcal{T}$, an *implementable expressiveness reduction* of $\mathcal{S}$ in $\mathcal{T}$ based on σ is defined as $\sigma' = \langle \sigma_\Gamma, \sigma_\Psi, \sigma_Q, \sim, R \rangle$, where:

- $\sim \subseteq \Gamma^{\mathcal{S}} \times \Gamma^{\mathcal{T}}$ is a state correspondence relation, and $\forall \gamma \in \Gamma^{\mathcal{S}}, \gamma \sim \sigma_\Gamma(\gamma)$
- R is a reachability restriction

$\Diamond$

We define all properties over the expressiveness reduction $\sigma = \langle \sigma_\Gamma, \sigma_\Psi, \sigma_Q, \sim, R \rangle$. Unless otherwise noted, properties within a dimension are totally ordered from most to least strict.

8.4.2 State correspondence properties

As discussed in Section 8.3.2, the state correspondence of an implementable reduction of $\mathcal{S}$ in $\mathcal{T}$ is a function, $\sigma_\Gamma : \Gamma^{\mathcal{S}} \to \Gamma^{\mathcal{T}}$ mapping each state in $\mathcal{S}$ to a state in $\mathcal{T}$. There are several ways in which we can restrict this mapping.

Dimension (SC: State correspondence structure)**.**

This dimension of properties restricts the way in which corresponding states are structurally similar. All properties within this dimension were inspired by state correspondence relations from prior expressiveness reductions; other application-specific state correspondence relations are conceivable.

Property (SCs: Structure-correspondent)**.** $\gamma^{\mathcal{S}} \overset{\mathsf{s}}{\sim} \gamma^{\mathcal{T}} \triangleq \forall S_i \in \left[\gamma^{\mathcal{S}}\right] . (S_i \in \left[\gamma^{\mathcal{T}}\right])$

Property (SCq: Query-correspondent). $\gamma^{\mathcal{S}} \overset{\mathsf{q}}{\sim} \gamma^{\mathcal{T}} \triangleq \forall q \in Q^{\mathcal{S}}.(\gamma^{\mathcal{S}} \vdash q \iff \sigma_Q(q, \gamma^{\mathcal{T}}) = \text{TRUE})$

Property (SCa: Authorization-correspondent). $\gamma^{\mathcal{S}} \overset{\mathsf{a}}{\sim} \gamma^{\mathcal{T}} \triangleq \forall r \in \mathcal{R}^{\mathcal{S}}.(\gamma^{\mathcal{S}} \vdash r \iff \sigma_Q(r, \gamma^{\mathcal{T}}) = \text{TRUE})$

Authorization-correspondent reductions enforce that every $\gamma^{\mathcal{S}}$ maps to a $\gamma^{\mathcal{T}}$ that agrees on all authorization requests: any permission granted/denied in $\gamma^{\mathcal{S}}$ must also be granted/denied in $\gamma^{\mathcal{T}}$. Requests that exist in $\mathcal{T}$ but not in $\mathcal{S}$ are not restricted. This type of correspondence is used in [21, 85, 89, 101, 105, 107]. *Query-correspondence* requires that $\gamma^{\mathcal{S}}$ and $\gamma^{\mathcal{T}}$ agree on *all* queries, not just authorization requests. This type of correspondence is used in the expressiveness reductions of [59, 114].

Finally, *structure-correspondent* reductions require all corresponding state elements to be identical. If $\gamma^{\mathcal{S}}$ structure-corresponds to $\gamma^{\mathcal{T}}$, then every set in $\gamma^{\mathcal{S}}$ exists in $\gamma^{\mathcal{T}}$, and contains all the same elements ($\gamma^{\mathcal{T}}$ may contain additional sets or relations). Thus, if $\gamma^{\mathcal{S}}$ contains sets of users and permissions, and a relation between them (a subset of users $\times$ permissions) specifying accesses, $\gamma^{\mathcal{T}}$ must contain identical sets of users and permissions, and an identical set of $\langle \text{user}, \text{permission} \rangle$ pairs. This notion of state correspondence is used in [3].

The type of state correspondence used is a central characteristic of a type of reduction. Enforcing a state correspondence that is too weak can allow the emulating system to diverge from the emulated system in unexpected ways, while a state correspondence that is too strong will cause the emulating system to track the emulated system more closely than necessary (e.g., by constraining the values of queries that the deployment never needs to ask). Thus, choosing a particular state correspondence is choosing how closely the emulating system must stay to the emulated system.

Dimension (SS: State storage).

An orthogonal class of restrictions that can be placed on the state correspondence relation involve its allowed storage. Here, we restrict the size of $\gamma^{\mathcal{T}} = \sigma_\Gamma(\gamma^{\mathcal{S}})$ with respect to $\gamma^{\mathcal{S}}$.

Property (SSl: Linear storage). $\exists c \in \mathbb{R}^+, s \in \mathbb{Z}^+ : \forall \gamma \in \Gamma^{\mathcal{S}} : |\gamma| \geq s \Rightarrow |\sigma_\Gamma(\gamma)| \leq c|\gamma|$

Property (SSp: Polynomial storage). $\exists k \in \mathbb{R}^+, s \in \mathbb{Z}^+ : \forall \gamma \in \Gamma^{\mathcal{S}} : |\gamma| \geq s \Rightarrow |\sigma_\Gamma(\gamma)| \leq |\gamma|^k$

Property (SS∞: Unbounded storage)**.** No restriction.

A *linear storage* reduction says that $\gamma^{\mathcal{T}}$ can grow at most linearly with $\gamma^{\mathcal{S}}$, while in a *polynomial storage* reduction, the size of $\gamma^{\mathcal{T}}$ is bounded by a polynomial in the size of $\gamma^{\mathcal{S}}$. The most obvious result of enforcing properties within SS is limited trusted storage, but it can also limit iteration over the resulting state (e.g., if an action must be taken for each document in the emulating system, SSI ensures that this sequence of actions is linear in the size of the emulated state).

8.4.3 Command mapping properties

Recall that the command mapping for an implementable reduction (Definition 30) is a function $\sigma_\Psi : \Psi^{\mathcal{S}} \times \Gamma^{\mathcal{T}} \to \left(\Psi^{\mathcal{T}}\right)^*$ that returns the sequence of $\mathcal{T}$ commands needed to emulate an $\mathcal{S}$ command starting from a particular $\mathcal{T}$ state. Thus, it allows us to emulate $\mathcal{S}$ commands in an active emulation using $\mathcal{T}$. We now discuss the ways in which we can restrict this mapping.

Dimension (CD: Command mapping dependence)**.**

While Definition 30 maps each $\mathcal{S}$ command and $\mathcal{T}$ state to a sequence of $\mathcal{T}$ commands, some previous works use more strict command mappings, mapping each $\mathcal{S}$ command to a sequence of $\mathcal{T}$ commands without considering the state [21]. In between these options, we may map each $\mathcal{S}$ command and $\mathcal{T}$ *theory*, calculating the sequence of $\mathcal{T}$ commands by observing only the queriable portions of the $\mathcal{T}$ state. *Command mapping dependence* thus restricts the information that the command mapping can consider about a $\mathcal{T}$ state when calculating the trace of $\mathcal{T}$ commands to execute.

Property (CDi: Independent command mapping)**.** $\exists \sigma' : \Psi^{\mathcal{S}} \to \left(\Psi^{\mathcal{T}}\right)^*.(\sigma_\Psi(\psi, \gamma) \equiv \sigma'(\psi))$

Property (CDt: Theory-dependent command mapping)**.** $\exists \sigma' : \Psi^{\mathcal{S}} \times \mathit{Th}(\mathcal{T}) \to \left(\Psi^{\mathcal{T}}\right)^*.(\sigma_\Psi(\psi, \gamma) \equiv \sigma'(\psi, \mathit{Th}(\gamma)))$

Property (CDs: State-dependent command mapping)**.** No restriction.

With *independent command* reductions, $\mathcal{S}$ commands must be precompiled to $\mathcal{T}$ commands which will work in any reachable $\mathcal{T}$ state. This is a restriction placed by [21]. *Theory dependent*

command mappings allow limited inspection of the $\mathcal{T}$ state; this restriction allows the sequence of $\mathcal{T}$ commands to be determined based only on the *theory* of the $\mathcal{T}$ state: the values of all $\mathcal{T}$ queries in the state. If two $\mathcal{T}$ states answer all queries the same way, the same $\mathcal{T}$ commands would be used in both to emulate an $\mathcal{S}$ command. With this restriction, the monitor that transforms $\mathcal{S}$ inputs into $\mathcal{T}$ procedures need not be more privileged than users of the access control system, since queries are the user's only API to observe the state.

Finally, *state-dependent* command mappings can arbitrarily observe the state. This requires a monitor that is privileged enough to observe elements of the state that are not queriable, and two states that answer all queries identically may emulate commands differently depending on unobservable state.

Dimension (CC: Command mapping complexity)**.**

Having considered the inputs available to the command mapping, we now consider the time complexity of this mapping. Note that this is measured as the increase in time as the *state* grows and thus is meaningless for independent command.

Property (CCc: Constant command mapping)**.** $\forall \psi \in \Psi^{\mathcal{S}}$, the algorithm for $\sigma_\psi(\gamma) = \sigma_\Psi(\psi, \gamma)$ has time complexity $T(n) \in \mathcal{O}(1)$

Property (CCl: Linear command mapping)**.** $\forall \psi \in \Psi^{\mathcal{S}}$, the algorithm for $\sigma_\psi(\gamma) = \sigma_\Psi(\psi, \gamma)$ has time complexity $T(n) \in \mathcal{O}(n)$

Property (CC∞: Unbounded command mapping)**.** No restriction.

Constant command reductions do not allow more processing time for bigger states. Thus, the command mapping cannot loop over sets within the state. With *linear command*, the command mapping can take time linear in the size of the state, e.g., looping over sets in the state, but cannot contain double loops over sets, sort sets, etc. Finally, *unbounded command* reductions put no limit on the complexity of the command mapping (though we may expect it to have to be tractable, e.g., poly-time).

Dimension (CS: Command mapping stuttering)**.**

Since the command mapping maps an $\mathcal{S}$ state to a sequence of $\mathcal{T}$ states, we may restrict the number of commands that can be used to emulate a single $\mathcal{S}$ command.

Property (CS1: Lock-step)**.** $\forall \psi \in \Psi^{\mathcal{S}}, \gamma \in \Gamma^{\mathcal{T}} : |\sigma_{\Psi}(\psi, \gamma)| \leq 1$

Property (CSc: Constant step)**.** $\exists c : \forall \psi \in \Psi^{\mathcal{S}}, \gamma \in \Gamma^{\mathcal{T}} : |\sigma_{\Psi}(\psi, \gamma)| \leq c$

Property (CS∞: Unbounded step)**.** No restriction.

A *lock-step* reduction allows at most one $\mathcal{T}$ command for each emulated $\mathcal{S}$ command. This mitigates concurrency issues for multiuser systems, since the system does not pass through potentially inconsistent states between command executions. *Constant step* reductions allow multiple commands to be used, but only a number constant in the size of the state. Thus, multiple actions can be taken, but not, e.g., a command for each user in the system. Finally, *unbounded step* does not restrict how many $\mathcal{T}$ commands can be executed per $\mathcal{S}$ command.

Dimension (CT: Trace structure)**.**

This class of properties enforces structural constraints on the traces of commands returned by the command mapping. This can address the potentially inconsistent states between start and end states in traces generated by the command mapping. Here, we present several examples of trace restrictions, using the notation $terminal(\gamma, \psi_1, \cdots, \psi_j)$ to denote the end state resulting from executing the sequence of commands $\psi_1, \cdots, \psi_j$, starting from the state γ. Note that this dimension of properties is not totally ordered.

Property (CT1: Semantic lock-step)**.**

$$
\begin{aligned}
&\forall \psi \in \Psi^{\mathcal{S}}, \gamma^{\mathcal{S}} \in \Gamma^{\mathcal{S}}, \gamma^{\mathcal{T}} \in \Gamma^{\mathcal{T}}.(\\
&\quad \exists \overline{\psi} = \langle \psi_1, \psi_2, \ldots, \psi_m \rangle \in \left(\Psi^{\mathcal{T}}\right)^*, i \in (1, m].(\\
&\qquad \sigma_{\Psi}\left(\psi, \gamma^{\mathcal{T}}\right) = \overline{\psi} \wedge \\
&\qquad \forall j \in [1, i).(\gamma^{\mathcal{S}} \sim \gamma^{\mathcal{T}} \Rightarrow \\
&\qquad\quad \gamma^{\mathcal{S}} \sim terminal\left(\gamma^{\mathcal{T}}, \psi_1 \cdots \psi_j\right)) \wedge \\
&\qquad \forall j \in [i, m].(\gamma^{\mathcal{S}} \sim \gamma^{\mathcal{T}} \Rightarrow \\
&\qquad\quad next\left(\gamma^{\mathcal{S}}, \psi\right) \sim terminal\left(\gamma^{\mathcal{T}}, \psi_1 \cdots \psi_j\right))))
\end{aligned}
$$

First, a *semantic lock-step* reduction can appear to be lock-step (i.e., it does not enter any inconsistent states), because even though it is allowed to execute multiple $\mathcal{T}$ commands to

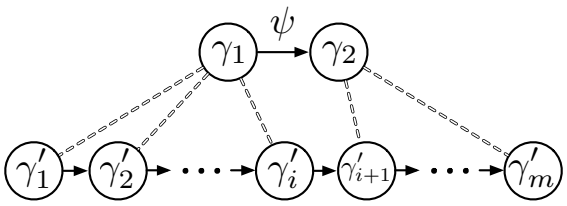


Figure 30: A graphical representation of semantic lock-step

emulate a single $\mathcal{S}$ command, only one of those commands is allowed to make correspondence-related changes. That is, consider the sequence of $\mathcal{T}$ states constructed by executing the sequence of commands $\sigma_\Psi\left(\psi^{\mathcal{S}}, \gamma^{\mathcal{T}}\right)$. In semantic lock-step, all of these states must correspond to the either the start state in $\mathcal{S}$ or the end state in $\mathcal{S}$, and once the transition from start state to end state is made, the remaining states must all be equivalent to the end state. Thus, from the point of view of a user who can ask any combination of queries, the reduction appears to be lock-step. This restriction is depicted in Fig. 30.

Property (CTq: Query monotonic)**.** $\forall \psi \in \Psi^{\mathcal{S}}, \gamma \in \Gamma^{\mathcal{T}}, q \in Q^{\mathcal{T}}.monotonic(\psi, \gamma, q)$, where:

$$
\begin{aligned}
&monotonic(\psi, \gamma, q) \triangleq \exists \overline{\psi} = \langle \psi_1, \psi_2, \ldots, \psi_m \rangle \in \left(\Psi^{\mathcal{T}}\right)^*.(\\
&\quad \sigma_\Psi(\psi, \gamma) = \overline{\psi} \wedge \\
&\quad \forall i \in (1, m).(\\
&\qquad terminal(\gamma, \psi_1 \cdots \psi_i) \vdash q \Rightarrow \\
&\qquad\quad (terminal(\gamma, \psi_1 \cdots \psi_{i-1}) \vdash q \vee terminal(\gamma, \psi_1 \cdots \psi_m) \vdash q) \wedge \\
&\qquad terminal(\gamma, \psi_1 \cdots \psi_i) \nvdash q \Rightarrow \\
&\qquad\quad (terminal(\gamma, \psi_1 \cdots \psi_{i-1}) \nvdash q \vee terminal(\gamma, \psi_1 \cdots \psi_m) \nvdash q)))
\end{aligned}
$$

Consider the start and end states of a trace in $\mathcal{T}$, γ and γ', respectively. Let Q^+ be the set of queries that become true in γ' that were false in γ, and Q^- be the set of queries that become false in γ' that were true in γ. During the trace from γ to γ', *query monotonicity* enforces that no queries are made true except Q^+, and no queries are made false except Q^-.

Thus, from the point of view of a user who can ask only single queries, the reduction appears to be lock-step.

Property (CTa: Access monotonic)**.** $\forall \psi \in \Psi^{\mathcal{S}}, \gamma \in \Gamma^{\mathcal{T}}, r \in \mathcal{R}^{\mathcal{T}}.monotonic(\psi, \gamma, r)$

Access monotonicity is similar to query monotonicity but considering only authorization requests. Let $\mathcal{R}^{+}$ be the set of requests that become allowed in γ' that were denied in γ, and $\mathcal{R}^{-}$ be the set of requests that become denied in γ' that were allowed in γ. During the trace from γ to γ', access monotonicity enforces that no requests are granted except $\mathcal{R}^{+}$, and no requests are revoked except $\mathcal{R}^{-}$.

Property (CTs: Non-contaminating)**.**

$$
\begin{aligned}
&\forall \psi \in \Psi^{\mathcal{S}}, \gamma^{\mathcal{T}} \in \Gamma^{\mathcal{T}}.(\\
&\quad \exists \overline{\psi} = \langle \psi_1, \psi_2, \ldots, \psi_m \rangle \in \left(\Psi^{\mathcal{T}}\right)^{*}.(\\
&\qquad \sigma_{\Psi}\left(\psi, \gamma^{\mathcal{T}}\right) = \overline{\psi} \wedge \\
&\qquad \forall \gamma_i^{\mathcal{T}} \in \left\{ \gamma_i^{\mathcal{T}} \mid \exists \psi_i \in \overline{\psi} : \gamma_i^{\mathcal{T}} = terminal\left(\gamma^{\mathcal{T}}, \psi_1 \cdots \psi_i\right) \right\}.(\\
&\qquad\quad Allowed\left(\gamma_i^{\mathcal{T}}\right) \subseteq Allowed\left(\gamma^{\mathcal{T}}\right) \vee \\
&\qquad\quad Allowed\left(\gamma_i^{\mathcal{T}}\right) \subseteq Allowed\left(terminal\left(\gamma^{\mathcal{T}}, \overline{\psi}\right)\right)))))
\end{aligned}
$$

The *non-contaminating* trace property ensures that no two accesses are allowed in the same state that are not both allowed in either the start or end state. This prevents, e.g., an intermediate state where a file can be accessed by two users simultaneously when emulating a command intended to *switch* which user can access the file. This definition uses the $Allowed(\gamma)$ notation, indicating the set of all permissions p allowed in state γ (i.e., such that $\gamma \vdash p$).

Dimension (CA: Actor preservation)**.**

Actor preservation properties restrict which users can be invoked in $\mathcal{T}$ to handle $\mathcal{S}$ commands. Here, we assume that $\alpha(\psi)$ denotes the actor executing the command ψ. Note that this requires system support (e.g., the executing actor being an implicit argument passed to a command) in order for a reduction to be executable.

Property (CA$\top$: Self-execution)**.** $\forall \psi^{\mathcal{S}} \in \Psi^{\mathcal{S}}, \gamma \in \Gamma^{\mathcal{T}}, \forall \psi^{\mathcal{T}} \in \sigma_{\Psi}\left(\psi^{\mathcal{S}}, \gamma\right), \alpha\left(\psi^{\mathcal{S}}\right) = \alpha\left(\psi^{\mathcal{T}}\right)$

Property (CAa: Administration-preservation)**.** Let A be the administrative subset of executing entities in the system. $\forall \psi^{\mathcal{S}} \in \Psi^{\mathcal{S}}, \gamma \in \Gamma^{\mathcal{T}}, \forall \psi^{\mathcal{T}} \in \sigma_{\Psi}\big(\psi^{\mathcal{S}}, \gamma\big), \alpha\big(\psi^{\mathcal{T}}\big) \in A \Rightarrow \alpha\big(\psi^{\mathcal{S}}\big) \in A$

Self-execution says that any command in $\mathcal{S}$ executed by any user u must be mapped to a sequence of commands in $\mathcal{T}$, all of which are executed by u. *Administration-preservation* prevents the invocation of administrators in $\mathcal{T}$ where they were not needed in $\mathcal{S}$. In an administration-preserving reduction, any command in $\mathcal{S}$ executed by a non-administrative user is mapped to a sequence of commands in $\mathcal{T}$, none of which is executed by an administrator. Other forms of actor preservation, as well as defining the set of administrators, are application-specific.

8.4.4 Query decider properties

Recall (Definition 30) that the query decider is a function $\sigma_Q = Q^{\mathcal{S}} \times \Gamma^{\mathcal{T}} \to \{\text{TRUE}, \text{FALSE}\}$ that assigns a truth value to each query in system $\mathcal{S}$, in each state in system $\mathcal{T}$. As with the command mapping, we can restrict what properties of the state the query decider can observe, but in general it sees the whole state. Thus, it allows us to answer $\mathcal{S}$ queries in an active emulation using $\mathcal{T}$. We now discuss the ways in which we can restrict this decider.

Dimension (QD: Query decider dependence)**.**

While Definition 30 maps each $\mathcal{S}$ query and $\mathcal{T}$ state to a truth value, previous works use more strict query deciders, ranging from mapping each $\mathcal{S}$ query q to a $\mathcal{T}$ query q' and returning $\gamma^{\mathcal{T}} \vdash q'$ [114], to mapping each $\mathcal{S}$ query and $\mathcal{T}$ *theory* to a truth value [59] (calculating the truth value by observing only the queriable portions of the $\mathcal{T}$ state). *Query decider dependence*, similar to command mapping dependence, restricts the information that the query decider can consider about a $\mathcal{T}$ state when deciding the truth value of an $\mathcal{S}$ query in that state.

Property (QD1: Independent, unitary-range query decider)**.** $\exists \sigma' : Q^{\mathcal{S}} \to Q^{\mathcal{T}}.(\sigma_Q(q, \gamma) \equiv \gamma \vdash \sigma'(q))$

Property (QDi: Independent query decider)**.** $\exists \sigma' : Q^{\mathcal{S}} \to \mathscr{L}\big(Q^{\mathcal{T}}\big).(\sigma_Q(q, \gamma) \equiv \gamma \vdash \sigma'(q))$

Property (QDt: Theory-dependent query decider)**.** $\exists\sigma' : Q^{\mathcal{S}} \times Th(\mathcal{T}) \rightarrow \{\text{TRUE}, \text{FALSE}\}.\sigma_Q(q, \gamma) \equiv \sigma'(q, Th(\gamma))$

Property (QDs: State-dependent query decider)**.** No restriction.

First, *independent, unitary-range query* reductions first map each $\mathcal{S}$ query $q^{\mathcal{S}}$ to a single $\mathcal{T}$ query $q^{\mathcal{T}}$, then return the truth value of asking this $\mathcal{T}$ query in $\gamma^{\mathcal{T}}$ (i.e., returns $\gamma^{\mathcal{T}} \vdash q^{\mathcal{T}}$). This type of reduction was used in [114]. *Independent query* reductions, similarly, map $q^{\mathcal{S}}$ independent of the state, but in this case can map it to an element of $\mathscr{L}(Q^{\mathcal{T}})$, a boolean expression over $Q^{\mathcal{T}}$, rather than a single query. This allows the reduction, for example, to map the $\mathcal{S}$ query "Is user u a member of role r" to $\mathcal{T}$ query sentence "Does user u exist and does user u have attribute role:r?"

Theory-dependent query allows the truth value to be determined based only on the *theory* of the $\mathcal{T}$ state, or the values of all $\mathcal{T}$ queries in the state. This allows the query decider to inspect the values of (potentially) all $\mathcal{T}$ queries, but does not allow the decider to consider any features of the state that cannot be queried using $Q^{\mathcal{T}}$. This version of the query decider is used in [59]. Finally, we refer to the general case as *state-dependent query*, since in this case the query decider can arbitrarily inspect the $\mathcal{T}$ state before returning a truth value for an $\mathcal{S}$ query, rather than being restricted only to those elements of the state which are observable by asking queries in $Q^{\mathcal{T}}$.

The biggest impact a selection in QD has is to limit the privilege of the emulation agent: under QD1 and QDi, the emulation agent need know nothing about the current state of the system to map queries, and emulated queries are mapped statically to dynamic queries. Under QDt, the emulation agent must be privileged to view the values of all queries at runtime, while QDs assumes the ability to arbitrarily inspect the state at runtime (even the portions of state that are not queriable).

Dimension (QC: Query decider complexity)**.**

As with the command mapping, due to potential resource constraints, we may enforce limits on the runtime complexity of the query decider routine, with respect to the size of the state it is executed in. As with commands, this is not applicable for independent query deciders.

Property (QCc: Constant query decider)**.** $\forall q \in Q^{\mathcal{S}}$, the algorithm for $\sigma_q(\gamma) = \sigma_Q(q, \gamma)$ has time complexity $T(n) \in \mathcal{O}(1)$

Property (QC∞: Unbounded query decider)**.** No restriction.

For theory-dependent and state-dependent query deciders, we can limit the complexity of the procedure—here, we consider only the constant-time restriction explicitly, though other restrictions may be useful in some cases.

Dimension (QP: Query preservation)**.**

Query preservation is a property dimension that indicates which queries need to stay the same as they are mapped from system $\mathcal{S}$ to system $\mathcal{T}$. A particular application may require any given set of queries to be preserved; here, we present several generic examples.

Property (QPf: Complete query preservation)**.** $\forall q \in Q^{\mathcal{S}}, \gamma \in \Gamma^{\mathcal{T}} : \sigma_Q(q, \gamma) \equiv \gamma \vdash q$

Property (QPa: Authorization preservation)**.** $\forall r \in \mathcal{R}^{\mathcal{S}}, \gamma \in \Gamma^{\mathcal{T}} : \sigma_Q(r, \gamma) \equiv \gamma \vdash r$

Property (QPw: Weak authorization preservation)**.** $\exists f : \mathcal{R}^{\mathcal{S}} \to \mathcal{R}^{\mathcal{T}}$ such that the following two conditions hold:

- $\forall r \in \mathcal{R}^{\mathcal{S}}, \gamma \in \Gamma^{\mathcal{T}}.(\sigma_Q(r, \gamma) = \text{TRUE} \Rightarrow \gamma \vdash f(r))$
- $\forall r \in \mathcal{R}^{\mathcal{T}}, \gamma \in \Gamma^{\mathcal{T}}.(\gamma \vdash r \Rightarrow \exists r^{\mathcal{S}}.(\sigma_Q\left(r^{\mathcal{S}}, \gamma\right) = \text{TRUE} \wedge f\left(r^{\mathcal{S}}\right) = r))$

The most common property in this dimension is *authorization preservation*, which roughly enforces that the query decider maps each $\mathcal{S}$ request to the value of the identical request in the $\mathcal{T}$ state. This requires $\mathcal{T}$ to accept at least the same authorization requests as $\mathcal{S}$, and can be seen as ensuring that $\mathcal{T}$ is using its model "as intended" (i.e., forcing it to answer emulated requests as it would its own native requests). Complete query preservation restricts the query decider in the same way, but for all supported queries.

Authorization preservation was formalized in [59] (as AC-preservation), but has been used implicitly in other reductions (e.g., [21]) that do not include any mapping from $\mathcal{S}$ requests to $\mathcal{T}$ requests (i.e., assume the identity mapping). For formalisms that do not consider queries other than requests, authorization preservation and complete query preservation are equivalent.

A related property is *weak authorization preservation*, which we defined in Section 4.5.2 (as weak AC-preservation, Definition 25). This property is a weakened version of authorization preservation: its intentions are similar, but the weak form can be used even when $\mathcal{S}$ and $\mathcal{T}$ do not support the exact same requests (i.e., emulating a system with requests of the form, "Does user u have access to permission p?" in a system with requests of the form, "Does subject s have access *read* to object o?"). Weak authorization preservation allows a request transformation function, which maps $\mathcal{S}$ requests to $\mathcal{T}$ requests. The definition of this property ensures that each $\mathcal{S}$ request is mapped into $\mathcal{T}$, and each $\mathcal{T}$ request that is granted represents some $\mathcal{S}$ request.

8.4.5 Reachability

Dimension (R: Reachability)**.**

The last dimension of properties we consider ties the mappings together to ensure the reduction is indeed what one could consider a reduction in the classic sense. A state correspondence, query decider, and command mapping do not automatically define a reduction without reachability constraints. Here, we define forward and bidirectional reachability, two variants of this type of constraint (note that these properties are presented in *increasing* strictness since the latter builds upon the former).

Property (R$\rightarrow$: Forward reachability)**.**

$$\begin{aligned}&\forall \gamma_0^{\mathcal{S}}, \gamma_1^{\mathcal{S}} \in \Gamma^{\mathcal{S}}, \gamma_0^{\mathcal{T}} \in \Gamma^{\mathcal{T}}.(\\ &\quad \gamma_0^{\mathcal{S}} \sim \gamma_0^{\mathcal{T}} \wedge \gamma_0^{\mathcal{S}} \mapsto \gamma_1^{\mathcal{S}} \Rightarrow \exists \gamma_1^{\mathcal{T}} \in \Gamma^{\mathcal{T}}.(\\ &\quad\quad \gamma_0^{\mathcal{T}} \overset{*}{\mapsto} \gamma_1^{\mathcal{T}} \wedge \gamma_1^{\mathcal{S}} \sim \gamma_1^{\mathcal{T}}))\end{aligned}$$

Property (R$\leftrightarrow$: Bidirectional reachability)**.** Forward reachability, and:

$$\begin{aligned}&\forall \gamma_0^{\mathcal{S}} \in \Gamma^{\mathcal{S}}, \gamma_0^{\mathcal{T}}, \gamma_1^{\mathcal{T}} \in \Gamma^{\mathcal{T}}.(\\ &\quad \gamma_0^{\mathcal{S}} \sim \gamma_0^{\mathcal{T}} \wedge \gamma_0^{\mathcal{T}} \mapsto \gamma_1^{\mathcal{T}} \Rightarrow \exists \gamma_1^{\mathcal{S}} \in \Gamma^{\mathcal{S}}.(\\ &\quad\quad \gamma_0^{\mathcal{S}} \overset{*}{\mapsto} \gamma_1^{\mathcal{S}} \wedge \gamma_1^{\mathcal{S}} \sim \gamma_1^{\mathcal{T}}))\end{aligned}$$

In *forward reachability*, any transition made in $\mathcal{S}$ must be possible in $\mathcal{T}$. If $\gamma_0^{\mathcal{S}}$ corresponds to $\gamma_0^{\mathcal{T}}$, and $\gamma_1^{\mathcal{S}}$ can be reached from $\gamma_0^{\mathcal{S}}$ via the commands of $\mathcal{S}$, then $\gamma_1^{\mathcal{S}}$ must correspond to a state $\gamma_1^{\mathcal{T}}$ in $\mathcal{T}$ that is reachable from $\gamma_0^{\mathcal{T}}$. The notion of state correspondence is determined by the property chosen in dimension SC.

Bidirectional reachability (or bi-reachability), also requries that $\mathcal{T}$ cannot enter a state that does not correspond to a reachable state in $\mathcal{S}$. If $\gamma_0^{\mathcal{S}}$ corresponds to $\gamma_0^{\mathcal{T}}$, and $\gamma_1^{\mathcal{T}}$ is reachable from $\gamma_0^{\mathcal{T}}$ by executing a command, then there must exist an $\mathcal{S}$ state $\gamma_1^{\mathcal{S}}$ that corresponds to $\gamma_1^{\mathcal{T}}$ and that is reachable from $\gamma_0^{\mathcal{S}}$ by executing one or more commands. This process may make use of multiple steps, since the procedure for finding the corresponding $\mathcal{S}$ states does not need to be constructed, these states must simply exist. The operational advantage of enforcing R$\leftrightarrow$ is that, even if the emulating system's native operations are exposed to users, the system can never enter a state that does not have an equivalent in the emulated system.

8.5 POSITIONING EXISTING REDUCTIONS

As mentioned in Section 8.4.1, no set of properties can be proven to describe all conceivable reductions. In this section, we support the set of properties defined in this work by showing that it can *precisely* describe the wide range of existing expressiveness reductions.

8.5.1 Expressiveness using Reduction Properties

We will now draw the formal distinction between a reduction and expressiveness. Here, we use $\mathcal{T} \underset{\mathcal{X}}{emu} \mathcal{S}$ to denote, "$\mathcal{T}$ can admit a reduction of type $\mathcal{X}$ of $\mathcal{S}$," and $\mathcal{S} \leq^{\mathcal{X}} \mathcal{T}$ to denote, "$\mathcal{T}$ is at least as expressive as $\mathcal{S}$ with respect to reductions of type $\mathcal{X}$."

While previous work considers the expressiveness result to be equivalent to a reduction (i.e., $\mathcal{T} \underset{\mathcal{X}}{emu} \mathcal{S} \equiv \mathcal{S} \leq^{\mathcal{X}} \mathcal{T}$), expressiveness in a practical sense is subject to a subtle distinction. Since we mean for expressiveness to be *implementable* (i.e., if $\mathcal{T}$ is as expressive as $\mathcal{S}$, then $\mathcal{T}$ can be used in place of $\mathcal{S}$), expressiveness within the domain of reduction properties should

mean the following: if $\mathcal{T}$ is as expressive as $\mathcal{S}$, then $\mathcal{T}$ can emulate any system that $\mathcal{S}$ can emulate. Thus, we define *expressiveness* in the context of a set of reduction properties.

Definition 31 (Expressiveness)**.** Given access control systems $\mathcal{S}$ and $\mathcal{T}$ and a set of reduction properties $\mathcal{P}$, we say that $\mathcal{T}$ is *at least as expressive* as $\mathcal{S}$ with respect to $\mathcal{P}$ (denoted $\mathcal{S} \leq^{\mathcal{P}} \mathcal{T}$) to mean that, for every system $\mathcal{U}$, if $\mathcal{S}$ can emulate $\mathcal{U}$ while enforcing $\mathcal{P}$, then $\mathcal{T}$ can emulate $\mathcal{U}$ while enforcing $\mathcal{P}$ ($\forall \mathcal{U} : \mathcal{S} \underset{\mathcal{P}}{emu}\, \mathcal{U} \Rightarrow \mathcal{T} \underset{\mathcal{P}}{emu}\, \mathcal{U}$). $\Diamond$

We first point out that this definition of expressiveness is strictly more general than the more traditional (often implied) notion. Since $\mathcal{S}$ can trivially emulate itself, $\mathcal{S} \leq^{\mathcal{X}} \mathcal{T}$ implies $\mathcal{T} \underset{\mathcal{X}}{emu}\, \mathcal{S}$. The additional generalization can be viewed from a formal standpoint as dropping the (incorrect) assumption that all types of reduction are transitive (i.e., that $\mathcal{T}\ emu\ \mathcal{S}$ and $\mathcal{S}\ emu\ \mathcal{U}$ imply $\mathcal{T}\ emu\ \mathcal{U}$). For instance, assume that $\mathcal{T}$ can emulate $\mathcal{S}$ and $\mathcal{S}$ can emulate $\mathcal{U}$, each with a quadratic increase in state storage. While $\mathcal{T}$ may be able to emulate $\mathcal{U}$, this reduction may require greater than quadratic storage.

From a more intuitive standpoint, we point out that, except in the case of custom-built access control solutions, any deployment is an emulation of a workload (i.e., ideal operation) using an existing system. That is, unless $\mathcal{S}$ is custom-made to exactly satisfy the desired workload, replacing it with $\mathcal{T}$ is not a matter of whether $\mathcal{T}$ can emulate $\mathcal{S}$, but whether $\mathcal{T}$ can admit an equally good emulation of the (perhaps not formally specified) workload that $\mathcal{S}$ is known to emulate. This concept is discussed by Kane and Browne [69], who point out that an access control implementation is often only an approximation of the desired policy. In particular, as policy languages get more complex, deployments often make use of approximations that are easier to analyze and more efficient to enforce than the overly-expressive policy language.

8.5.2 Decomposing Expressiveness Reductions to Properties

In order to use the set of expressiveness reduction properties detailed in Section 8.4 to systematically compare previously proposed notions of reduction, we present our formal way of stating that a notion of reduction and a set of reduction properties are equivalent. We call this correspondence *reduction decomposition*: when a notion of reduction $\mathcal{X}$ can be

decomposed to a set of reduction properties $\mathcal{P}$, then analyses using $\mathcal{X}$ and $\mathcal{P}$ yield equivalent expressiveness results.

Definition 32 (Reduction Decomposition)**.** Given a notion of access control reduction $\mathcal{X}$ and a set of reduction properties $\mathcal{P}$, $\mathcal{X}$ can be *decomposed* to $\mathcal{P}$ (denoted $\mathcal{X} \doteq \mathcal{P}$) if and only if, for all systems $\mathcal{S}$ and $\mathcal{T}$, $\mathcal{T} \underset{\mathcal{X}}{emu} \mathcal{S} \iff \mathcal{S} \leq^{\mathcal{P}} \mathcal{T}$. That is, $\mathcal{T}$ admits an $\mathcal{X}$ reduction of $\mathcal{S}$ if and only if $\mathcal{T}$ is at least as expressive as $\mathcal{S}$ with respect to properties $\mathcal{P}$. ◊

Recall from Definition 31 that $\mathcal{S} \leq^{\mathcal{P}} \mathcal{T}$ says that any system that can be emulated by $\mathcal{S}$ while preserving properties $\mathcal{P}$ can can also emulated by $\mathcal{T}$ while preserving $\mathcal{P}$. In light of this, we will position an existing notion of reduction, $\mathcal{X}$, within the lattice formed by our reduction properties (i.e., prove $\mathcal{X} \doteq \mathcal{P}$) by proving the following for the set of properties $\mathcal{P}$:

1. (*Only-if direction*) $\mathcal{T} \underset{\mathcal{X}}{emu} \mathcal{S} \wedge \mathcal{S} \underset{\mathcal{P}}{emu} \mathcal{U} \Rightarrow \mathcal{T} \underset{\mathcal{P}}{emu} \mathcal{U}$
2. (*If direction*) $\mathcal{S} \leq^{\mathcal{P}} \mathcal{T} \Rightarrow \mathcal{T} \underset{\mathcal{X}}{emu} \mathcal{S}$

We give an example of such a proof in the following section.

8.5.3 Example Decomposition

To demonstrate how reduction decomposition proofs are written, we now consider the Ammann-Lipton-Sandhu (ALS) reduction [3], which we first discussed in Chapter 2. The ALS reduction considers access control states as graphs: sets of primitive objects are node types, and sets of relations are edge types. The set of node types and edge types in the states of system $\mathcal{S}$ are denoted $NT(\mathcal{S})$ and $ET(\mathcal{S})$, respectively. The ALS state correspondence is then defined as follows (reworded slightly from [3]).

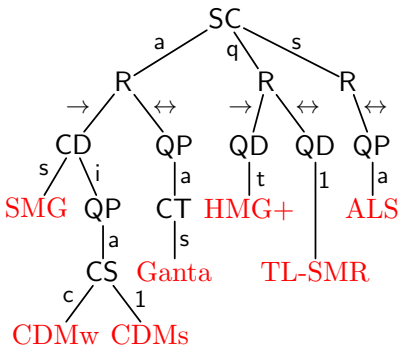
Definition 5 (ALS State Correspondence, restated)**.** A state in system $\mathcal{S}$, an emulated system, and a state in system $\mathcal{T}$, an emulating system, *correspond* iff the graph defining the state in $\mathcal{S}$ is identical to the subgraph obtained by taking the state in $\mathcal{T}$ and discarding all nodes (edges) not in $NT(\mathcal{S})$ ($ET(\mathcal{S})$). ◊

The ALS reduction is defined with respect to this state correspondence.

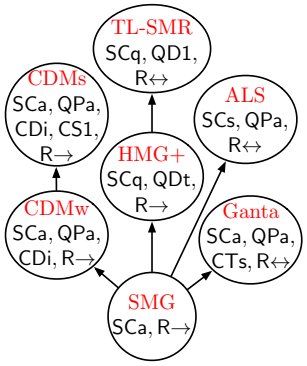
Definition 6 (ALS Reduction, restated)**.** Under the definition of correspondence in Definition 5, system $\mathcal{T}$ *emulates* system $\mathcal{S}$ iff the following conditions hold:

Reduction	Decomposition
ALS	SCs QPa R↔
CDMw	SCa QPa CDi R→
CDMs	SCa QPa CDi CS1 R→
Ganta	SCa QPa CTs R↔
HMG+	SCq QDt R→
HMG+a	QPa
HMG+s	CTa
HMG+p	CAa
SMG	SCa R→
TL-SMR	SCq QD1 R↔

(a) Decompositions of surveyed reductions



(b) Taxonomy of reductions



(c) Partial lattice of reductions

Figure 31: Results of decomposing notions of access control reduction from the literature

1. If system $\mathcal{S}$ can reach a given state, system $\mathcal{T}$ can reach a corresponding state.
2. If system $\mathcal{T}$ can reach a given state, system $\mathcal{S}$ can reach a corresponding state.

$\Diamond$

We will now demonstrate the two-step reduction decomposition proof technique described in Section 8.5.2 for the ALS reduction. For the purposes of this proof, let the set of reduction properties $\mathcal{P} = \{\mathsf{SCs}, \mathsf{QPa}, \mathsf{R}\leftrightarrow\}$. Recall that SCs is structure state correspondence, which says that the emulating state must include all of the unaltered sets from the emulated state; QPa is authorization preservation, which says that each authorization request must be mapped identically from emulated to emulating system (and thus the emulating system must support the same set of requests as the emulated system); and R↔ is bireachability, which says that the emulating system can reach a state which corresponds to each reachable emulated state, and cannot reach a state which does not correspond to a reachable state in the emulated system.

We will demonstrate the two steps of the proof technique by proving two requesite lemmas. First, step 1 (only-if direction):

Lemma 19. *Given access control systems $\mathcal{S}$, $\mathcal{T}$, and $\mathcal{U}$,*

$$\mathcal{T} \underset{ALS}{emu} \mathcal{S} \wedge \mathcal{S} \underset{\mathcal{P}}{emu} \mathcal{U} \Rightarrow \mathcal{T} \underset{\mathcal{P}}{emu} \mathcal{U}$$

That is, if $\mathcal{T}$ admits an ALS reduction of $\mathcal{S}$, and $\mathcal{S}$ admits a reduction of $\mathcal{U}$ with properties $\{\mathsf{SCs}, \mathsf{QPa}, \mathsf{R}\leftrightarrow\}$, then $\mathcal{T}$ admits a reduction of $\mathcal{U}$ with properties $\{\mathsf{SCs}, \mathsf{QPa}, \mathsf{R}\leftrightarrow\}$.

Proof. To prove this lemma, we let $\mathcal{S}$, $\mathcal{T}$, and $\mathcal{U}$ be access control systems such that $\mathcal{T} \underset{ALS}{emu} \mathcal{S}$ and $\mathcal{S} \underset{\mathcal{P}}{emu} \mathcal{U}$ but are otherwise arbitrary, and we show that $\mathcal{T} \underset{\mathcal{P}}{emu} \mathcal{U}$.

Choose an arbitrary state $\gamma_0^{\mathcal{U}} \in \Gamma^{\mathcal{U}}$ and command $\psi^{\mathcal{U}} \in \Psi^{\mathcal{U}}$, and let $next\big(\gamma_0^{\mathcal{U}}, \psi^{\mathcal{U}}\big) = \gamma_1^{\mathcal{U}}$. Let $\gamma_0^{\mathcal{S}} \in \Gamma^{\mathcal{S}}$ such that $\gamma_0^{\mathcal{U}} \overset{s}{\sim} \gamma_0^{\mathcal{S}}$. Since $\mathcal{S} \underset{\mathcal{P}}{emu} \mathcal{U}$,

$$\exists \gamma_1^{\mathcal{S}} \in \Gamma^{\mathcal{S}}.(terminal\big(\gamma_0^{\mathcal{S}}, \sigma_\Psi\big(\psi^{\mathcal{U}}, \gamma_0^{\mathcal{S}}\big)\big) = \gamma_1^{\mathcal{S}} \wedge \gamma_1^{\mathcal{U}} \overset{s}{\sim} \gamma_1^{\mathcal{S}})$$

Let $\gamma_0^{\mathcal{T}} \in \Gamma^{\mathcal{T}}$ such that $\gamma_0^{\mathcal{S}} \overset{s}{\sim} \gamma_0^{\mathcal{T}}$. Since $\mathcal{T} \underset{ALS}{emu} \mathcal{S}$,

$$\exists \gamma_1^{\mathcal{T}} \in \Gamma^{\mathcal{T}}.(\gamma_0^{\mathcal{T}} \overset{*}{\mapsto} \gamma_1^{\mathcal{T}} \wedge \gamma_1^{\mathcal{S}} \overset{s}{\sim} \gamma_1^{\mathcal{T}})$$

Thus, there exists a sequence of $\mathcal{T}$ commands $\Psi_0^{\mathcal{T}}$ such that $terminal\big(\gamma_0^{\mathcal{T}}, \Psi_0^{\mathcal{T}}\big) = \gamma_1^{\mathcal{T}}$. Define $\sigma_\Psi : \Psi^{\mathcal{U}} \times \Gamma^{\mathcal{T}} \to \big(\Psi^{\mathcal{T}}\big)^*$ such that it returns $\Psi_0^{\mathcal{T}}$ for $\gamma_0^{\mathcal{T}}, \psi^{\mathcal{U}}$.

Then, given $\gamma_0^{\mathcal{U}}, \gamma_1^{\mathcal{U}} \in \Gamma^{\mathcal{U}}, \gamma_0^{\mathcal{T}} \in \Gamma^{\mathcal{T}}, \psi^{\mathcal{U}} \in \Psi^{\mathcal{U}}$ such that $next\big(\gamma_0^{\mathcal{U}}, \psi^{\mathcal{U}}\big) = \gamma_1^{\mathcal{U}}$, and $\gamma_0^{\mathcal{U}} \overset{s}{\sim} \gamma_0^{\mathcal{T}}$,

$$\exists \gamma_1^{\mathcal{T}} \in \Gamma^{\mathcal{T}}.(terminal\big(\gamma_0^{\mathcal{T}}, \sigma_\Psi\big(\psi, \gamma_0^{\mathcal{T}}\big)\big) = \gamma_1^{\mathcal{T}} \wedge \gamma_1^{\mathcal{S}} \overset{s}{\sim} \gamma_1^{\mathcal{T}})$$

Hence, $\mathcal{T} \underset{\{\mathsf{SCs},\mathsf{R}\rightarrow\}}{emu} \mathcal{U}$. Next, we show QPa.

Choose some arbitrary request $r_0^{\mathcal{U}} \in \mathcal{R}^{\mathcal{U}}$ and state $\gamma_0^{\mathcal{T}} \in \Gamma^{\mathcal{T}}$.

Since $\mathcal{S} \underset{\mathcal{P}}{emu} \mathcal{U}$,

$$\forall r^{\mathcal{U}} \in \mathcal{R}^{\mathcal{U}}, \gamma^{\mathcal{S}} \in \Gamma^{\mathcal{S}}, \sigma_Q\big(r^{\mathcal{U}}, \gamma^{\mathcal{S}}\big) = \gamma^{\mathcal{S}} \vdash r^{\mathcal{U}}$$

Thus, we know that $\mathcal{S}$ supports all $\mathcal{U}$ requests, and corresponding $\mathcal{S}$ and $\mathcal{U}$ states will answer $\mathcal{U}$ requests identically. Therefore, $r_0^{\mathcal{U}} \in \mathcal{R}^{\mathcal{S}}$. Since $\mathcal{T} \underset{ALS}{emu} \mathcal{S}$,

$$\forall r^{\mathcal{S}} \in \mathcal{R}^{\mathcal{S}}, \gamma^{\mathcal{T}} \in \Gamma^{\mathcal{T}}, \sigma_Q\big(r^{\mathcal{S}}, \gamma^{\mathcal{T}}\big) = \gamma^{\mathcal{T}} \vdash r^{\mathcal{S}}$$

Thus, $\sigma_Q\big(r_0^{\mathcal{U}}, \gamma^{\mathcal{T}}\big) = \gamma^{\mathcal{T}} \vdash r_0^{\mathcal{U}}$.

Hence, $\mathcal{T} \underset{\{\mathsf{SCs},\mathsf{QPa},\mathsf{R}\rightarrow\}}{emu} \mathcal{U}$. Next, we show $\mathsf{R}\leftrightarrow$.

Choose some arbitrary states $\gamma_0^{\mathcal{T}}, \gamma_1^{\mathcal{T}} \in \Gamma^{\mathcal{T}}$ such that $\gamma_0^{\mathcal{T}} \mapsto \gamma_1^{\mathcal{T}}$. Let $\gamma_0^{\mathcal{S}} \in \Gamma^{\mathcal{S}}$ such that $\gamma_0^{\mathcal{S}} \overset{s}{\sim} \gamma_0^{\mathcal{T}}$. Since $\mathcal{T} \underset{ALS}{emu} \mathcal{S}$,

$$\exists \gamma_1^{\mathcal{S}}.(\gamma_0^{\mathcal{S}} \overset{*}{\mapsto} \gamma_1^{\mathcal{S}} \wedge \gamma_1^{\mathcal{S}} \overset{s}{\sim} \gamma_1^{\mathcal{T}})$$

Let $\gamma_0^{\mathcal{U}} \in \Gamma^{\mathcal{U}}$ such that $\gamma_0^{\mathcal{U}} \overset{s}{\sim} \gamma_0^{\mathcal{S}}$. Since $\mathcal{S} \underset{\mathcal{P}}{emu} \mathcal{U}$,

$$\exists \gamma_1^{\mathcal{U}}.(\gamma_0^{\mathcal{U}} \overset{*}{\mapsto} \gamma_1^{\mathcal{U}} \wedge \gamma_1^{\mathcal{U}} \overset{s}{\sim} \gamma_1^{\mathcal{S}})$$

Thus, given $\gamma_0^{\mathcal{T}}, \gamma_1^{\mathcal{T}} \in \Gamma^{\mathcal{T}}, \gamma_0^{\mathcal{U}} \in \Gamma^{\mathcal{U}}$ such that $\gamma_0^{\mathcal{T}} \mapsto \gamma_1^{\mathcal{T}}$ and $\gamma_0^{\mathcal{U}} \overset{s}{\sim} \gamma_0^{\mathcal{T}}$,

$$\exists \gamma_1^{\mathcal{U}} \in \Gamma^{\mathcal{U}}.(\gamma_0^{\mathcal{U}} \overset{*}{\mapsto} \gamma_1^{\mathcal{U}} \wedge \gamma_1^{\mathcal{U}} \overset{s}{\sim} \gamma_1^{\mathcal{T}})$$

Hence, $\mathcal{T} \underset{\mathcal{P}}{emu} \mathcal{U}$. □

Next, we demonstrate step 2 (if direction):

Lemma 20. *Given access control systems $\mathcal{S}$ and $\mathcal{T}$ and reduction properties $\mathcal{P} = \{\mathsf{SCs}, \mathsf{QPa}, \mathsf{R}\leftrightarrow\}$, $\mathcal{S} \leq^{\mathcal{P}} \mathcal{T} \Rightarrow \mathcal{T} \underset{ALS}{emu} \mathcal{S}$. That is, if $\mathcal{T}$ is at least as expressive as $\mathcal{S}$ with respect to properties $\mathcal{P}$, then $\mathcal{T}$ admits an ALS reduction of $\mathcal{S}$.*

Proof. To prove this lemma, we let $\mathcal{S}$ and $\mathcal{T}$ be arbitrary access control systems such that $\mathcal{S} \leq^{\mathcal{P}} \mathcal{T}$, and we show that $\mathcal{T} \underset{ALS}{emu} \mathcal{S}$.

Since $\mathcal{S} \leq^{\mathcal{P}} \mathcal{T}$, for any access control system $\mathcal{U}$, if $\mathcal{S} \underset{\mathcal{P}}{emu} \mathcal{U}$, then $\mathcal{T} \underset{\mathcal{P}}{emu} \mathcal{U}$.

Since $\mathcal{S}$ can trivially emulate itself, $\mathcal{S} \underset{\mathcal{P}}{emu} \mathcal{S}$, and thus $\mathcal{T} \underset{\mathcal{P}}{emu} \mathcal{S}$.

Thus, given $\gamma_0^{\mathcal{S}}, \gamma_1^{\mathcal{S}} \in \Gamma^{\mathcal{S}}, \gamma_0^{\mathcal{T}} \in \Gamma^{\mathcal{T}}$, by forward reachability, if $\gamma_0^{\mathcal{S}} \overset{s}{\sim} \gamma_0^{\mathcal{T}}$ and $\gamma_0^{\mathcal{S}} \mapsto \gamma_1^{\mathcal{S}}$, then

$$\exists \gamma_1^{\mathcal{T}}.(\gamma_0^{\mathcal{T}} \overset{*}{\mapsto} \gamma_1^{\mathcal{T}} \wedge \gamma_1^{\mathcal{S}} \overset{s}{\sim} \gamma_1^{\mathcal{T}})$$

Since SCs and QPa satisfy the ALS definition of state correspondence, this means we have satisfied the first property of the ALS reduction.

1. If $\mathcal{S}$ can reach a given state, $\mathcal{T}$ can reach a corresponding state.

And by bidirectional reachability, given $\gamma_0^{\mathcal{S}} \in \Gamma^{\mathcal{S}}, \gamma_0^{\mathcal{T}}, \gamma_1^{\mathcal{T}} \in \Gamma^{\mathcal{T}}$, if $\gamma_0^{\mathcal{S}} \overset{s}{\sim} \gamma_0^{\mathcal{T}}$ and $\gamma_0^{\mathcal{T}} \mapsto \gamma_1^{\mathcal{T}}$, then

$$\exists \gamma_1^{\mathcal{S}}.(\gamma_0^{\mathcal{S}} \overset{*}{\mapsto} \gamma_1^{\mathcal{S}} \wedge \gamma_1^{\mathcal{S}} \overset{s}{\sim} \gamma_1^{\mathcal{T}})$$

And therefore, we have satisfied the second property of the ALS reduction:

2. If $\mathcal{T}$ can reach a given state, $\mathcal{S}$ can reach a corresponding state.

These properties satisfy the definition for ALS reduction, and hence $\mathcal{T}$ admits an ALS reduction of $\mathcal{S}$ ($\mathcal{T} \underset{ALS}{emu} \mathcal{S}$). □

Therefore, we have proved the decomposition of the ALS reduction:

Theorem 21. *ALS* $\doteq \{\mathsf{SCs}, \mathsf{QPa}, \mathsf{R}\leftrightarrow\}$*; that is, the ALS reduction decomposes to structure correspondence, authorization preservation, and bidirectional reachability.*

Proof. By Lemma 19, if $\mathcal{T} \underset{ALS}{emu} \mathcal{S}$, then $\mathcal{S} \leq^{\mathcal{P}} \mathcal{T}$. By Lemma 20, if $\mathcal{S} \leq^{\mathcal{P}} \mathcal{T}$, then $\mathcal{T} \underset{ALS}{emu} \mathcal{S}$. Thus, $\mathcal{S} \leq^{\mathcal{P}} \mathcal{T}$ if and only if $\mathcal{T} \underset{ALS}{emu} \mathcal{S}$, and thus the ALS reduction decomposes to $\{\mathsf{SCs}, \mathsf{QPa}, \mathsf{R}\leftrightarrow\}$. □

All other decomposition proofs can be found in Appendix B.

8.5.4 Results

Now, we present the results of decomposing the reductions from the series of previous works discussed in Section 8.2 into sets of reduction properties from Section 8.4. First, a chart of our results is shown in Figure 31a, which states the decomposition of the SMG reduction [85, 89, 101, 105–107], the Ganta reduction [41], the ALS reduction [2, 3], the CDM weak and strong reductions [21], the TL state-matching reduction [113, 114], and HMG+ parameterized expressiveness (along with several parameterized expressiveness properties) [59]. Properties are omitted if they are not explicitly required by the reduction's definition but are implied by other, explicit properties (e.g., CDMs decomposes to a set including CDi, which also implies CCc). Section 8.6.1 discusses which properties imply others.

In Figure 31b, we arrange this data as a taxonomy, with each split representing a dimension, with weaker properties positioned to the left and stronger properties to the right. We split first on the state correspondence, which is perhaps the biggest difference among the surveyed reductions. This separates reductions that preserve only the answers to authorization requests (SCa) from those that preserve all queries (SCq) and those that preserve full state structure (SCs). We note that the ALS reduction is alone in its decomposition including SCs; all other surveyed reductions allowed the emulating system to store information in a different

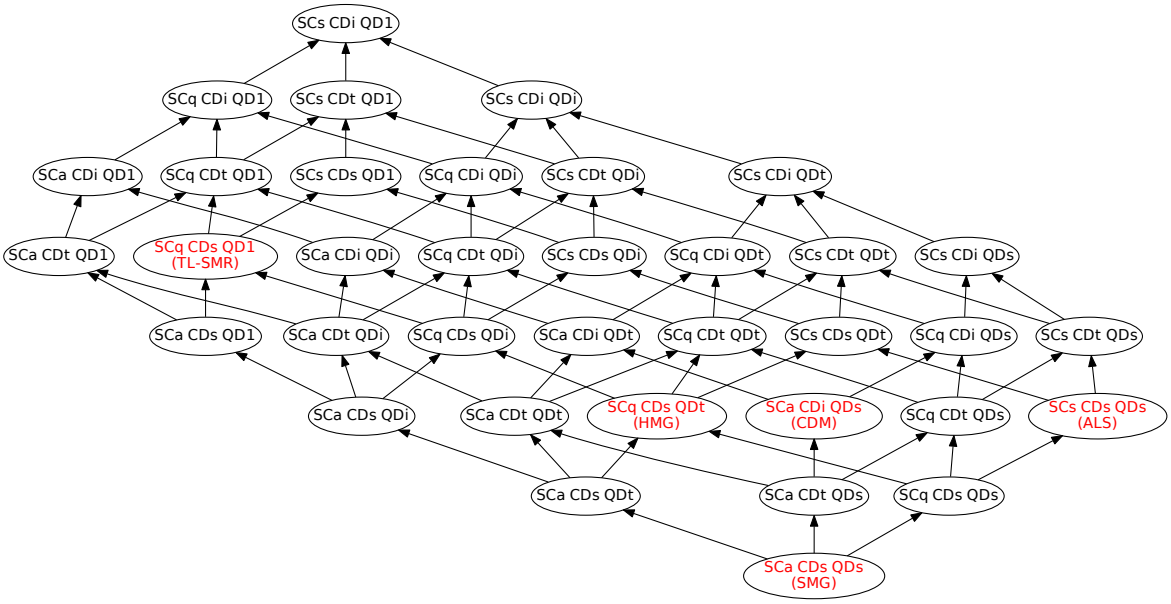


Figure 32: Lattice of state correspondence, command dependence, and query dependence with positioned surveyed reductions

organization than the emulated system, so long as the required queriable information (requests or queries) can be recovered. We also note that the predominant difference between the SMG reduction and the CDM reductions is the command dependence: in SMG, a command can be mapped completely differently if it is to be executed in different states, while in CDM, each command must be mapped without knowing the state in which it will be executed. The Ganta reduction is unique in enforcing the non-contamination trace restriction. HMG+ and TL-SMR use the same state correspondence, but HMG+ enforces a more relaxed query dependence and does not require bireachability. Reductions that are positioned farther apart are the most dissimilar. Most starkly different are SMG and ALS, positioned far left and far right, which share no reduction properties except in dimensions in which both enforce only minimum properties, despite their similar publication times.

In Figure 31c, we position the surveyed reductions within a lattice. Higher reductions decompose to more strict properties, and an arrow from reduction $\mathcal{X}$ to reduction $\mathcal{X}'$ indicates

that $\mathcal{X}'$ decomposes to strictly stronger properties than $\mathcal{X}$. Here we can see that the SMG reduction is strictly weakest, which supports previous claims to this effect [41, 114]. Several orthogonal directions were taken in defining other reductions to enforce stronger properties. The CDM reductions, as noted above, restrict the command dependence. The Ganta reduction requires non-contamination and bireachability. The TL state-matching reduction and HMG+ parameterized expressiveness consider queries, and thus strengthen the state correspondence. The ALS reduction enforces an even more strict state correspondence, requiring the structure of a emulated system's state to be preserved in the emulating system. Interestingly, we note that while all are stronger than SMG, most pairs are incomparable due to being stronger in orthogonal ways. In particular, while TL-SMR is considered to be a relatively strong notion of reduction, this is not substantiated by the lattice, which shows TL-SMR to be stronger than HMG+ and SMG, but incomparable to the CDM, ALS, and Ganta reductions.

Figure 32 presents a lattice of state correspondence, command dependence, and query dependence, with the surveyed reductions positioned within it (in this space, the Ganta reduction is at the same point as the SMG reduction). This figure makes evident the wide range of points between existing reductions that have not been explored. In this figure, we omit several dimensions for readability, namely reachability (which further separates Ganta, ALS, and TL-SMR from SMG, CDM and HMG+) and stuttering (which would break CDM into its weak and strong counterparts). Perhaps the most interesting points to explore within this lattice are those that exist between two surveyed reductions. For example, $\{\mathsf{SCq}, \mathsf{CDs}, \mathsf{QDs}\}$ adds to SMG the preservation of queries beyond requests, but stops short of HMG+ by not restricting the query decider to consider only the theory of the state while mapping queries. Similarly, $\{\mathsf{SCa}, \mathsf{CDt}, \mathsf{QDs}\}$ takes away some of SMG's freedom to inspect the state mapping commands, but rather than go all the way to the independent command mapping of CDM, it still allows it to inspect the state's responses to queries. We also point out $\{\mathsf{SCq}, \mathsf{CDs}, \mathsf{QDi}\}$, which differs from HMG+ by enforcing query decider independence (mapping queries cannot consider the state or theory), but can map each emulated query to a boolean expression over emulating queries.

8.6 SELECTING NEW SETS OF PROPERTIES

In Section 8.5, we positioned the reductions used in previous works within a comparative lattice, allowing them to be formally compared for the first time. In this section, we enable a second use of our lattice of expressiveness reduction properties: crafting new notions of expressiveness by choosing the properties that most closely correspond to the scenario in which an access control system will be deployed. We first discuss interactions between dimensions; this discussion should act as a warning against choosing individual properties in isolation. We then interpret the impact each identified dimension has on the reduction, and identify properties of a deployment scenario that may dictate particular choices in each dimension. Finally, we discuss the potential impact these techniques could have on future expressiveness analysis.

8.6.1 Interactions Between Dimensions

We noted in Section 8.5 that some reductions decompose to sets of properties that include *implied* properties, or properties that are redundant given the others in the set. For instance, command independence (CDi) implies constant-time command mapping (CCc); if the command mapping does not depend on the state, then its procedure must be constant-time in the size of the state. Further, CCc implies constant step (CSc), since a constant-time procedure must have constant-size output.

Independence query decider (QDi) implies constant query decider (QCc), since a query decider that does not depend on the state must be constant-time with respect to the state size. Full query preservation (QPf) implies constant query decider (QCc), since the identity mapping is a constant-time procedure, and implies unitary-range query decider (QD1), since the identity mapping always outputs only one query.

An additional type of interaction is between basic properties and those properties whose definition relies on the basic properties in the abstract. For example, the definition of forward reachability (R$\rightarrow$) refers to sequences of commands output by σ^{Ψ}, the length of which may be limited by command mapping stuttering (CS). Further, the definitions of both reachability

properties (R) and trace structure properties (CT) refer to corresponding states. Here, the details of what makes states correspond is left to the state correspondence structure (SC).

These dependencies show that the proof of a property in one dimension may rely on the properties chosen in another. Thus, e.g., changing to a stronger state correspondence requires re-proving a reduction's results for reachability and trace structure, since these are dependent on state correspondence.

Several property dimensions are defined over the size of the emulated state: command mapping complexity (CC), command mapping stuttering (CS), and query decider time-complexity (QC). Thus, these dimensions can be altered with respect to the original, emulated state by the state storage size (SS). For example, enforcing polynomial storage (SSp) and linear-time command mapping (CCl) will guarantee a command mapping that is linear-time with respect to the *emulating state*, which is a polynomial expansion over the original *emulated state*.

8.6.2 Interpreting the Dimensions

We now discuss the practical impacts each identified dimensions, and what types of environments may cause one to prefer a particular property in these dimensions over others.

SC: State correspondence structure allows one to change what needs to be preserved about the state during a reduction. If the deployment scenario in question assumes only that the reduction allows the proper authorization requests, SCa should suffice. For scenarios that require the access control system to support (and provide correct answers to) additional queries such as, *"Is user u a member of role r?"*, SCq is more appropriate. Finally, in scenarios that make use of additional code that has access to (and assumes a particular arrangement of) the access control system's internal data structures, SCs is the best choice.

SS: State storage limits the size of the emulated state with respect to the original state (i.e., the state of the system being emulated). This can be restricted for several reasons. The most obvious is storage space: if trusted storage for representing access control state is limited, we may restrict the reduction from mapping states in a way that increases storage by more than a linear factor (SSl) or a polynomial factor (SSp). However, the more interesting reason

comes from an interaction described in Section 8.6.1. Since other dimensions place restrictions (e.g., on the number of commands executed) based on the size of the emulating state, we may restrict the state expansion to linear (SSl) in order, e.g., to restrict the command mapping procedure to be linear-time in the size of the *original, emulated state*. If state storage is polynomial (SSp), then even if we enforce a command mapping that is linear in the emulating state (CCl), this only restricts it to being polynomial-time with respect to the emulated state. Thus, even when trusted storage space is unbounded in the deployment scenario, one may desire to limit state size to limit later iteration over this state.

CD: Command mapping dependence allows one to require that the command mapping be computable without full knowledge and inspection of the state in which a command will be executed in. Independent command (CDi) requires that each command is mapped independent of the state, and is useful in deployment scenarios in which the agent calculating the emulating commands is completely unprivileged, and cannot inspect the state. It is also useful when commands must be precompiled, thus adding no computation at runtime beyond that of the emulating commands themselves. Theory-dependent command mapping (CDt) allows the command mapping to inspect the *theory* of the state (i.e., the answers to all queries). This property is useful in deployment scenarios in which the simulation agent is no more privileged than normal users—calculating the mapped commands requires only information available by asking queries. Finally, state-dependent command mapping (CDs) allows the command mapping to arbitrarily inspect the state, requiring a powerful simulation agent.

CC: Command mapping complexity restricts the time-complexity of the command mapping with respect to the size of the emulating state. Constant command mapping (CCc) can restrict the command mapping from taking any longer for larger states, and is thus appropriate when states can be large but mapping commands must always remain fast. Linear command mapping (CCl) prevents expensive nested loops over access control state as well as operations such as sorting, while still allowing more processing for larger states.

CS: Command stuttering restricts the number of emulating commands executed for each emulated command. Lock-step (CS1) reductions must execute no more than one emulating command per emulated command, and thus ensure there is no intermediate state

exposed to users. In deployment scenarios without the ability to force atomic execution of a sequence of commands (or without built-in data structure locking), this property is crucial to preventing the inspection of intermediate (potentially inconsistent) states. Constant step (CSc) reductions are allowed a constant number of commands for each emulated command, and are thus appropriate when the state can grow to be large but the deployment scenario requires that the number of steps for any emulated action remain bounded (e.g., to prevent starvation due to locked structures).

CT: Trace structure properties restrict the path that the emulating system can take during the simulation of a single command. Semantic lock-step (CT1, depicted in Fig. 30) provides the benefits of a lock-step reduction in a slightly relaxed way: a "setup" phase prepares for the transition by changing only internal data (i.e., while remaining equivalent to the start state), then the transition occurs to a state equivalent to the end state, and then the "cleanup" phase cleans up any unnecessary leftover data (again, while remaining equivalent to the same end state). This is particularly useful when lock-step is too strict, but the deployment scenario is sensitive to the exposure of intermediate states (since, in CT1, no states are exposed except those equivalent to the start and end states). Query monotonicity (CTq) ensures that no query changes its truth value except those that are required to change between the start and end state. This allows multiple steps, but ensures that intermediate states, while not corresponding with the start or end state, never answer any query in a way that neither the start nor end state would. This is useful in scenarios where intermediate states are undesirable, but users are not expected to execute more than a single query between "valid" states (and will thus never detect the inconsistency). Access monotonicity (CTa) is similar, but applies only to authorization requests, and is useful in scenarios where inconsistent states are not a danger as long as they do not wrongly allow or forbid a request. Finally, non-contamination (CTs) ensures that no two accesses are allowed in an intermediate state that are not *both* allowed in either the start or end state. Thus, the emulating system is restricted not only from allowing accesses forbidden in the emulated system, but also combinations of individually-allowed accesses that are never combined in the emulated system. This restriction is particularly useful in environments with operations that "swap" accesses from one subject or object to another, or where separation of privilege

is utilized.

CA: Actor preservation restricts which users can be invoked to emulate commands. Self-execution (CA$\top$) requires each emulating command be executed by the same user as the original, emulated command. This allows the emulation agent to be completely unprivileged, mapping commands as a service to the user, but without executing them with any privilege beyond the user's own. Administration-preservation (CAa) requires any non-administrative emulated command be mapped to a sequence of non-administrative commands (i.e., a command that does not invoke administrative privileges cannot be emulated by an administrative command). This corresponds to scenarios in which users will be expected to operate largely without administrative intervention. No restriction in this dimension means that the command mapping can return commands to be executed by any other user. This is most appropriate when the emulating agent is trusted to execute administrative actions on behalf of untrusted users, or when the commands returned can then be delegated to other users to be approved and executed.

QD: Query dependence, similar to command dependence, can restrict the query decider from using the full knowledge of the state in question when mapping queries to truth values. Independent query decider (QD1 and QDi) map each emulated query to its emulating queries, and its truth value is then determined by checking the values of these mapped queries in the emulating state. This is useful, as with commands, for precompilation, and restricts the emulation agent's role to be akin to a Karp reduction (i.e., it can only return the value of its emulating queries with no modifications) [51, p. 60]. Theory-dependent query decider (QDt) allows the inspection of the full theory of the state, and thus arbitrary computation over the values of all queries in the emulating system, corresponding to a query decider that is only privileged enough to ask queries, but not inspect state. Finally, state-dependent query decider (QDs) allows arbitrary inspection of the state, and thus may answer emulated queries using state that is usually unqueriable in the emulating system; this requires a powerful emulation agent that is trusted to view the full access control data structures.

QC: Query complexity restricts the time-complexity of the query decider with respect to the size of the emulating state, and is thus chosen for reasons analogous to command complexity (CC).

QP: Query preservation restricts certain queries to be mapped by a sort of "identity function." That is, certain emulated queries are mapped TRUE if and only if the query is also present in the emulating system, and the query returns TRUE in the current emulating state. This dimension of restrictions is strict in that it requires the emulating system to support all of the included queries of the emulated system, but ensures that the emulating system is used as per normal. If the deployment scenario is unable to cope with reduced throughput caused by the query decider during emulation, QPf also ensures that we can pipe emulated queries directly to the emulating system. Particularly useful is authorization preservation (QPa) when using a emulating system based on a model with formally-proven properties; Since all authorization requests must be mapped by the identity, the emulating system allows an emulated request exactly when it would natively allow the same request.

Finally, **R: reachability** specifies whether the emulating system should be restricted from entering a state that does not correspond to an emulated state. If the simulation agent is users' only interface to the deployed access control system, forward reachability (R$\rightarrow$) is sufficient. However, if users can access the emulating system's native commands, bireachability (R$\leftrightarrow$) ensures that the system cannot transition to a state that is inconsistent with the emulated system.

8.6.3 Studying Canonical Usages

Next, we use the above interpretation of our expressiveness reduction properties to guide a discussion about how each of the notions of reduction that we studied in Section 8.5 is used by its creators. In many cases, the definition for a particular notion of reduction is underconstrained, and the reductions written within the framework actually satisfy stronger properties than the defined lower bound. We refer to the set of properties that the authors seem to intend for a reduction to uphold as its *canonical usage*. In the case of Sandhu's reduction, the author recognizes that the given constructions are stronger than the definition, noting that formalizing the definition of the stronger reduction is beyond the scope of the work [101]. Here, we make conjectures regarding the decomposition of the canonical usage of these reductions. A lattice view of these conjectures is shown in Fig. 33, where $\overline{\mathcal{X}}$ indicates

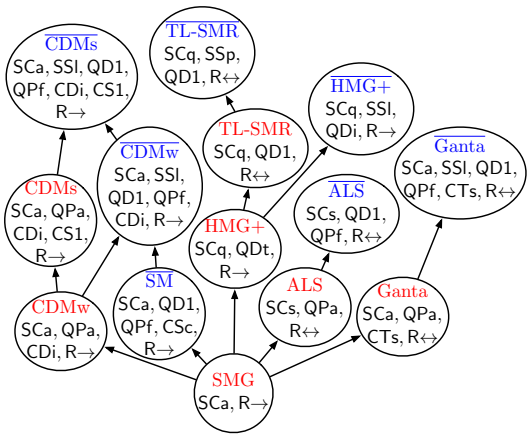


Figure 33: Partial lattice of canonical usage

the canonical usage of reduction type $\mathcal{X}$. For example, $\overline{\text{SM}}$ refers to the form of the SMG reduction used in [85, 107].

It is interesting to note that the relationships between notions of reduction are not necessarily preserved in the canonical usage. While SMG by definition is the weakest reduction, the canonical usage $\overline{\text{SM}}$ is incomparable to any reduction's definition and positioned strictly weaker than the canonical usage of the CDM reductions. While, by definition, the TL state-matching reduction is more strict than HMG+ parameterized expressiveness, their canonical usages are incomparable due to $\overline{\text{TL-SMR}}$ enforcing bireachability (R↔) and using polynomial state size (SSp), compared to $\overline{\text{HMG+}}$ enforcing forward reachability (R→) and using linear state size (SSI). Finally, we note that all of CDMs, CDMw, SMG, and ALS reductions are canonically used in such a way that enforces full query preservation (QPf); that is, all of the constructed mappings of these types use the identity mapping for all supported queries, despite the fact that none of them specifically require this by definition. This trend of a notion of reduction's usage being consistently more strict than its definition reveals the difficulty in fully specifying the set of properties that a notion of expressiveness reduction is intended to enforce. The discussion in this section, aimed at helping analysts choose a reasonable set of properties for a deployment, can also help ensure that newer notions of reduction are fully specified, and best match their intended usages.

8.7 SUMMARY AND FUTURE WORK

In this chapter, we organize the existing knowledge of expressiveness reductions by formalizing a granular, property-based representation, proposing a wide range of dimensions of reduction properties, and positioning influential notions of expressiveness reduction from the literature within the lattice of these properties. In doing so, we provide the first systematic comparison of existing reductions that were not previously known to be directly comparable, showing how these notions of expressiveness reduction relate to one another.

Looking away from existing notions of reduction and rather *between* them, this work allows us to explore an organized space of reductions to identify areas to explore in future research. For instance, knowing expressiveness results derived using the SMG and ALS reductions, which of these hold true for notions of reduction "between" the two existing notions? What results can be shown for a reduction decomposing to the union of the properties of two existing notions? How far up the lattice do all systems become incomparable? These questions can only be explored thanks to the systematic means of reduction decomposition.

Finally, understanding the systems implications of various reduction properties will enable analysts to select the notion of access control expressiveness that corresponds most closely to the scenario in which they plan to deploy the target access control system(s). Thus, we make inroads toward bringing expressiveness analysis techniques out of the strictly formal realm, and repurpose these techniques for use as fine-grained metrics of capability within the context of suitability analysis.

A question to be explored in future work is the identification of the set of analysis questions that a particular set of reduction properties preserve. For example, Tripunitara and Li showed that the state-matching reduction preserves *compositional security analysis instances*: the set of questions containing a single quantifier ($\exists$ or $\forall$), a propositional formula over queries φ, and a start state γ [114]. Semantically, the question asks whether φ it is {ever, always} true in states reachable from γ. If $\mathcal{T}$ admits a state-matching reduction of $\mathcal{S}$, then all compositional security analysis instances have the same value in $\mathcal{S}$ and $\mathcal{T}$. Identifying the types of analysis questions preserved by other notions of reduction would allow us even greater understanding of the practical and theoretical impacts of reduction property choices.

9.0 CASE STUDY: THE OTHER STATE-MATCHING REDUCTIONS

The state-matching reduction [113, 114] is a form of access control expressiveness reduction that is justified through the proof that it preserves all compositional security analysis instances. More precisely, a mapping from $\mathcal{S}$ to $\mathcal{T}$ is said to be *strongly security-preserving* when any compositional security analysis instance in $\mathcal{S}$ is true if and only if its image is true in $\mathcal{T}$. Further, a mapping is strongly security-preserving if and only if it is a state-matching reduction. In this chapter, we present several other related notions of reduction (both more and less strict), each of which can also be shown to preserve this exact type of security analysis question.

To show that the additional separations enabled by the properties lattice are meaningful, we then investigate these different notions of reduction from the perspective of the expressiveness results attained using each. We show that, although these types of reduction are indistinguishable from the perspective of preserving security analysis questions, they do not lead to the same main results as the original state-matching reduction. We thus showcase the importance of precisely specifying the type of reduction used in an analysis, further supporting the use of the properties lattice over static justifications such as preserving security analysis answers.

9.1 INTRODUCTION

The state-matching reduction [113, 114] is a form of access control expressiveness reduction that is thought to be more fundamentally interesting (or at least more principled) than earlier forms from the literature. This claim is primarily based on the evaluation from [114], in

which it is proven that the state-matching reduction preserves all compositional security analysis instances, a result that is used as the primary justification for the mapping. Here, we restate the definition of this security analysis question.

Definition 10 (Compositional Security Analysis, restated)**.** Given an access control system $\langle \Gamma, \Psi, Q \rangle$, a *compositional security analysis instance* has the form $\langle \gamma, \phi, \psi, \Pi \rangle$, where $\gamma \in \Gamma$ is a state, ϕ is a propositional formula over Q, $\psi \in \Psi$ is a state-transition rule, and $\Pi \in \{\forall, \exists\}$ is a quantifier.

An instance $\langle \gamma, \phi, \psi, \exists \rangle$ is said to be *existential*: it asks whether there exists state γ_1 such that $\gamma \overset{*}{\mapsto}_{\psi} \gamma_1$ and $\gamma_1 \vdash \phi$.

An instance $\langle \gamma, \phi, \psi, \forall \rangle$ is said to be *universal*: it asks whether for every state γ_1 such that $\gamma \overset{*}{\mapsto}_{\psi} \gamma_1$, $\gamma_1 \vdash \phi$. $\Diamond$

More precisely, a mapping from $\mathcal{S}$ to $\mathcal{T}$ is said to be *strongly security-preserving* when any compositional security analysis instance in $\mathcal{S}$ is true if and only if its image is true in $\mathcal{T}$, and a mapping is strongly security-preserving if and only if it is a state-matching reduction. We note that this does not take into account any properties of the analysis in question or the application for which it is being conducted. In contrast, our thesis statement motivates fully application-aware evaluation metrics, and as such we believe that relying on preserved analysis questions for identifying meaningful notions of reduction is too imprecise and fragile to changes in the application. We thus advocate instead for determining the best notion of reduction for a particular analysis by considering the lattice of properties presented in Chapter 8, particularly how each dimension affects the application under consideration.

In this chapter, we support this view by showing that the relationship between the state-matching reduction and the compositional security analysis instance is not fundamentally unique. We show corresponding proofs of preservation of these analysis questions by other, related notions of reduction (some more strict, and some less strict). We identify each of these notions of reduction using the lattice of properties presented in Chapter 8. We thus support our assertion that a modular lattice of reduction properties gives a more fine-grained tool for selection notions of reduction than previously existed.

To show that the additional separations enabled by the properties lattice are meaningful,

we then investigate the differences between these notions of reduction, and the effects they have on the expressiveness results attained. In doing so, we show within the context of the alternative reductions (which are indistinguishable from the original state-matching reduction from the perspective of preserving security analysis questions), several main conclusions proved in [114] are invalidated. We thus showcase the importance of precisely specifying the type of reduction used in an analysis, as such statements as, "TAM cannot emulate ATAM while preserving compositional security," are imprecise in a way that has a very real effect. We also argue that an analyst should not rely on the overly broad and static concept of preservation of security analysis questions when identifying a notion of reduction to use for an analysis and instead should carefully consider appropriate values in each of the dimensions of reduction properties.

In this chapter, we make the following contributions.

- Utilizing the space of implementable expressiveness reductions presented in Chapter 8, we propose several alternative reductions to the state-matching reduction, one more strict and two more relaxed. These variants highlight design decisions that were made by Tripunitara and Li with the state-matching reduction [114] that are not explicitly discussed, and which are made apparent by our lattice of reduction properties.
- We show that the state-matching reduction is not fundamentally unique in its preservation of compositional security properties. We propose an accompanying variant on *strong security preservation* for each of our alternative reductions, thus showing what it means to preserve compositional security analysis instances within the context of each. We then show that these variants are no less principled than the state-matching reduction, by showing that each is necessary and sufficient for its own corresponding version of strong security preservation (the same characteristic that previous work indicated as the defining strength of the state-matching reduction). This shows the relative imprecision of using preservation of security questions to evaluate a notion of reduction, and shows that our properties lattice is able to more precisely separate expressiveness metrics.
- To show that the separations we discuss are meaningful, we conduct a case study to investigate the effects on expressiveness results of different reductions that are equally capable of preserving the same security questions. We show that considering a slightly

different notion of reduction can yield vastly different expressiveness results, even if those perturbations that do not alter the security properties that are preserved. In particular, we show that within the context of the alternative reductions to the state-matching reduction (which also preserve compositional security), several main conclusions proved using the original are invalidated. We thus showcase the importance of precisely specifying a type of reduction used in an analysis, as such statements as, "TAM cannot emulate ATAM while preserving compositional security," are imprecise in a way that has a very real effect.

- We identify the reduction properties that seem to have the greatest impact on expressiveness results based on our case study and prior work. That is, we identify those dimensions of properties whose values have the greatest effect on whether a particular reduction is possible or not. We then discuss how each changes the meaning of an expressiveness result from the perspective of the application, highlighting the application-based issues that are ignored entirely when evaluating the state-matching reduction in an application-agnostic way (i.e., considering only preserved security questions).

The remainder of this chapter is structured as follows. In Section 9.2, we discuss the state-matching reduction, its justification, and the results proved using it. We propose our variants of the state-matching reduction using the implementable reductions properties lattice in Section 9.3, and show that each preserves compositional security in Section 9.4. In Section 9.5, we summarize the claims made in [113, 114] and the ways in which our results call them into question, particularly in light of the application-aware focus of this dissertation as a whole. In Section 9.6, we show that the effect of the imprecision of these claims is very impactful on the expressiveness results attained. We do so by evaluating several conclusions drawn from the use of the state-matching reduction, and showing that they are invalidated within the context of the variant reductions. We conclude with a summary of the presented results in Section 9.7, including a discussion of the significance of the dimensions in which the variants differ from the original.

9.2 THE STATE-MATCHING REDUCTION

Recall from Chapter 2 the motivation behind the state-matching reduction. Tripunitara and Li [113, 114] noted that many existing notions of reduction did not correspond to any security analysis questions, and thus did not make any particular security guarantees. They thus began by formalizing the security analysis questions that they felt a reduction should satisfy: compositional security analysis (Definition 10). Intuitively, this analysis question covers those that ask whether a certain set of access control queries will always, never, or sometimes become true in any reachable state. It is a generalization of simple safety analysis (Definition 3) that considers a propositional formula over queries and may use the $\forall$ quantifier rather than being restricted only to the $\exists$ quantifier.

To prove that a mapping *preserves* compositional security analysis instances, a formal notion of *preservation* of this analysis question was presented in the form of the strongly security-preserving mapping.

Definition 33 (Strongly security-preserving mapping)**.** Given a mapping σ from system $\mathcal{S}$ to $\mathcal{T}$, the image under σ of a compositional security analysis instance, $\langle \gamma^{\mathcal{S}}, \phi^{\mathcal{S}}, \psi^{\mathcal{S}}, \Pi \rangle$ in $\mathcal{S}$ is $\langle \gamma^{\mathcal{T}}, \phi^{\mathcal{T}}, \psi^{\mathcal{T}}, \Pi \rangle$, where $\langle \gamma^{\mathcal{T}}, \psi^{\mathcal{T}} \rangle = \sigma\big(\langle \gamma^{\mathcal{S}}, \psi^{\mathcal{S}} \rangle\big)$ and $\phi^{\mathcal{T}}$ is obtained by substituting every $\mathcal{S}$ query $q^{\mathcal{S}}$ in $\phi^{\mathcal{S}}$ with $\mathcal{T}$ query $\sigma\big(q^{\mathcal{S}}\big)$.

A mapping σ is said to be *strongly security-preserving* when every compositional security analysis instance in $\mathcal{S}$ is TRUE if and only if the image of the instance under σ is TRUE. $\Diamond$

The authors then present their notion of reduction that is tailor-made to preserve these types of analysis questions: the state-matching reduction.

Definition 11 (State-Matching Reduction, restated)**.** Given a mapping from $\mathcal{S}$ to $\mathcal{T}$, $\sigma : (\Gamma^{\mathcal{S}} \times \Psi^{\mathcal{S}}) \cup Q^{\mathcal{S}} \rightarrow (\Gamma^{\mathcal{T}} \times \Psi^{\mathcal{T}}) \cup Q^{\mathcal{T}}$, we say that two states $\gamma^{\mathcal{S}}$ and $\gamma^{\mathcal{T}}$ are *equivalent* under the mapping σ when, for every $q^{\mathcal{S}} \in Q^{\mathcal{S}}$, $\gamma^{\mathcal{S}} \vdash q^{\mathcal{S}}$ if and only if $\gamma^{\mathcal{T}} \vdash \sigma\big(q^{\mathcal{S}}\big)$. A mapping σ from $\mathcal{S}$ to $\mathcal{T}$ is said to be a *state-matching reduction* if, for every $\gamma^{\mathcal{S}} \in \Gamma^{\mathcal{S}}$ and every $\psi^{\mathcal{S}} \in \Psi^{\mathcal{S}}$, $\langle \gamma^{\mathcal{T}}, \psi^{\mathcal{T}} \rangle = \sigma\big(\langle \gamma^{\mathcal{S}}, \psi^{\mathcal{S}} \rangle\big)$ has the following two properties:

1. For every state $\gamma_1^{\mathcal{S}}$ in $\mathcal{S}$ such that $\gamma^{\mathcal{S}} \overset{*}{\mapsto}_{\psi^{\mathcal{S}}} \gamma_1^{\mathcal{S}}$, there exists a state $\gamma_1^{\mathcal{T}}$ such that $\gamma^{\mathcal{T}} \overset{*}{\mapsto}_{\psi^{\mathcal{T}}} \gamma_1^{\mathcal{T}}$, and $\gamma_1^{\mathcal{S}}$ and $\gamma_1^{\mathcal{T}}$ are equivalent under σ.

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

The state-matching reduction uses a state correspondence that considers the values of all possible queries. It maps each query $q^{\mathcal{S}}$ in $\mathcal{S}$ to a single query $q^{\mathcal{T}}$ in $\mathcal{T}$, and the reduction must then determine the value of $q^{\mathcal{S}}$ in any state in $\mathcal{T}$ by checking the value of $q^{\mathcal{T}}$. Reachability constraints ensure that $\mathcal{T}$ can reach a state corresponding to each reachable $\mathcal{S}$ state, and cannot reach any state that does not have a reachable corresponding state in $\mathcal{S}$. Thus, it is expressed as SCq QD1 R$\leftrightarrow$ within the reductions properties lattice, as proved in Theorem 31.

Tripunitara and Li then justify Definition 11 through the following theorem.

Theorem 22. *Given two access control systems $\mathcal{S}$ and $\mathcal{T}$, a mapping σ from $\mathcal{S}$ to $\mathcal{T}$ is strongly security-preserving if and only if σ is a state-matching reduction.*

The state-matching reduction, then, preserves compositional security analysis in the following sense. If there exists a state-matching reduction, σ, from $\mathcal{S}$ to $\mathcal{T}$, then any compositional security analysis instance $\langle \gamma, \phi, \psi, \Pi \rangle$ evaluates to TRUE in $\mathcal{S}$ if and only if its *image over* σ evaluates to TRUE in $\mathcal{T}$. The image of instance $\langle \gamma, \phi, \psi, \Pi \rangle$ over σ is $\langle \gamma', \phi', \psi', \Pi \rangle$, where $\langle \gamma', \psi' \rangle = \sigma(\langle \gamma, \psi \rangle)$ and ϕ' is obtained by substituting every query q in ϕ with $\sigma(q)$.

Tripunitara and Li then use the existence or non-existence of state-matching reductions between pairs of systems to make expressiveness statements, including several we will look at closely in Section 9.6. It is shown that there is no state-matching reduction from ATAM to TAM, formalizing the benefit of checking for the absence of rights in addition to the presence of rights. A state-matching reduction is constructed from a particular RBAC administrative scheme (AAR) to a trust management scheme (RT[$\cap$]). A state-matching reduction is also shown from a simple discretionary scheme, Strict DAC with Change of Ownership (SDCO), to RBAC with ARBAC97, the first evidence of RBAC's limited expressiveness compared to DAC.

Although it is certainly true that the state-matching reduction is a seminal notion of reduction, some of the conclusions drawn from these results can be questioned by using of the reduction properties lattice proposed in Chapter 8. In particular, the idea that the state-matching reduction is a fundamentally better choice of reduction for use in an analysis

is called into question, as we show that other reductions also satisfy the property of the state-matching reduction that is used as its main justification. Furthermore, the expressiveness results should be viewed through the specific lens of the state-matching reduction, since more lofty claims (e.g., "TAM cannot emulate ATAM while preserving compositional security") are overestimating the reach of what the state-matching reduction can prove.

9.3 VARIANTS OF THE STATE-MATCHING REDUCTION

In this section, we define several variants of the state-matching reduction by *perturbing* its properties along axes of the reduction properties lattice discussed in Chapter 8. These reductions will highlight design decisions implicitly made in the development of the state-matching reduction [114]; we will later exploit these assumptions to prove that this reduction is not unique in its preservation of compositional security properties.

To assist us in defining our alternative reductions, as well as make it clearer what they have in common, we first abstract the notion of the *reachability requirement* as it applies to these reductions.

Definition 34 (Reachability)**.** Given a mapping σ from $\mathcal{S}$ to $\mathcal{T}$ and a state equivalence relation $\sim : \Gamma^{\mathcal{S}} \times \Gamma^{\mathcal{T}}$, σ enforces the *reachability requirement* over $\sim$ if, for every $\gamma^{\mathcal{S}} \in \Gamma^{\mathcal{S}}$ and every $\psi^{\mathcal{S}} \in \Psi^{\mathcal{S}}$, $\langle \gamma^{\mathcal{T}}, \psi^{\mathcal{T}} \rangle = \sigma\big(\langle \gamma^{\mathcal{S}}, \psi^{\mathcal{S}} \rangle\big)$ has the following two properties:

1. For every state $\gamma_1^{\mathcal{S}}$ in $\mathcal{S}$ such that $\gamma^{\mathcal{S}} \overset{*}{\mapsto}_{\psi^{\mathcal{S}}} \gamma_1^{\mathcal{S}}$, there exists a state $\gamma_1^{\mathcal{T}}$ such that $\gamma^{\mathcal{T}} \overset{*}{\mapsto}_{\psi^{\mathcal{T}}} \gamma_1^{\mathcal{T}}$, and $\gamma_1^{\mathcal{S}} \sim \gamma_1^{\mathcal{T}}$.
2. For every state $\gamma_1^{\mathcal{T}}$ in $\mathcal{T}$ such that $\gamma^{\mathcal{T}} \overset{*}{\mapsto}_{\psi^{\mathcal{T}}} \gamma_1^{\mathcal{T}}$, there exists a state $\gamma_1^{\mathcal{S}}$ such that $\gamma^{\mathcal{S}} \overset{*}{\mapsto}_{\psi^{\mathcal{S}}} \gamma_1^{\mathcal{S}}$, and $\gamma_1^{\mathcal{S}} \sim \gamma_1^{\mathcal{T}}$. ◊

To demonstrate how this definition can be used, we restate the definition of state-matching reduction in terms of the reachability requirement.

Definition 11 (State-matching reduction, restated)**.** Given a mapping from $\mathcal{S}$ to $\mathcal{T}$ of the form $\sigma : (\Gamma^{\mathcal{S}} \times \Psi^{\mathcal{S}}) \cup Q^{\mathcal{S}} \to (\Gamma^{\mathcal{T}} \times \Psi^{\mathcal{T}}) \cup Q^{\mathcal{T}}$, we say that two states $\gamma^{\mathcal{S}}$ and $\gamma^{\mathcal{T}}$ are *equivalent*

Reduction	Decomposition
SMR	SCq QD1 R↔
SMR+1	SCq QPf QD1 R↔
SMR-1	SCq QDi R↔
SMR-2	SCq QDs R↔

Table 3: Decompositions of the state-matching reduction and its variants

under σ ($\gamma^{\mathcal{S}} \sim \gamma^{\mathcal{T}}$) when, for every $q^{\mathcal{S}} \in Q^{\mathcal{S}}$, $\gamma^{\mathcal{S}} \vdash q^{\mathcal{S}}$ if and only if $\gamma^{\mathcal{T}} \vdash \sigma\big(q^{\mathcal{S}}\big)$. Mapping σ is said to be a *state-matching reduction* if it preserves the reachability requirement over $\sim$. $\Diamond$

We then define a variant reduction that is *more* strict than the state-matching reduction. This reduction, which we call SMR+1, is defined as a state-matching reduction that additionally enforces the property of full query preservation (QPf). Intuitively, this reduction is more strict than the state-matching reduction in that it does not allow queries to be mapped at all (in contrast to the state-matching reduction's mapping from single $\mathcal{S}$ query to single $\mathcal{T}$ query). A summary of the reduction properties to which the SMR+1 (as well as the other reductions considered in this section) decomposes is shown in Table 3.

Definition 35 (SMR+1)**.** Given a mapping from $\mathcal{S}$ to $\mathcal{T}$ of the form $\sigma : \Gamma^{\mathcal{S}} \times \Psi^{\mathcal{S}} \to \Gamma^{\mathcal{T}} \times \Psi^{\mathcal{T}}$, we say that two states $\gamma^{\mathcal{S}}$ and $\gamma^{\mathcal{T}}$ are *equivalent* under σ ($\gamma^{\mathcal{S}} \sim \gamma^{\mathcal{T}}$) when, for every $q^{\mathcal{S}} \in Q^{\mathcal{S}}$, $\gamma^{\mathcal{S}} \vdash q^{\mathcal{S}}$ if and only if $\gamma^{\mathcal{T}} \vdash q^{\mathcal{S}}$. Mapping σ is said to be a *SMR+1* if it preserves the reachibility requirement over $\sim$. $\Diamond$

Next, we define reductions that are *less* strict than the state-matching reduction, in this case considering the dimension of query decider dependence (QD). The first allows each $\mathcal{S}$ query to map to a *propositional formula* over queries in $\mathcal{T}$ (in contrast with the state-matching reduction, which maps each $\mathcal{S}$ query to a single $\mathcal{T}$ query). That is, it weakens the state-matching reduction from enforcing QD1 to QDi; we call this reduction SMR-1.

Definition 36 (SMR-1)**.** Given a mapping from $\mathcal{S}$ to $\mathcal{T}$ of the form $\sigma : (\Gamma^{\mathcal{S}} \times \Psi^{\mathcal{S}}) \cup Q^{\mathcal{S}} \to (\Gamma^{\mathcal{T}} \times \Psi^{\mathcal{T}}) \cup \mathscr{L}\big(Q^{\mathcal{T}}\big)$, we say that two states $\gamma^{\mathcal{S}}$ and $\gamma^{\mathcal{T}}$ are *equivalent* under σ ($\gamma^{\mathcal{S}} \sim \gamma^{\mathcal{T}}$)

when, for every $q^{\mathcal{S}} \in Q^{\mathcal{S}}$, $\gamma^{\mathcal{S}} \vdash q^{\mathcal{S}}$ if and only if $\gamma^{\mathcal{T}} \vdash \sigma\big(q^{\mathcal{S}}\big)$. Mapping σ from $\mathcal{S}$ to $\mathcal{T}$ is said to be a *SMR-1* if it preserves the reachability requirement over $\sim$. $\Diamond$

Finally, our least strict reduction carries the trend of SMR-1 a step farther by weakening the QDi all the way to QDs. That is, rather than defining a function from $\mathcal{S}$ queries to (some form of) $\mathcal{T}$ queries, the SMR-2 defines a function whose domain is a $\mathcal{S}$ query and a $\mathcal{T}$ state and whose range is $\{\text{TRUE}, \text{FALSE}\}$. Then, $\mathcal{S}$ state $\gamma^{\mathcal{S}}$ corresponds to $\mathcal{T}$ state if and only if, for any $\mathcal{S}$ query $q \in Q^{\mathcal{S}}$, $\mathcal{S} \vdash q \iff \sigma\big(q, \gamma^{\mathcal{T}}\big) = \text{TRUE}$.

Definition 37 (SMR-2)**.** Given a mapping from $\mathcal{S}$ to $\mathcal{T}$ of the form $\sigma : (\Gamma^{\mathcal{S}} \times \Psi^{\mathcal{S}}) \cup (Q^{\mathcal{S}} \times \Gamma^{\mathcal{T}}) \rightarrow (\Gamma^{\mathcal{T}} \times \Psi^{\mathcal{T}}) \cup \{\text{TRUE}, \text{FALSE}\}$, we say that two states $\gamma^{\mathcal{S}}$ and $\gamma^{\mathcal{T}}$ are *equivalent* under σ ($\gamma^{\mathcal{S}} \sim \gamma^{\mathcal{T}}$) when, for every $q^{\mathcal{S}} \in Q^{\mathcal{S}}$, $\gamma^{\mathcal{S}} \vdash q^{\mathcal{S}}$ if and only if $\sigma\big(q^{\mathcal{S}}, \gamma^{\mathcal{T}}\big) = \text{TRUE}$. Mapping σ from $\mathcal{S}$ to $\mathcal{T}$ is said to be a *SMR-2* if it preserves the reachability requirement over $\sim$. $\Diamond$

Although the state-matching reduction is very intuitive, inspection of the above reductions shows that it is not fundamentally more or less valid than other alternatives; past work has used QPf [2, 3, 21, 41], while QDi is a natural generalization of QD1. Even QDs in SMR-2 is closely related to the form used in parameterized expressiveness [59].

Thus, intuitively, the state-matching reduction does not seem to be fundamentally more sound than the variants defined above. However, the state-matching reduction is evaluated by showing its preservation of compositional security properties. In order to formalize our claim that the state-matching reduction is not unique, we will next investigate the abilities of these variants to similarly preserve such properties.

9.4 PRESERVING COMPOSITIONAL SECURITY PROPERTIES

When proposed, the state-matching reduction was evaluated formally by showing that it preserves compositional security properties. In this section, we show that the same is true for the reductions presented in Section 9.3.

Tripinatara and Li prove that a mapping is strongly security-preserving if and only if it is a state-matching reduction [114]. However, although Definition 33 does encode a

particular method of preserving compositional security, it is far from the only way. A cursory inspection reveals that Definition 33 is closely tied to the form of mapping upon which the state-matching reduction is built. We therefore define SSP+1, SSP-1, and SSP-2, variants of strongly security-preserving mapping that share a corresponding relationship with our reduction variants SMR+1, SMR-1, and SMR-2, respectively. Note that none of these mappings changes the definition of compositional security analysis instance (Definition 10), only what it means to *preserve* those instances.

SSP+1 defines the image of a compositional security analysis instance to be more strict, corresponding to the jump from the state-matching reduction to SMR+1. That is, it eliminates the mapping's ability to substitute different $\mathcal{T}$ queries for $\mathcal{S}$ queries.

Definition 38 (SSP+1)**.** Given a mapping σ from system $\mathcal{S}$ to $\mathcal{T}$, the image under σ of a compositional security analysis instance, $\langle \gamma^{\mathcal{S}}, \phi^{\mathcal{S}}, \psi^{\mathcal{S}}, \Pi \rangle$ in $\mathcal{S}$ is $\langle \gamma^{\mathcal{T}}, \phi^{\mathcal{S}}, \psi^{\mathcal{T}}, \Pi \rangle$, where $\langle \gamma^{\mathcal{T}}, \psi^{\mathcal{T}} \rangle = \sigma\big(\langle \gamma^{\mathcal{S}}, \psi^{\mathcal{S}} \rangle\big)$.

A mapping σ is said to be *SSP+1* when every compositional security analysis instance in $\mathcal{S}$ is TRUE if and only if the image of the instance under σ is TRUE. $\Diamond$

Similarly, SSP-1 allows the mapping to map each $\mathcal{S}$ query to a propositional formula over $\mathcal{T}$ queries. Since ϕ in a compositional security analysis instance is already a propositional formula over queries, the resulting image of a compositional security analysis instance in $\mathcal{S}$ is still a compositional security analysis instance in $\mathcal{T}$ by the closure of propositional logic.

Definition 39 (SSP-1)**.** Given a mapping σ from system $\mathcal{S}$ to $\mathcal{T}$, the image under σ of a compositional security analysis instance, $\langle \gamma^{\mathcal{S}}, \phi^{\mathcal{S}}, \psi^{\mathcal{S}}, \Pi \rangle$ in $\mathcal{S}$ is $\langle \gamma^{\mathcal{T}}, \phi^{\mathcal{T}}, \psi^{\mathcal{T}}, \Pi \rangle$, where $\langle \gamma^{\mathcal{T}}, \psi^{\mathcal{T}} \rangle = \sigma\big(\langle \gamma^{\mathcal{S}}, \psi^{\mathcal{S}} \rangle\big)$ and $\phi^{\mathcal{T}}$ is obtained by substituting every $\mathcal{S}$ query $q^{\mathcal{S}}$ in $\phi^{\mathcal{S}}$ with the propositional formula over $\mathcal{T}$ queries, $\sigma\big(q^{\mathcal{S}}\big)$.

A mapping σ is said to be *SSP-1* when every compositional security analysis instance in $\mathcal{S}$ is TRUE if and only if the image of the instance under σ is TRUE. $\Diamond$

Finally, since the query decider in SMR-2 can arbitrarily inspect the state, SSP-2 does not define the image of a compositional security analysis instance at all. Instead, it maps the semantics of the compositional security analysis instance across the mapping, rather than preserving the original semantics and mapping the compositional security analysis instance

itself.

Definition 40 (SSP-2)**.** Given a mapping σ from system $\mathcal{S}$ to $\mathcal{T}$, σ is said to be *SSP-2* when the following conditions hold (let $\langle \gamma^{\mathcal{T}}, \psi^{\mathcal{T}} \rangle = \sigma\big(\langle \gamma^{\mathcal{S}}, \psi^{\mathcal{S}} \rangle\big)$):

1. Every existential compositional security analysis instance $\langle \gamma^{\mathcal{S}}, \phi^{\mathcal{S}}, \psi^{\mathcal{S}}, \exists \rangle$ is TRUE in $\mathcal{S}$ if and only if, there exists state $\gamma_1^{\mathcal{T}}$ such that $\gamma^{\mathcal{T}} \overset{*}{\mapsto}_{\psi^{\mathcal{T}}} \gamma_1^{\mathcal{T}}$ and $\sigma\big(\phi, \gamma_1^{\mathcal{T}}\big) = \text{TRUE}$.
2. Every universal compositional security analysis instance $\langle \gamma^{\mathcal{S}}, \phi^{\mathcal{S}}, \psi^{\mathcal{S}}, \forall \rangle$ is TRUE in $\mathcal{S}$ if and only if, for every state $\gamma_1^{\mathcal{T}}$ such that $\gamma^{\mathcal{T}} \overset{*}{\mapsto}_{\psi^{\mathcal{T}}} \gamma_1^{\mathcal{T}}$, $\sigma\big(\phi, \gamma_1^{\mathcal{T}}\big) = \text{TRUE}$. ◊

Note that here we use $\sigma(\phi, \gamma)$ to denote the result of evaluating ϕ with every query q substituted with $\sigma(q, \gamma)$.

Now, we show that each of our variant reductions is necessary and sufficient for its corresponding method of preserving compositional security properties.

Theorem 23. *Given two systems $\mathcal{S}$ and $\mathcal{T}$, a mapping σ from $\mathcal{S}$ to $\mathcal{T}$ is SSP+1 if and only if σ is a SMR+1.*

Proof. **The "if" direction** Given SMR+1 σ from $\mathcal{S}$ to $\mathcal{T}$ and compositional security analysis instance $\langle \gamma^{\mathcal{S}}, \phi, \psi^{\mathcal{S}}, \Pi \rangle$ in $\mathcal{S}$, let $\langle \gamma^{\mathcal{T}}, \psi^{\mathcal{T}} \rangle = \sigma\big(\langle \gamma^{\mathcal{S}}, \psi^{\mathcal{S}} \rangle\big)$. We show that $\langle \gamma^{\mathcal{S}}, \phi, \psi^{\mathcal{S}}, \Pi \rangle$ is true if and only if $\langle \gamma^{\mathcal{T}}, \phi, \psi^{\mathcal{T}}, \Pi \rangle$ is true, and thus that σ is SSP+1.

First, consider the case where $\Pi = \exists$ (existential instance). If $\langle \gamma^{\mathcal{S}}, \phi, \psi^{\mathcal{S}}, \Pi \rangle$ is true, then there exists a state $\gamma_1^{\mathcal{S}}$ that is $\psi^{\mathcal{S}}$-reachable from $\gamma^{\mathcal{S}}$ such that $\gamma_1^{\mathcal{S}} \vdash \phi$. By Definition 35 and Property 1 of Definition 34, there exists a state $\gamma_1^{\mathcal{T}}$ that is $\psi^{\mathcal{T}}$-reachable from $\gamma^{\mathcal{T}}$ such that $\gamma_1^{\mathcal{S}} \sim \gamma_1^{\mathcal{T}}$. By the definition of $\sim$ in Definition 35, $\gamma_1^{\mathcal{T}} \vdash \phi$, and thus $\langle \gamma^{\mathcal{T}}, \phi, \psi^{\mathcal{T}}, \Pi \rangle$ is true.

On the other hand, if $\langle \gamma^{\mathcal{T}}, \phi, \psi^{\mathcal{T}}, \Pi \rangle$ is true, then there exists a state $\gamma_1^{\mathcal{T}}$ that is $\psi^{\mathcal{T}}$-reachable from $\gamma^{\mathcal{T}}$ such that $\gamma_1^{\mathcal{T}} \vdash \phi$. By Definition 35 and Property 2 of Definition 34, there exists a state $\gamma_1^{\mathcal{S}}$ that is $\psi^{\mathcal{S}}$-reachable from $\gamma^{\mathcal{S}}$ such that $\gamma_1^{\mathcal{S}} \sim \gamma_1^{\mathcal{T}}$. By the definition of $\sim$ in Definition 35, $\gamma_1^{\mathcal{S}} \vdash \phi$, and thus $\langle \gamma^{\mathcal{S}}, \phi, \psi^{\mathcal{S}}, \Pi \rangle$ is true.

Next, consider the case where $\Pi = \forall$ (universal instance). If $\langle \gamma^{\mathcal{S}}, \phi, \psi^{\mathcal{S}}, \Pi \rangle$ is false, then there exists a state $\gamma_1^{\mathcal{S}}$ that is $\psi^{\mathcal{S}}$-reachable from $\gamma^{\mathcal{S}}$ such that $\gamma_1^{\mathcal{S}} \nvdash \phi$. By Definition 35 and Property 1 of Definition 34, there exists a state $\gamma_1^{\mathcal{T}}$ that is $\psi^{\mathcal{T}}$-reachable from $\gamma^{\mathcal{T}}$ such that $\gamma_1^{\mathcal{S}} \sim \gamma_1^{\mathcal{T}}$. By the definition of $\sim$ in Definition 35, $\gamma_1^{\mathcal{T}} \nvdash \phi$, and thus $\langle \gamma^{\mathcal{T}}, \phi, \psi^{\mathcal{T}}, \Pi \rangle$ is false.

On the other hand, if $\langle \gamma^{\mathcal{T}}, \phi, \psi^{\mathcal{T}}, \Pi \rangle$ is false, then there exists a state $\gamma_1^{\mathcal{T}}$ that is $\psi^{\mathcal{T}}$-reachable from $\gamma^{\mathcal{T}}$ such that $\gamma_1^{\mathcal{T}} \nvdash \phi$. By Definition 35 and Property 2 of Definition 34, there exists a state $\gamma_1^{\mathcal{S}}$ that is $\psi^{\mathcal{S}}$-reachable from $\gamma^{\mathcal{S}}$ such that $\gamma_1^{\mathcal{S}} \sim \gamma_1^{\mathcal{T}}$. By the definition of $\sim$ in Definition 35, $\gamma_1^{\mathcal{S}} \nvdash \phi$, and thus $\langle \gamma^{\mathcal{S}}, \phi, \psi^{\mathcal{S}}, \Pi \rangle$ is false.

The "only if" direction Assume a mapping is not a SMR+1, and thus there is a violation of one of the two properties of SMR+1. We show that this leads to a compositional security analysis instance that is not (SSP+1) preserved, and thus the mapping is not SSP+1.

Given mapping σ from $\mathcal{S}$ to $\mathcal{T}$, assume σ is not a SMR+1. Thus, there exists $\gamma^{\mathcal{S}} \in \Gamma^{\mathcal{S}}$ and $\psi^{\mathcal{S}} \in \Psi^{\mathcal{S}}$ such that $\langle \gamma^{\mathcal{T}}, \psi^{\mathcal{T}} \rangle = \sigma\big(\langle \gamma^{\mathcal{S}}, \psi^{\mathcal{S}} \rangle\big)$ violates one of the reachability requirements over $\sim$. We show that σ does not preserve some compositional security analysis instance $\langle \gamma^{\mathcal{S}}, \phi, \psi^{\mathcal{S}}, \Pi \rangle$, and thus that it is not SSP+1.

First, consider the case where Property 1 is violated. Thus, there exists a state $\gamma_1^{\mathcal{S}}$ reachable from $\gamma^{\mathcal{S}}$ such that no state $\gamma_1^{\mathcal{T}}$ reachable from $\gamma^{\mathcal{T}}$ satisfies $\gamma_1^{\mathcal{S}} \sim \gamma_1^{\mathcal{T}}$. Construct ϕ as a long conjunction such that, for every query $q^{\mathcal{S}} \in Q^{\mathcal{S}}$, ϕ includes $q^{\mathcal{S}}$ if $\gamma_1^{\mathcal{S}} \vdash q^{\mathcal{S}}$, and $\neg q^{\mathcal{S}}$ if $\gamma_1^{\mathcal{S}} \nvdash q^{\mathcal{S}}$. It is clear that ϕ is true in $\gamma_1^{\mathcal{S}}$, but since no state $\gamma_1^{\mathcal{T}}$ reachable from $\gamma^{\mathcal{T}}$ satisfies $\gamma_1^{\mathcal{S}} \sim \gamma_1^{\mathcal{T}}$, ϕ is false in all states reachable from $\gamma^{\mathcal{T}}$. Thus, compositional security analysis instance $\langle \gamma^{\mathcal{S}}, \phi, \psi^{\mathcal{S}}, \exists \rangle$ is true but mapped compositional security analysis instance $\langle \gamma^{\mathcal{T}}, \phi, \psi^{\mathcal{T}}, \exists \rangle$ is false, and therefore σ is not SSP+1.

Next, consider the case where Property 2 is violated. Thus, there exists a state $\gamma_1^{\mathcal{T}}$ reachable from $\gamma^{\mathcal{T}}$ such that no state $\gamma_1^{\mathcal{S}}$ reachable from $\gamma^{\mathcal{S}}$ satisfies $\gamma_1^{\mathcal{S}} \sim \gamma_1^{\mathcal{T}}$. Construct ϕ as a long conjunction such that, for every query $q^{\mathcal{S}} \in Q^{\mathcal{S}}$, ϕ includes $q^{\mathcal{S}}$ if $\gamma_1^{\mathcal{T}} \vdash q^{\mathcal{S}}$, and $\neg q^{\mathcal{S}}$ if $\gamma_1^{\mathcal{T}} \nvdash q^{\mathcal{S}}$. It is clear that ϕ is true in $\gamma_1^{\mathcal{T}}$, but since no state $\gamma_1^{\mathcal{S}}$ reachable from $\gamma^{\mathcal{S}}$ satisfies $\gamma_1^{\mathcal{S}} \sim \gamma_1^{\mathcal{T}}$, ϕ is false in all states reachable from $\gamma^{\mathcal{S}}$. Thus, compositional security analysis instance $\langle \gamma^{\mathcal{S}}, \phi, \psi^{\mathcal{S}}, \exists \rangle$ is false but mapped compositional security analysis instance $\langle \gamma^{\mathcal{T}}, \phi, \psi^{\mathcal{T}}, \exists \rangle$ is true, and therefore σ is not SSP+1. □

Theorem 24. *Given two systems $\mathcal{S}$ and $\mathcal{T}$, a mapping σ from $\mathcal{S}$ to $\mathcal{T}$ is SSP-1 if and only if σ is a SMR-1.*

Proof. **The "if" direction** Given SMR-1 σ from $\mathcal{S}$ to $\mathcal{T}$ and compositional security analysis instance $\langle \gamma^{\mathcal{S}}, \phi^{\mathcal{S}}, \psi^{\mathcal{S}}, \Pi \rangle$ in $\mathcal{S}$, let $\langle \gamma^{\mathcal{T}}, \psi^{\mathcal{T}} \rangle = \sigma\big(\langle \gamma^{\mathcal{S}}, \psi^{\mathcal{S}} \rangle\big)$ and $\phi^{\mathcal{T}} = \sigma\big(\phi^{\mathcal{S}}\big)$. We show that

$\langle \gamma^{\mathcal{S}}, \phi^{\mathcal{S}}, \psi^{\mathcal{S}}, \Pi \rangle$ is true if and only if $\langle \gamma^{\mathcal{T}}, \phi^{\mathcal{T}}, \psi^{\mathcal{T}}, \Pi \rangle$ is true, and thus that σ is SSP-1.

First, consider the case where $\Pi = \exists$ (existential instance). If $\langle \gamma^{\mathcal{S}}, \phi^{\mathcal{S}}, \psi^{\mathcal{S}}, \Pi \rangle$ is true, then there exists a state $\gamma_1^{\mathcal{S}}$ that is $\psi^{\mathcal{S}}$-reachable from $\gamma^{\mathcal{S}}$ such that $\gamma_1^{\mathcal{S}} \vdash \phi^{\mathcal{S}}$. By Definition 36 and Property 1 of Definition 34, there exists a state $\gamma_1^{\mathcal{T}}$ that is $\psi^{\mathcal{T}}$-reachable from $\gamma^{\mathcal{T}}$ such that $\gamma_1^{\mathcal{S}} \sim \gamma_1^{\mathcal{T}}$. By the definition of $\sim$ in Definition 36, $\gamma_1^{\mathcal{T}} \vdash \phi^{\mathcal{T}}$, and thus $\langle \gamma^{\mathcal{T}}, \phi^{\mathcal{T}}, \psi^{\mathcal{T}}, \Pi \rangle$ is true.

On the other hand, if $\langle \gamma^{\mathcal{T}}, \phi^{\mathcal{T}}, \psi^{\mathcal{T}}, \Pi \rangle$ is true, then there exists a state $\gamma_1^{\mathcal{T}}$ that is $\psi^{\mathcal{T}}$-reachable from $\gamma^{\mathcal{T}}$ such that $\gamma_1^{\mathcal{T}} \vdash \phi^{\mathcal{T}}$. By Definition 36 and Property 2 of Definition 34, there exists a state $\gamma_1^{\mathcal{S}}$ that is $\psi^{\mathcal{S}}$-reachable from $\gamma^{\mathcal{S}}$ such that $\gamma_1^{\mathcal{S}} \sim \gamma_1^{\mathcal{T}}$. By the definition of $\sim$ in Definition 36, $\gamma_1^{\mathcal{S}} \vdash \phi^{\mathcal{S}}$, and thus $\langle \gamma^{\mathcal{S}}, \phi^{\mathcal{S}}, \psi^{\mathcal{S}}, \Pi \rangle$ is true.

Next, consider the case where $\Pi = \forall$ (universal instance). If $\langle \gamma^{\mathcal{S}}, \phi^{\mathcal{S}}, \psi^{\mathcal{S}}, \Pi \rangle$ is false, then there exists a state $\gamma_1^{\mathcal{S}}$ that is $\psi^{\mathcal{S}}$-reachable from $\gamma^{\mathcal{S}}$ such that $\gamma_1^{\mathcal{S}} \nvdash \phi^{\mathcal{S}}$. By Definition 36 and Property 1 of Definition 34, there exists a state $\gamma_1^{\mathcal{T}}$ that is $\psi^{\mathcal{T}}$-reachable from $\gamma^{\mathcal{T}}$ such that $\gamma_1^{\mathcal{S}} \sim \gamma_1^{\mathcal{T}}$. By the definition of $\sim$ in Definition 36, $\gamma_1^{\mathcal{T}} \nvdash \phi^{\mathcal{T}}$, and thus $\langle \gamma^{\mathcal{T}}, \phi^{\mathcal{T}}, \psi^{\mathcal{T}}, \Pi \rangle$ is false.

On the other hand, if $\langle \gamma^{\mathcal{T}}, \phi^{\mathcal{T}}, \psi^{\mathcal{T}}, \Pi \rangle$ is false, then there exists a state $\gamma_1^{\mathcal{T}}$ that is $\psi^{\mathcal{T}}$-reachable from $\gamma^{\mathcal{T}}$ such that $\gamma_1^{\mathcal{T}} \nvdash \phi^{\mathcal{T}}$. By Definition 36 and Property 2 of Definition 34, there exists a state $\gamma_1^{\mathcal{S}}$ that is $\psi^{\mathcal{S}}$-reachable from $\gamma^{\mathcal{S}}$ such that $\gamma_1^{\mathcal{S}} \sim \gamma_1^{\mathcal{T}}$. By the definition of $\sim$ in Definition 36, $\gamma_1^{\mathcal{S}} \nvdash \phi^{\mathcal{S}}$, and thus $\langle \gamma^{\mathcal{S}}, \phi^{\mathcal{S}}, \psi^{\mathcal{S}}, \Pi \rangle$ is false.

The "only if" direction Assume a mapping is not a SMR-1, and thus there is a violation of one of the two properties of SMR-1. We show that this leads to a compositional security analysis instance that is not (SSP-1) preserved, and thus the mapping is not SSP-1.

Given mapping σ from $\mathcal{S}$ to $\mathcal{T}$, assume σ is not a SMR-1. Thus, there exists $\gamma^{\mathcal{S}} \in \Gamma^{\mathcal{S}}$ and $\psi^{\mathcal{S}} \in \Psi^{\mathcal{S}}$ such that $\langle \gamma^{\mathcal{T}}, \psi^{\mathcal{T}} \rangle = \sigma\big(\langle \gamma^{\mathcal{S}}, \psi^{\mathcal{S}} \rangle\big)$ violates one of the reachability requirements over $\sim$. We show that σ does not preserve some compositional security analysis instance $\langle \gamma^{\mathcal{S}}, \phi^{\mathcal{S}}, \psi^{\mathcal{S}}, \Pi \rangle$, and thus that it is not SSP-1.

First, consider the case where Property 1 is violated. Thus, there exists a state $\gamma_1^{\mathcal{S}}$ reachable from $\gamma^{\mathcal{S}}$ such that no state $\gamma_1^{\mathcal{T}}$ reachable from $\gamma^{\mathcal{T}}$ satisfies $\gamma_1^{\mathcal{S}} \sim \gamma_1^{\mathcal{T}}$. Construct $\phi^{\mathcal{S}}$ as a long conjunction such that, for every query $q^{\mathcal{S}} \in Q^{\mathcal{S}}$, $\phi^{\mathcal{S}}$ includes $q^{\mathcal{S}}$ if $\gamma_1^{\mathcal{S}} \vdash q^{\mathcal{S}}$, and $\neg q^{\mathcal{S}}$ if $\gamma_1^{\mathcal{S}} \nvdash q^{\mathcal{S}}$. It is clear that $\phi^{\mathcal{S}}$ is true in $\gamma_1^{\mathcal{S}}$, but since no state $\gamma_1^{\mathcal{T}}$ reachable from $\gamma^{\mathcal{T}}$ satisfies $\gamma_1^{\mathcal{S}} \sim \gamma_1^{\mathcal{T}}$, $\sigma\big(\phi^{\mathcal{S}}\big)$ is false in all states reachable from $\gamma^{\mathcal{T}}$. Thus, compositional security analysis instance $\langle \gamma^{\mathcal{S}}, \phi^{\mathcal{S}}, \psi^{\mathcal{S}}, \exists \rangle$ is true but mapped compositional security analysis instance

$\langle \gamma^{\mathcal{T}}, \sigma(\phi^{\mathcal{S}}), \psi^{\mathcal{T}}, \exists \rangle$ is false, and therefore σ is not SSP-1.

Next, consider the case where Property 2 is violated. Thus, there exists a state $\gamma_1^{\mathcal{T}}$ reachable from $\gamma^{\mathcal{T}}$ such that no state $\gamma_1^{\mathcal{S}}$ reachable from $\gamma^{\mathcal{S}}$ satisfies $\gamma_1^{\mathcal{S}} \sim \gamma_1^{\mathcal{T}}$. Construct $\phi^{\mathcal{S}}$ as a long conjunction such that, for every query $q^{\mathcal{S}} \in Q^{\mathcal{S}}$, $\phi^{\mathcal{S}}$ includes $q^{\mathcal{S}}$ if $\gamma_1^{\mathcal{T}} \vdash \sigma(q^{\mathcal{S}})$, and $\neg q^{\mathcal{S}}$ if $\gamma_1^{\mathcal{T}} \nvdash \sigma(q^{\mathcal{S}})$. It is clear that $\sigma(\phi^{\mathcal{S}})$ is true in $\gamma_1^{\mathcal{T}}$, but since no state $\gamma_1^{\mathcal{S}}$ reachable from $\gamma^{\mathcal{S}}$ satisfies $\gamma_1^{\mathcal{S}} \sim \gamma_1^{\mathcal{T}}$, $\phi^{\mathcal{S}}$ is false in all states reachable from $\gamma^{\mathcal{S}}$. Thus, compositional security analysis instance $\langle \gamma^{\mathcal{S}}, \phi^{\mathcal{S}}, \psi^{\mathcal{S}}, \exists \rangle$ is false but mapped compositional security analysis instance $\langle \gamma^{\mathcal{T}}, \sigma(\phi^{\mathcal{S}}), \psi^{\mathcal{T}}, \exists \rangle$ is true, and therefore σ is not SSP-1. □

Theorem 25. *Given two systems $\mathcal{S}$ and $\mathcal{T}$, a mapping σ from $\mathcal{S}$ to $\mathcal{T}$ is SSP-2 if and only if σ is a SMR-2.*

Proof. **The "if" direction** Given SMR-2 σ from $\mathcal{S}$ to $\mathcal{T}$ and compositional security analysis instance $\langle \gamma^{\mathcal{S}}, \phi, \psi^{\mathcal{S}}, \Pi \rangle$ in $\mathcal{S}$, let $\langle \gamma^{\mathcal{T}}, \psi^{\mathcal{T}} \rangle = \sigma(\langle \gamma^{\mathcal{S}}, \psi^{\mathcal{S}} \rangle)$. We show that σ is SSP-2.

First, consider the case where $\Pi = \exists$ (existential instance). If $\langle \gamma^{\mathcal{S}}, \phi, \psi^{\mathcal{S}}, \Pi \rangle$ is true, then there exists a state $\gamma_1^{\mathcal{S}}$ that is $\psi^{\mathcal{S}}$-reachable from $\gamma^{\mathcal{S}}$ such that $\gamma_1^{\mathcal{S}} \vdash \phi$. By Definition 37 and Property 1 of Definition 34, there exists a state $\gamma_1^{\mathcal{T}}$ that is $\psi^{\mathcal{T}}$-reachable from $\gamma^{\mathcal{T}}$ such that $\gamma_1^{\mathcal{S}} \sim \gamma_1^{\mathcal{T}}$. By the definition of $\sim$ in Definition 37, $\sigma(\phi, \gamma_1^{\mathcal{T}})$ is true.

On the other hand, assume that there exists a state $\gamma_1^{\mathcal{T}}$ that is $\psi^{\mathcal{T}}$-reachable from $\gamma^{\mathcal{T}}$ such that $\sigma(\phi, \gamma_1^{\mathcal{T}})$ is true. By Definition 37 and Property 2 of Definition 34, there exists a state $\gamma_1^{\mathcal{S}}$ that is $\psi^{\mathcal{S}}$-reachable from $\gamma^{\mathcal{S}}$ such that $\gamma_1^{\mathcal{S}} \sim \gamma_1^{\mathcal{T}}$. By the definition of $\sim$ in Definition 37, $\gamma_1^{\mathcal{S}} \vdash \phi$, and thus $\langle \gamma^{\mathcal{S}}, \phi, \psi^{\mathcal{S}}, \Pi \rangle$ is true.

Next, consider the case where $\Pi = \forall$ (universal instance). If $\langle \gamma^{\mathcal{S}}, \phi, \psi^{\mathcal{S}}, \Pi \rangle$ is false, then there exists a state $\gamma_1^{\mathcal{S}}$ that is $\psi^{\mathcal{S}}$-reachable from $\gamma^{\mathcal{S}}$ such that $\gamma_1^{\mathcal{S}} \nvdash \phi$. By Definition 37 and Property 1 of Definition 34, there exists a state $\gamma_1^{\mathcal{T}}$ that is $\psi^{\mathcal{T}}$-reachable from $\gamma^{\mathcal{T}}$ such that $\gamma_1^{\mathcal{S}} \sim \gamma_1^{\mathcal{T}}$. By the definition of $\sim$ in Definition 37, $\sigma(\phi, \gamma_1^{\mathcal{T}})$ is false.

On the other hand, assume that there exists a state $\gamma_1^{\mathcal{T}}$ that is $\psi^{\mathcal{T}}$-reachable from $\gamma^{\mathcal{T}}$ such that $\sigma(\phi, \gamma_1^{\mathcal{T}})$ is false. By Definition 37 and Property 2 of Definition 34, there exists a state $\gamma_1^{\mathcal{S}}$ that is $\psi^{\mathcal{S}}$-reachable from $\gamma^{\mathcal{S}}$ such that $\gamma_1^{\mathcal{S}} \sim \gamma_1^{\mathcal{T}}$. By the definition of $\sim$ in Definition 37, $\gamma_1^{\mathcal{S}} \nvdash \phi$, and thus $\langle \gamma^{\mathcal{S}}, \phi, \psi^{\mathcal{S}}, \Pi \rangle$ is false.

The "only if" direction Assume a mapping is not a SMR-2, and thus there is a

violation of one of the two properties of SMR-2. Show that this leads to a compositional security analysis instance that is not (SSP-2) preserved, and thus the mapping is not SSP-2.

Given mapping σ from $\mathcal{S}$ to $\mathcal{T}$, assume σ is not a SMR-2. Thus, there exists $\gamma^{\mathcal{S}} \in \Gamma^{\mathcal{S}}$ and $\psi^{\mathcal{S}} \in \Psi^{\mathcal{S}}$ such that $\langle \gamma^{\mathcal{T}}, \psi^{\mathcal{T}} \rangle = \sigma\big(\langle \gamma^{\mathcal{S}}, \psi^{\mathcal{S}} \rangle\big)$ violates one of the reachability requirements over $\sim$. We show that σ does not preserve some compositional security analysis instance $\langle \gamma^{\mathcal{S}}, \phi, \psi^{\mathcal{S}}, \Pi \rangle$, and thus that it is not SSP-2.

First, consider the case where Property 1 is violated. Thus, there exists a state $\gamma_1^{\mathcal{S}}$ reachable from $\gamma^{\mathcal{S}}$ such that no state $\gamma_1^{\mathcal{T}}$ reachable from $\gamma^{\mathcal{T}}$ satisfies $\gamma_1^{\mathcal{S}} \sim \gamma_1^{\mathcal{T}}$. Construct ϕ as a long conjunction such that, for every query $q^{\mathcal{S}} \in Q^{\mathcal{S}}$, ϕ includes $q^{\mathcal{S}}$ if $\gamma_1^{\mathcal{S}} \vdash q^{\mathcal{S}}$, and $\neg q^{\mathcal{S}}$ if $\gamma_1^{\mathcal{S}} \nvdash q^{\mathcal{S}}$. It is clear that ϕ is true in $\gamma_1^{\mathcal{S}}$, but since no state $\gamma_1^{\mathcal{T}}$ reachable from $\gamma^{\mathcal{T}}$ satisfies $\gamma_1^{\mathcal{S}} \sim \gamma_1^{\mathcal{T}}$, $\sigma\big(\phi, \gamma_1^{\mathcal{T}}\big)$ is false for all states $\gamma_1^{\mathcal{T}}$ reachable from $\gamma^{\mathcal{T}}$. Thus, compositional security analysis instance $\langle \gamma^{\mathcal{S}}, \phi, \psi^{\mathcal{S}}, \exists \rangle$ is true but $\forall \gamma_1^{\mathcal{T}}$ such that $\gamma^{\mathcal{T}} \overset{*}{\mapsto}_{\psi^{\mathcal{T}}} \gamma_1^{\mathcal{T}}$, $\sigma\big(\phi, \gamma_1^{\mathcal{T}}\big) = \textsc{false}$, and therefore σ is not SSP-2.

Next, consider the case where Property 2 is violated. Thus, there exists a state $\gamma_1^{\mathcal{T}}$ reachable from $\gamma^{\mathcal{T}}$ such that no state $\gamma_1^{\mathcal{S}}$ reachable from $\gamma^{\mathcal{S}}$ satisfies $\gamma_1^{\mathcal{S}} \sim \gamma_1^{\mathcal{T}}$. Construct ϕ as a long conjunction such that, for every query $q^{\mathcal{S}} \in Q^{\mathcal{S}}$, ϕ includes $q^{\mathcal{S}}$ if $\sigma\big(q^{\mathcal{S}}, \gamma_1^{\mathcal{T}}\big)$ is true, $\neg q^{\mathcal{S}}$ if $\sigma\big(q^{\mathcal{S}}, \gamma_1^{\mathcal{T}}\big)$ is false. It is clear that $\sigma\big(\phi, \gamma_1^{\mathcal{T}}\big)$ is true, but since no state $\gamma_1^{\mathcal{S}}$ reachable from $\gamma^{\mathcal{S}}$ satisfies $\gamma_1^{\mathcal{S}} \sim \gamma_1^{\mathcal{T}}$, ϕ is false in all states reachable from $\gamma^{\mathcal{S}}$. Thus, compositional security analysis instance $\langle \gamma^{\mathcal{S}}, \phi, \psi^{\mathcal{S}}, \exists \rangle$ is false but there exists state $\gamma_1^{\mathcal{T}}$ such that $\gamma^{\mathcal{T}} \overset{*}{\mapsto}_{\psi^{\mathcal{T}}} \gamma_1^{\mathcal{T}}$ and $\sigma\big(\phi, \gamma_1^{\mathcal{T}}\big) = \textsc{true}$, and therefore σ is not SSP-2. □

9.5 OVERVIEW OF REDUCTION FINDINGS

In this section, we overview the results of our reductions as they relate to the results presented by Tripunitara and Li [113, 114].

Primarily, we show that the justification of the state-matching reduction is incomplete. Consider the following quote from [114].

With the following theorem [(Theorem 22)], we justify [Definition 11, the definition

of the state-matching reduction].

What have shown is that this justification is not unique: we can just as easily use it to justify variants of reduction, both more and less strict.

Further, consider the motivation for the approach.

> *Our approach is to first identify the desirable and intuitive properties one would like simulations to have and then come up with the conditions on simulations that are both sufficient and necessary to satisfy those properties.*

It is intuitively true that preserving security analysis questions is desirable, and the proof that the state-matching reduction satisfies that requirement in a particular way is correct. However, the step that is missing in this approach is justifying why the state-matching reduction satisfies the properties in the *right* way, i.e., why this is the best way to preserve compositional security, given that there are many others. Similarly, the following statement is true, but only for a particular notion of "preservation" of compositional security properties that is specific to the state matching reduction.

> *We show that state-matching reductions are necessary and sufficient for preserving compositional security properties...*

We note that we do not seek to delegitimize the state-matching reduction or the work of Tripunitara and Li, but merely to point out that, given the development of techniques since the publication of [114], the arguments are incomplete, and a full argument in favor of a particular notion of expressiveness reduction needs to take into account additional information about its properties. That is, the preservation of security analysis questions, while clearly a desirable property, is not precise enough to justify a form of reduction alone. In addition, we argue that the final decision for a particular analysis on, e.g., whether we should use SMR-1 or the state-matching reduction, needs to take into account the application that the analysis is meant to consider.

9.6 EXPRESSIVENESS RESULTS USING VARIANTS OF THE STATE-MATCHING REDUCTION

In this section, we investigate the effects of perturbing reduction properties on the results of expressiveness results. First, we consider the comparison of TAM to ATAM. TAM, the typed access matrix, is similar to HRU (see Section 2.2) except with added state to track the type of each subject and object, which cannot change after creation. This allows the parameters to commands to have a specified type. ATAM, the augmented typed access matrix, extends TAM with the capability to check for the absence of rights in addition to checking for the presence of rights.

Work using early forms of reduction showed that TAM was as expressive as ATAM, and hence that checking for the absence of rights was theoretically unnecessary [105]. However, with respect to the state-matching reduction, ATAM is strictly more expressive than TAM (i.e., there does not exist a state-matching reduciton from ATAM to TAM, but there is a trivial state-matching reduction from TAM to ATAM) [114].

The proof of the latter relies on a counting argument. Since TAM cannot check for the absence of rights directly, a state-matching reduction from ATAM to TAM would have to emulate such queries using presence queries. Observe that a TAM command can enter a bounded number of rights in cells in the matrix at once, but a state can contain an unbounded number of objects. Thus, adding a new subject would require changing an unbounded number of absence queries to true (one for each object), which cannot be emulated in TAM without passing through an invalid state.

However, when considering an SMR-1 reduction, we can represent an ATAM query using a propositional formula over TAM queries. We can achieve an SMR-1 from ATAM to TAM by using the following general strategy.

$$r \notin M[s, o] \rightarrow \neg(r \in M[s, o]) \wedge s \in S \wedge o \in O \tag{9.1}$$

Theorem 26. *While there does not exist a state-matching reduction from ATAM to TAM, there exists a SMR-1 (and thus a SMR-2) from ATAM to TAM.*

Proof. By construction. Map all ATAM states to TAM unchanged. Map all ATAM state-change rules and queries unchanged, except modify absence checks via the rule in Eq. (9.1).

It is easy to see that individual states are equivalent under the mapping. Unchanged queries trivially yield the same result. Queries modified via Eq. (9.1) are equivalent, and thus also yield the same result.

It is also easy to see that individual commands are unmodified under the transformation. Thus, any state reachable in either the ATAM or TAM system is reachable in the other in exactly the same way. Hence the reachability requirement over the SMR-1 state equivalence is satisfied. □

Next, we consider the Assignment and Revocation (AAR) administrative model of RBAC compared to the trust managenment system RT[∩]. Tripunitara and Li show that there exists a state-matching reduction from AAR to RT[∩] [77, 114]. We show that there is no corresponding SMR+1.

While RT[∩] [77, 114] is a role-based trust management system, the key to the lack of SMR+1 is simply that RBAC and AAR use unbounded roles, RT roles are bound to their owner. That is, roles in RT are of the form $A.r$, which represents the role r belonging to principle A.

Theorem 27. *While there exists a state-matching reduction from AAR to* RT[∩]*, there does not exist a SMR+1 from AAR to* RT[∩].

Proof. Assume there exists a SMR+1 from AAR to RT[∩].

Consider an AAR state $\gamma^{\mathcal{A}}$ where $\langle u, r\rangle \in UA$. Note that $\gamma^{\mathcal{A}} \vdash r \sqsupseteq u$. If there exists a SMR+1 from AAR to RT[∩], then there must be an RT[∩] state $\gamma^{\mathcal{R}}$ such that $\gamma^{\mathcal{A}} \sim \gamma^{\mathcal{R}}$. However, RT[∩] is not capable of including unbound role r in its state, and therefore cannot answer TRUE to $r \sqsupseteq u$.

Thus, by contridiction, there does not exist a SMR+1 from AAR to RT[∩]. □

We note that this is not an issue for a state-matching reduction, which can map the AAR query $r \sqsupseteq u$ to RT[∩] query $Sys.r \sqsupseteq u$. However, a SMR+1 enforces QPf, which requires the query mapping to be the identity. Thus, role r cannot be mapped to $A.r$ for any principle A.

Finally, we consider a comparison between SDCO and Graham-Denning, two discretionary access control systems. While SDCO defines the distinguished right *own*, Graham-Denning also defines the *control* right that can only be held over other subjects. A full comparison of these systems is available in [76].

Tripunitara and Li state without proof that there is a state-matching reduction from SDCO to Graham-Denning. However, we show that there is no corresponding SMR+1 due to the additional distinguished right in the latter.

Theorem 28. *While there exists a state-matching reduction from SDCO to Graham-Denning, there does not exist a SMR+1 from SDCO to Graham-Denning.*

Proof. Consider the SDCO system with state $\gamma^{\mathcal{S}}$ where $S = \{s_1\}$, $O = \{o_1\}$, $M[s_1, o_1] = own$, and $R = \{own, read, control\}$.

Assume there exists a SMR+1 reduction from SDCO to Graham-Denning. Thus, there exists a Graham-Denning system that is SMR+1-equivalent to $\gamma^{\mathcal{S}}$. Call this state $\gamma^{\mathcal{G}}$.

It must be true than $\gamma^{\mathcal{G}} \vdash q_i$ for each q_i among $o_1 \in O$, $s_1 \in S$, and $own \in M[s_1, o_1]$.

Consider $\gamma_1^{\mathcal{S}}$, reachable from $\gamma^{\mathcal{S}}$ by executing $grant\_control(s_1, s_1, o_1)$. Hence $\gamma_1^{\mathcal{S}} \vdash control \in M[s_1, o_1]$. By the definition of SMR+1, there must exist a Graham-Denning state $\gamma_1^{\mathcal{G}}$ reachable from $\gamma^{\mathcal{G}}$ such that $\gamma_1^{\mathcal{G}} \vdash control \in M[s_1, o_1]$. However, by the definition of Graham-Denning, *control* can only be held over other subjects, and o_1 is not a subject. Thus, there is no state that is reachable from $\gamma^{\mathcal{G}}$ and equivalent to $\gamma_1^{\mathcal{S}}$, since $\gamma_1^{\mathcal{S}} \vdash control \in M[s_1, o_1] \wedge o_1 \notin S$, which cannot both be true in Graham-Denning.

We note that, unlike in the state-matching reduction, SMR+1 enforces QPf, and thus effectively requires the query mapping to be the identity. This prevents, e.g., using escaping of the distinguished right *control*.

Thus, given the above contradiction, there does not exist a SMR+1 reduction from SDCO to Graham-Denning. □

9.7 SUMMARY AND DISCUSSION OF RESULTS

The results discussed in Section 9.6 reveal several key results from [113,114] that are invalidated when considering a varied notion of reduction. While Tripunitara and Li conclude that TAM cannot simulate ATAM while preserving compositional security properties, we show that it can when mapping compositional security analysis instances in a different way, corresponding to a more relaxed notion of reduction. Futher, while $\mathsf{RT}[\cap]$ admits a state-matching reduction of AAR, and Graham-Denning admits a state-matching reduction of SDCO, we show that neither is possible in the stricter sense of SMR+1.

These results demonstrate the fragility of relying solely on preservation of security questions when consider a notion of expressiveness reduction. In Section 9.4, we showed that each of SMR-2, SMR-1, and SMR+1 is *necessary and sufficient* for its own corresponding method of *preservation* for compositional security analysis instances. Thus, if such a property is sufficient justification for a notion of reduction as it has been viewed in prior work [114], each of these reductions is equally valid. However, we then went on to show in Section 9.6 that many results drawn from the state-matching reduction are invalid in these other reductions, including both existence and non-existence results. We therefore argue that justifying a notion of reduction via preservation of security analysis questions alone is insufficient.

We now discuss the dimensions of reduction properties as presented in Chapter 8 that seem to be the most impactful in determining whether a reduction in question exists.

Reachability (R) appears to be an extremely important dimension. Bireachability $(\mathsf{R}\leftrightarrow)$ seems to be the biggest factor in differentiating reductions such as the SMR from older, so-called "implementation paradigm" reductions that have been shown to be too relaxed to draw useful separations between systems. When considering applications whose access control systems are sandbox-style TCBs with very limited, verified APIs, it may be sufficient to consider forward reachability only, since it can be guaranteed that the simulating system's native commands cannot be used to manipulate the policy state. However, we postulate that, in most cases, access control expressiveness analyses moving forward should consider bireachability to be required, lest they fall into the implementation paradigm that has been heavily criticized for not being strict enough to enforce strong enough guarantees.

Query dependence (QD) is the next most impactful indicator of expressiveness results, and the biggest factor among our case studies. Query decider dependence separates the state-matching reduction from the SMR-1 and SMR-2; we invalidated an influential several state-matching reduction result simply by weakening the state-independence of the query decider. With a state-dependent query decider, any aspect of the simulating system's data structures may be used to answer the simulated system's queries. Depending on the application, this can enable various undesirable reductions, including extreme examples such as "encode the simulated system as a single string and store it as a user's name in the simulating system" (this particular type of undesirable reduction can be avoided by requiring homomorphism, Definition 27). While authorization preservation (QPa) can prevent such extreme abuses by requiring that authorization requests be mapped directly, it seems appropriate to at least enforce query decider independence (QDi) as in SMR-1 to ensure that the simulated queries are answered using queriable information from the simulating state.

Command dependence stuttering (CS) is an additional dimension of impactful reduction properties. Although it is not highlighted in our case study, we reference previous work by Chander et al. [21] that shows several separations made by constant vs. lock-step stuttering (CSc vs. CS1). The ability to execute multiple simulating commands in the mapping of a single simulated command is thus shown to be a powerful ability, and has practical implications for scenarios such as multi-user systems and data structure locking (see, e.g., the cost analysis in the case study presented in Chapter 6).

In future work, we aim to conduct a deeper case study investigating the effects of altering additional reduction properties on established expressiveness results. Of particular interest is investigating other points in the reduction properties lattice at which TAM can/cannot simulate ATAM. Comparing the relative expressiveness of DAC and RBAC systems (e.g., SDCO vs. ARBAC97) using various notions of reduction may also be enlightening, since it has similarly seen results on both sides of the spectrum in prior work.

Our main conclusion of this chapter is the advantage offered by the lattice of properties presented in Chapter 8 when selecting a notion of reduction for an analysis. The previous state of the art—preservation of security analysis questions—is shown here to be far too imprecise to use alone. As we express in our thesis statement, impactful analysis of access

control needs to take into account the environment in which the access control system will be expected to operate, and in this chapter we showed that this is true when considering which notion of access control reduction to use in an expressiveness evaluation (either in the context of suitability analysis or even in ordinary expressiveness analysis).

10.0 CONCLUSIONS

Access control has long been considered one of the most central components of computer security. As such, much work has been done on both proposing new access control systems and developing techniques for the formal study of access control systems. The primary focus of the latter up to now has been evaluating the relative expressiveness of different access control systems. Expressiveness measures the raw capabilities of a system, including both static properties (what policies can be represented) and dynamic properties (how the policy can or cannot be transformed over time).

There is no question that access control will continue to be an essential component of computer security, nor that expressiveness analysis will continue to be an important method for evaluating access control systems. That said, this dissertation has shown that considering expressiveness alone, isolated from the usage requirements of the application, is vastly insufficient for a complete analysis of how suitable an access control system is for satisfying a particular workload. We thus proposed techniques for *suitability analysis*. Suitability analysis is more precise due to its consideration of the desired usage scenario, but also more expansive in that it includes costs in addition to raw capabilities.

10.1 CONTRIBUTIONS

This dissertation explored access control analyses whose end goals include deploying systems within concrete applications. As such, the techniques developed therein took into account the demands of those applications. Considering formal analysis from this perspective has allowed us to make a range of novel contributions.

In Chapter 3, we justified the need for the application-aware analysis techniques that would go on to inspire suitability analysis. We presented the first discussion of the shortcomings of application-agnostic access control evaluation, namely relative expressiveness. We proposed the novel concept of access control workload, a structure for capturing the access control demands of an application, to evaluate against. We motivated access control analysis that considers costs in addition to capabilities alone. We then discussed the general workflow of application-aware access control analysis.

In Chapter 4, we took these motivations a step farther and developed the mathematical underpinnings of suitability analysis, and in doing so proved that this conceptually new type of analysis was feasible. We formalized the suitability analysis problem and articulated a set of requirements for solving it. We developed the first two-phase suitability analysis framework, including an algorithm for conducting cost analysis. We then evaluated the framework formally by proving that it satisfied the stated requirements, and practically by providing a full analysis of a realistic case study.

In Chapter 5, we made even more concrete the claims that access control cost analysis was achievable by presenting Portuno, a Java-based simulation engine for conducting the cost analysis phase of suitability analysis. We described the methods for generating traces of representative workload usage, and discussed how they could be represented within Portuno. We showed how our simulation framework could accommodate the wide range of costs measures that were motivated in earlier chapters. Finally, we presented drivers for our general cost analysis algorithm from the preceeding chapter, allowing analysts to use that procedure to conduct Monte Carlo analysis to detect trends, or confidence-bounding simulation to target a specific expected cost within a desired confidence. We then demonstrated the applicability of Portuno by showing the details of how we represented the previous case study within the simulation engine to achieve the previously-discussed results.

In Chapter 6, we showed that suitability analysis can enable separations between access control systems and solve open questions that were not possible with prior work alone. We answered an open problem by showing cases in which group-centric information sharing workloads can be simulated by dissemination-centric access control systems, as well as cases in which dissemination-centric systems lack the expressiveness to satisfy the usage scenarios

described by this new paradigm. We then showed that, even in cases where dissemination-centric systems are capable of satisfying these workloads, they incur very large overhead in doing so.

In Chapter 7, we demonstrated the wide reach and large potential impact of suitability analysis by utilizing it toward analyzing a scenario outside of classic access control evaluation. We developed cryptographic constructions for satisfying the demands of an an application utilizing untrusted cloud storage provider. We then used two-phase suitability analysis to prove that these constructions are capable of enforcing the desired policies, but that in dynamic scenarios, are wildly inefficient in doing so. These findings provide a number of insights into areas of future research directions that could lead to better support for dynamic cryptographic access controls.

In Chapter 8, we deepened the existing view of expressiveness analysis by decomposing the various notions of expressiveness reduction into a lattice of reduction properties that can be enforced atop a minimal definition of mapping. This allowed for the consideration of a wide range of possible expressiveness metrics, in contrast to the ad hoc proposal of reduction techniques in prior work. We then decomposed the most influential notions of expressiveness reduction from the literature into their component properties, formally proving each one's position within our lattice of properties and allowing many to be formally compared for the first time.

Finally, in Chapter 9, we proved that the lattice of reduction properties is a better tool for choosing a type of reduction for an analysis than the previous state of the art. We noted that, previously, notions of reduction were justified either in a completely ad hoc manner, or through their preservation of security analysis questions. We then considered the state-matching reduction, a recent type of expressiveness reduction that has all but superceded prior notions due to its preservation of a very expressive analysis question. We presented several alternative notions of reduction, and proved using the same techniques that each also preserves this form of analysis question. We thus showed that our properties lattice for reductions is more fine-grained than considering analysis questions. Further, we then showed that these alternative reductions led to very different expressiveness results, proving that these separations are meaningful.

Recall that our goal in this dissertation was to support the following hypothesis.

> *We can develop techniques to evaluate access control systems against the specific demands of the intended usage while considering a wide range of both expressiveness and ordered cost metrics.*

The contributions of Chapter 3 justified why this type of analysis is necessary. Chapter 4 proved that the desired application-aware analysis question could be formalized with precision, and that developing the general framework for solving it was achievable. The contributions of Chapter 5 showed that scalable software tools for the cost analysis component of the analysis framework could be concretely developed, and Chapter 6 showed that it could be utilized in a large-scale analysis that solved open questions that were impossible to answer with previous techniques. We showed the framework's applicability to a wider class of problems in Chapter 7. We enabled a much wider range of expressiveness metrics in Chapter 8, and showed that this wider range contributed in a real way in Chapter 9.

10.2 FUTURE WORK

While we consider the concept of suitability analysis to be relatively mature at this point, there are several areas of future work that we will continue to explore. First, we note that there is much space left to explore within the lattice of reduction properties. We would like to continue to explore the guarantees offered by reductions of various strengths, especially in the areas of the reduction lattice that have seen little attention. Further, we will continue investigating case studies in the style of Section 9.6, determining what reduction properties cause an established result to be invalidated. This will enable not only more informed suitability analysis, but also a deeper understanding of foundational expressiveness questions surrounding *why* a reduction does or does not exist.

Another exciting area of future work is the continued application of the general suitability analysis workflow toward other problems in security. We have shown its use in cryptographic enforcement of untrusted cloud storage protections, and continue to investigate its use in an-

alyzing alternative web security solutions. Web security is built on public key infrastructures (PKIs) in which trusted certification authorities (CAs) provide certificates authorizing web services to assert a DNS name. This infrastructure, while typically thought of as an authentication system, can be modeled as an access control system that authorizes cryptographic keys held by web services to be bound to certain DNS names. Recently, there have been high profile compromises of CAs in the web PKI domain [8,25]. 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. 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.

We believe that problems such as these can be represented as suitability analysis problems. For example, web services and their certificates can be represented within the operational component of a workload. This workload's commands could represent such operations as issuing and revoking certificates, and its queries could formalize, for example, a web user verifying the identity of a server. Candidate systems, then, could be constructed to describe how each of these actions would be implemented in various proposed replacement architectures. The required type of implementation would need to enforce, e.g., that identities are not accepted without a certificate, and certificates are only be issued to the correct owner. The costs of individual actions could then be combined with an invocational workload component that describes the workflow of the typical web deployment, yielding a cost for each system.

Finally, we recognize the many components of the suitability analysis workflow that require manual specification and proof. Systems must be manually specified, workloads must be constructed, implementations built and formally proven, and Portuno data structures encoded. Some of these can be improved through the development of "libraries" of common access control systems (software and mathematical). For other tasks, we would like to consider whether it is possible to develop automated tools, at least for certain subproblems (e.g., determining that a particular reduction does not exist).

APPENDIX A

INFEASIBLE REDUCTION

Here, we describe the infeasible reduction from RBAC_1 to RBAC_0 mentioned in Section 6.4.3. The full reduction and proof are provided in the companion technical report [47].

In this reduction, we must store UR_1, PA_1, and RH_1 from RBAC_1 in only UR_0 and PA_0 in RBAC_0. We accomplish this using the following homomorphic encoding. For each $\langle u, r\rangle \in UR_1$, we generate two new constants a and b and store in UR_0 each of $\{\langle a, b\rangle, \langle u, a\rangle, \langle r, b\rangle\}$. For each $\langle r, p\rangle \in PA_1$, we generate two new constants c and d and store in PA_0 each of $\{\langle c, d\rangle, \langle c, r\rangle, \langle d, p\rangle\}$. Lastly, for each $\langle s, j\rangle \in RH_1$, we generate three new constants e, f, and g and store in PA_0 each of $\{\langle e, f\rangle, \langle f, g\rangle, \langle e, s\rangle, \langle g, j\rangle\}$.

Under this (partial) encoding, the second element of each tuple in UR_0 and the first element of each tuple in PA_0 are generated (information-less) constants. Since constants are generated to avoid collisions, there is no join over UR_0 and PA_0, which would violate AC-preservation. Finally, we add to the encoding the set of authorized requests, to fully satisfy AC-preservation. For each request $\langle u, p\rangle$ which is authorized (i.e., for each $\langle u, p\rangle$ such that $\exists s, j : \langle u, s\rangle \in UR_1 \wedge \langle j, p\rangle \in PA_1 \wedge \langle s, j\rangle \in RH_1$), we generate a new constant h and store $\langle u, h\rangle$ in UR_0 and $\langle h, p\rangle$ in PA_0.

The reduction answers queries (besides authorization requests) by extracting the relevent parts of UR_1, PA_1, and RH_1. Generated constants are identified by their positions in tuples. UR_1 tuples can be extracted from UR_0 by finding sets of three tuples which match the $\langle a, b\rangle$, $\langle u, a\rangle$, $\langle r, b\rangle$ pattern. Tuples from PA_1 and RH_1 can be extracted from PA_0 similarly.

Finally, the reduction must update the encoding after each command. For example, if

user u is assigned role r, $\langle u, r \rangle$ is encoded and stored in UR_0, then each permission p_i which u gains must be determined and encoded in UR_0 and PA_0 to satisfy AC-preservation.

APPENDIX B

DECOMPOSITION PROOFS

B.1 TL STATE-MATCHING REDUCTION

For the purposes of this proof, let the set of simulation properties $\mathcal{P} = \{\mathsf{SCq}, \mathsf{QD1}, \mathsf{R}{\leftrightarrow}\}$.

Lemma 29. *Given access control systems $\mathcal{S}$, $\mathcal{T}$, and $\mathcal{U}$,*

$$\mathcal{T} \underset{TL-SMR}{emu} \mathcal{S} \wedge \mathcal{S} \underset{\mathcal{P}}{emu} \mathcal{U} \Rightarrow \mathcal{T} \underset{\mathcal{P}}{emu} \mathcal{U}$$

That is, if $\mathcal{T}$ admits a state-matching reduction of $\mathcal{S}$, and $\mathcal{S}$ admits a simulation of $\mathcal{U}$ with properties $\{\mathsf{SCq}, \mathsf{QD1}, \mathsf{R}{\leftrightarrow}\}$, then $\mathcal{T}$ admits a simulation of $\mathcal{U}$ with properties $\{\mathsf{SCq}, \mathsf{QD1}, \mathsf{R}{\leftrightarrow}\}$.

Proof. Let $\mathcal{S}$, $\mathcal{T}$, and $\mathcal{U}$ be arbitrary access control systems such that $\mathcal{T} \underset{TL-SMR}{emu} \mathcal{S}$ and $\mathcal{S} \underset{\mathcal{P}}{emu} \mathcal{U}$. To prove Lemma 29, we must then show that $\mathcal{T} \underset{\mathcal{P}}{emu} \mathcal{U}$.

Since $\mathcal{S} \underset{\mathcal{P}}{emu} \mathcal{U}$, by $\mathsf{QD1}$,

$$\exists \sigma^{QD1} : Q^{\mathcal{U}} \rightarrow Q^{\mathcal{S}}.(\sigma_Q\big(q^{\mathcal{U}}, \gamma^{\mathcal{S}}\big) \equiv \gamma^{\mathcal{S}} \vdash \sigma^{QD1}\big(q^{\mathcal{U}}\big))$$

Since $\mathcal{T} \underset{TL-SMR}{emu} \mathcal{S}$, the state-matching reduction provides a mapping from $Q^{\mathcal{S}}$ to $Q^{\mathcal{T}}$. Call this mapping σ^{SMR}.

Thus, let $\sigma' : Q^{\mathcal{U}} \rightarrow Q^{\mathcal{T}} = \sigma^{SMR} \cdot \sigma^{QD1}$, and say $\sigma_Q\big(q^{\mathcal{U}}, \gamma^{\mathcal{T}}\big) \equiv \gamma^{\mathcal{T}} \vdash \sigma'\big(q^{\mathcal{U}}\big)\big)$. This forms a query decider that satisfies $\mathsf{QD1}$.

Choose an arbitrary state $\gamma_0^{\mathcal{U}} \in \Gamma^{\mathcal{U}}$ and command $\psi^{\mathcal{U}} \in \Psi^{\mathcal{U}}$, and let $next\big(\gamma_0^{\mathcal{U}}, \psi^{\mathcal{U}}\big) = \gamma_1^{\mathcal{U}}$. Let $\gamma_0^{\mathcal{S}} \in \Gamma^{\mathcal{S}}$ such that $\gamma_0^{\mathcal{U}} \overset{q}{\sim} \gamma_0^{\mathcal{S}}$. Since $\mathcal{S} \underset{\mathcal{P}}{emu} \mathcal{U}$,

$$\exists \gamma_1^{\mathcal{S}} \in \Gamma^{\mathcal{S}}.(terminal\big(\gamma_0^{\mathcal{S}}, \sigma_\Psi\big(\psi^{\mathcal{U}}, \gamma_0^{\mathcal{S}}\big)\big) = \gamma_1^{\mathcal{S}} \wedge \gamma_1^{\mathcal{U}} \overset{q}{\sim} \gamma_1^{\mathcal{S}})$$

Let $\gamma_0^{\mathcal{T}} \in \Gamma^{\mathcal{T}}$ such that $\gamma_0^{\mathcal{S}} \overset{q}{\sim} \gamma_0^{\mathcal{T}}$. Since $\mathcal{T} \underset{TL-SMR}{emu} \mathcal{S}$,

$$\exists \gamma_1^{\mathcal{T}} \in \Gamma^{\mathcal{T}}.(\gamma_0^{\mathcal{T}} \overset{*}{\mapsto} \gamma_1^{\mathcal{T}} \wedge \gamma_1^{\mathcal{S}} \overset{q}{\sim} \gamma_1^{\mathcal{T}})$$

Thus, there exists a sequence of $\mathcal{T}$ commands $\Psi_0^{\mathcal{T}}$ such that $terminal\big(\gamma_0^{\mathcal{T}}, \Psi_0^{\mathcal{T}}\big) = \gamma_1^{\mathcal{T}}$. Define $\sigma_\Psi : \Psi^{\mathcal{U}} \times \Gamma^{\mathcal{T}} \to \big(\Psi^{\mathcal{T}}\big)^*$ such that it returns $\Psi_0^{\mathcal{T}}$ for $\gamma_0^{\mathcal{T}}, \psi^{\mathcal{U}}$. This is formed by concatenating a sequence of sequences of commands: for each command $\psi_i^{\mathcal{S}}$ that $\mathcal{S}$ needs to execute to simulate $\psi^{\mathcal{U}}$, concatenate the commands that $\mathcal{T}$ needs to execute to simulate $\psi_i^{\mathcal{S}}$.

Then, given $\gamma_0^{\mathcal{U}}, \gamma_1^{\mathcal{U}} \in \Gamma^{\mathcal{U}}, \gamma_0^{\mathcal{T}} \in \Gamma^{\mathcal{T}}, \psi^{\mathcal{U}} \in \Psi^{\mathcal{U}}$ such that $next\big(\gamma_0^{\mathcal{U}}, \psi^{\mathcal{U}}\big) = \gamma_1^{\mathcal{U}}$, and $\gamma_0^{\mathcal{U}} \overset{q}{\sim} \gamma_0^{\mathcal{T}}$,

$$\exists \gamma_1^{\mathcal{T}} \in \Gamma^{\mathcal{T}}.(terminal\big(\gamma_0^{\mathcal{T}}, \sigma_\Psi\big(\psi, \gamma_0^{\mathcal{T}}\big)\big) = \gamma_1^{\mathcal{T}} \wedge \gamma_1^{\mathcal{S}} \overset{q}{\sim} \gamma_1^{\mathcal{T}})$$

Hence, $\mathcal{T} \underset{\{\mathsf{SCq},\mathsf{QD1},\mathsf{R}\rightarrow\}}{emu} \mathcal{U}$. Next we show $\mathsf{R}\leftrightarrow$.

Choose some arbitrary states $\gamma_0^{\mathcal{T}}, \gamma_1^{\mathcal{T}} \in \Gamma^{\mathcal{T}}$ such that $\gamma_0^{\mathcal{T}} \mapsto \gamma_1^{\mathcal{T}}$. Let $\gamma_0^{\mathcal{S}} \in \Gamma^{\mathcal{S}}$ such that $\gamma_0^{\mathcal{S}} \overset{q}{\sim} \gamma_0^{\mathcal{T}}$. Since $\mathcal{T} \underset{TL-SMR}{emu} \mathcal{S}$,

$$\exists \gamma_1^{\mathcal{S}}.(\gamma_0^{\mathcal{S}} \overset{*}{\mapsto} \gamma_1^{\mathcal{S}} \wedge \gamma_1^{\mathcal{S}} \overset{q}{\sim} \gamma_1^{\mathcal{T}})$$

Let $\gamma_0^{\mathcal{U}} \in \Gamma^{\mathcal{U}}$ such that $\gamma_0^{\mathcal{U}} \overset{q}{\sim} \gamma_0^{\mathcal{S}}$. Since $\mathcal{S} \underset{\mathcal{P}}{emu} \mathcal{U}$,

$$\exists \gamma_1^{\mathcal{U}}.(\gamma_0^{\mathcal{U}} \overset{*}{\mapsto} \gamma_1^{\mathcal{U}} \wedge \gamma_1^{\mathcal{U}} \overset{q}{\sim} \gamma_1^{\mathcal{S}})$$

Thus, given $\gamma_0^{\mathcal{T}}, \gamma_1^{\mathcal{T}} \in \Gamma^{\mathcal{T}}, \gamma_0^{\mathcal{U}} \in \Gamma^{\mathcal{U}}$ such that $\gamma_0^{\mathcal{T}} \mapsto \gamma_1^{\mathcal{T}}$ and $\gamma_0^{\mathcal{U}} \overset{q}{\sim} \gamma_0^{\mathcal{T}}$,

$$\exists \gamma_1^{\mathcal{U}} \in \Gamma^{\mathcal{U}}.(\gamma_0^{\mathcal{U}} \overset{*}{\mapsto} \gamma_1^{\mathcal{U}} \wedge \gamma_1^{\mathcal{U}} \overset{q}{\sim} \gamma_1^{\mathcal{T}})$$

Hence, $\mathcal{T} \underset{\mathcal{P}}{emu} \mathcal{U}$. □

Lemma 30. *Given access control systems $\mathcal{S}$ and $\mathcal{T}$ and simulation properties $\mathcal{P} = \{\mathsf{SCq}, \mathsf{QD1}, \mathsf{R}\leftrightarrow\}$, $\mathcal{S} \leq^{\mathcal{P}} \mathcal{T} \Rightarrow \mathcal{T} \underset{TL-SMR}{emu} \mathcal{S}$. That is, if $\mathcal{T}$ is at least as expressive as $\mathcal{S}$ with respect to properties $\mathcal{P}$, then $\mathcal{T}$ admits a state-matching reduction of $\mathcal{S}$.*

Proof. Let $\mathcal{S}$ and $\mathcal{T}$ be arbitrary access control systems such that $\mathcal{S} \leq^{\mathcal{P}} \mathcal{T}$. Since $\mathcal{S} \leq^{\mathcal{P}} \mathcal{T}$, for any access control system $\mathcal{U}$, if $\mathcal{S} \underset{\mathcal{P}}{emu} \mathcal{U}$, then $\mathcal{T} \underset{\mathcal{P}}{emu} \mathcal{U}$.

Since $\mathcal{S}$ can trivially simulate itself, $\mathcal{S} \underset{\mathcal{P}}{emu} \mathcal{S}$, and thus $\mathcal{T} \underset{\mathcal{P}}{emu} \mathcal{S}$.

By $\mathsf{QD1}$, $\exists \sigma' : Q^{\mathcal{S}} \to Q^{\mathcal{T}}.(\sigma_Q(q^{\mathcal{S}}, \gamma^{\mathcal{T}}) \equiv \gamma^{\mathcal{T}} \vdash \sigma'(q^{\mathcal{S}}))$ Thus, σ' satisfies the format of the TL-SMR query mapping (i.e., $\sigma_Q : Q^{\mathcal{S}} \to Q^{\mathcal{T}}$).

Then, given $\gamma_0^{\mathcal{S}}, \gamma_1^{\mathcal{S}} \in \Gamma^{\mathcal{S}}, \gamma_0^{\mathcal{T}} \in \Gamma^{\mathcal{T}}$, by $\mathsf{R}\!\!\rightarrow$, if $\gamma_0^{\mathcal{S}} \overset{q}{\sim} \gamma_0^{\mathcal{T}}$ and $\gamma_0^{\mathcal{S}} \mapsto \gamma_1^{\mathcal{S}}$, then

$$\exists \gamma_1^{\mathcal{T}}.(\gamma_0^{\mathcal{T}} \overset{*}{\mapsto} \gamma_1^{\mathcal{T}} \wedge \gamma_1^{\mathcal{S}} \overset{q}{\sim} \gamma_1^{\mathcal{T}})$$

Since SCq satisfies the TL-SMR definition of state correspondence, this means we have satisfied the first property of the state-matching reduction.

1. For every state $\gamma_1^{\mathcal{S}}$ in system $\mathcal{S}$ such that $\gamma_0^{\mathcal{S}} \overset{*}{\mapsto} \gamma_1^{\mathcal{S}}$, there exists a state $\gamma_1^{\mathcal{T}}$ such that $\gamma_0^{\mathcal{T}} \overset{*}{\mapsto} \gamma_1^{\mathcal{T}}$ and $\gamma_1^{\mathcal{S}}$ and $\gamma_1^{\mathcal{T}}$ are equivalent under σ.

And by bidirectional reachability, given $\gamma_0^{\mathcal{S}} \in \Gamma^{\mathcal{S}}, \gamma_0^{\mathcal{T}}, \gamma_1^{\mathcal{T}} \in \Gamma^{\mathcal{T}}$, if $\gamma_0^{\mathcal{S}} \overset{q}{\sim} \gamma_0^{\mathcal{T}}$ and $\gamma_0^{\mathcal{T}} \mapsto \gamma_1^{\mathcal{T}}$, then

$$\exists \gamma_1^{\mathcal{S}}.(\gamma_0^{\mathcal{S}} \overset{*}{\mapsto} \gamma_1^{\mathcal{S}} \wedge \gamma_1^{\mathcal{S}} \overset{q}{\sim} \gamma_1^{\mathcal{T}})$$

And therefore, we have satisfied the second property of the state-matching reduction:

2. For every state $\gamma_1^{\mathcal{T}}$ in system $\mathcal{T}$ such that $\gamma_0^{\mathcal{T}} \overset{*}{\mapsto} \gamma_1^{\mathcal{T}}$, there exists a state $\gamma_1^{\mathcal{S}}$ such that $\gamma_0^{\mathcal{S}} \overset{*}{\mapsto} \gamma_1^{\mathcal{S}}$ and $\gamma_1^{\mathcal{T}}$ and $\gamma_1^{\mathcal{S}}$ are equivalent under σ.

These properties satisfy the definition for a state-matching reduction, and hence $\mathcal{T}$ admits a state-matching reduction of $\mathcal{S}$ ($\mathcal{T} \underset{TL-SMR}{emu} \mathcal{S}$). □

Theorem 31. *TL-SMR* $\doteqdot \{\mathsf{SCq}, \mathsf{QD1}, \mathsf{R}\!\!\leftrightarrow\}$*; that is, the TL state-matching reduction decomposes to query correspondence; independent, unitary-range query; and bidirectional reachability.*

Proof. By Lemma 29, if $\mathcal{T} \underset{TL-SMR}{emu} \mathcal{S}$, then $\mathcal{S} \leq^{\mathcal{P}} \mathcal{T}$. By Lemma 30, if $\mathcal{S} \leq^{\mathcal{P}} \mathcal{T}$, then $\mathcal{T} \underset{TL-SMR}{emu} \mathcal{S}$. Thus, $\mathcal{S} \leq^{\mathcal{P}} \mathcal{T}$ if and only if $\mathcal{T} \underset{TL-SMR}{emu} \mathcal{S}$, and thus the state-matching reduction decomposes to $\{\mathsf{SCq}, \mathsf{QD1}, \mathsf{R}\!\!\leftrightarrow\}$. □

B.2 HMG+ PARAMETERIZED EXPRESSIVENESS SIMULATION

For the purposes of this proof, let the set of simulation properties $\mathcal{P} = \{\mathsf{SCq}, \mathsf{QDt}, \mathsf{R}{\rightarrow}\}$.

Lemma 32. *Given access control systems $\mathcal{S}$, $\mathcal{T}$, and $\mathcal{U}$,*

$$\mathcal{T} \underset{HMG+}{emu} \mathcal{S} \wedge \mathcal{S} \underset{\mathcal{P}}{emu} \mathcal{U} \Rightarrow \mathcal{T} \underset{\mathcal{P}}{emu} \mathcal{U}$$

That is, if $\mathcal{T}$ admits an HMG+ simulation of $\mathcal{S}$, and $\mathcal{S}$ admits a simulation of $\mathcal{U}$ with properties $\{\mathsf{SCq}, \mathsf{QDt}, \mathsf{R}{\rightarrow}\}$, then $\mathcal{T}$ admits a simulation of $\mathcal{U}$ with properties $\{\mathsf{SCq}, \mathsf{QDt}, \mathsf{R}{\rightarrow}\}$.

Proof. Let $\mathcal{S}$, $\mathcal{T}$, and $\mathcal{U}$ be arbitrary access control systems such that $\mathcal{T} \underset{HMG+}{emu} \mathcal{S}$ and $\mathcal{S} \underset{\mathcal{P}}{emu} \mathcal{U}$. To prove Lemma 32, we must then show that $\mathcal{T} \underset{\mathcal{P}}{emu} \mathcal{U}$.

Since $\mathcal{S} \underset{\mathcal{P}}{emu} \mathcal{U}$, by QDt,

$$\exists \sigma' : Q^{\mathcal{U}} \times Th(\mathcal{S}) \rightarrow \{\text{TRUE}, \text{FALSE}\}. \sigma_Q(q, \gamma) \equiv \sigma'(q, Th(\gamma))$$

Since $\mathcal{T} \underset{HMG+}{emu} \mathcal{S}$, the HMG+ simulation contains the mapping $\pi : Q^{\mathcal{S}} \times Th(\mathcal{T}) \rightarrow \{\text{TRUE}, \text{FALSE}\}$.

Thus, let $\sigma'' : Q^{\mathcal{U}} \times Th(\mathcal{T}) \rightarrow \{\text{TRUE}, \text{FALSE}\}$ be constructed as follows. Use $\sigma' : Q^{\mathcal{U}} \times Th(\mathcal{S})$, and for each query q in $Th(\mathcal{S})$ that is needed by σ', consult $\pi : Q^{\mathcal{S}} \times Th(\mathcal{T})$ to obtain a truth value in the current $\mathcal{T}$ state. This forms a query decider that satisfies QDt.

Choose an arbitrary state $\gamma_0^{\mathcal{U}} \in \Gamma^{\mathcal{U}}$ and command $\psi^{\mathcal{U}} \in \Psi^{\mathcal{U}}$, and let $next\big(\gamma_0^{\mathcal{U}}, \psi^{\mathcal{U}}\big) = \gamma_1^{\mathcal{U}}$. Let $\gamma_0^{\mathcal{S}} \in \Gamma^{\mathcal{S}}$ such that $\gamma_0^{\mathcal{U}} \overset{q}{\sim} \gamma_0^{\mathcal{S}}$. Since $\mathcal{S} \underset{\mathcal{P}}{emu} \mathcal{U}$,

$$\exists \gamma_1^{\mathcal{S}} \in \Gamma^{\mathcal{S}}. (terminal\big(\gamma_0^{\mathcal{S}}, \sigma_\Psi\big(\psi^{\mathcal{U}}, \gamma_0^{\mathcal{S}}\big)\big) = \gamma_1^{\mathcal{S}} \wedge \gamma_1^{\mathcal{U}} \overset{q}{\sim} \gamma_1^{\mathcal{S}})$$

Let $\gamma_0^{\mathcal{T}} \in \Gamma^{\mathcal{T}}$ such that $\gamma_0^{\mathcal{S}} \overset{q}{\sim} \gamma_0^{\mathcal{T}}$. Since $\mathcal{T} \underset{HMG+}{emu} \mathcal{S}$, we can map each command of the sequence $\sigma_\Psi\big(\psi^{\mathcal{U}}, \gamma_0^{\mathcal{S}}\big)$ to a sequence of $\mathcal{T}$ commands using the HMG+ simulation. Concatenating this sequence of sequences to a single sequence $\Psi_0^{\mathcal{T}}$, and using HMG+ correctness:

$$\exists \gamma_1^{\mathcal{T}} \in \Gamma^{\mathcal{T}}. (terminal\big(\gamma_0^{\mathcal{T}}, \Psi_0^{\mathcal{T}}\big) = \gamma_1^{\mathcal{T}} \wedge \gamma_1^{\mathcal{S}} \overset{q}{\sim} \gamma_1^{\mathcal{T}})$$

Define $\sigma_\Psi : \Psi^{\mathcal{U}} \times \Gamma^{\mathcal{T}} \rightarrow \big(\Psi^{\mathcal{T}}\big)^*$ such that it returns $\Psi_0^{\mathcal{T}}$ for $\gamma_0^{\mathcal{T}}, \psi^{\mathcal{U}}$.

Then, given $\gamma_0^{\mathcal{U}}, \gamma_1^{\mathcal{U}} \in \Gamma^{\mathcal{U}}, \gamma_0^{\mathcal{T}} \in \Gamma^{\mathcal{T}}, \psi^{\mathcal{U}} \in \Psi^{\mathcal{U}}$ such that $next(\gamma_0^{\mathcal{U}}, \psi^{\mathcal{U}}) = \gamma_1^{\mathcal{U}}$, and $\gamma_0^{\mathcal{U}} \overset{q}{\sim} \gamma_0^{\mathcal{T}}$,

$$\exists \gamma_1^{\mathcal{T}} \in \Gamma^{\mathcal{T}}.(terminal(\gamma_0^{\mathcal{T}}, \sigma_\Psi(\psi, \gamma_0^{\mathcal{T}})) = \gamma_1^{\mathcal{T}} \wedge \gamma_1^{\mathcal{S}} \overset{q}{\sim} \gamma_1^{\mathcal{T}})$$

Hence, $\mathcal{T} \underset{\{\mathsf{SCq},\mathsf{QDt},\mathsf{R}\rightarrow\}}{emu} \mathcal{U}$. □

Lemma 33. *Given access control systems $\mathcal{S}$ and $\mathcal{T}$ and simulation properties $\mathcal{P} = \{\mathsf{SCq}, \mathsf{QDt}, \mathsf{R}\rightarrow\}$, $\mathcal{S} \leq^{\mathcal{P}} \mathcal{T} \Rightarrow \mathcal{T} \underset{HMG+}{emu} \mathcal{S}$. That is, if $\mathcal{T}$ is at least as expressive as $\mathcal{S}$ with respect to properties $\mathcal{P}$, then $\mathcal{T}$ admits an HMG+ simulation of $\mathcal{S}$.*

Proof. Let $\mathcal{S}$ and $\mathcal{T}$ be arbitrary access control systems such that $\mathcal{S} \leq^{\mathcal{P}} \mathcal{T}$. Since $\mathcal{S} \leq^{\mathcal{P}} \mathcal{T}$, for any access control system $\mathcal{U}$, if $\mathcal{S} \underset{\mathcal{P}}{emu} \mathcal{U}$, then $\mathcal{T} \underset{\mathcal{P}}{emu} \mathcal{U}$.

Since $\mathcal{S}$ can trivially simulate itself, $\mathcal{S} \underset{\mathcal{P}}{emu} \mathcal{S}$, and thus $\mathcal{T} \underset{\mathcal{P}}{emu} \mathcal{S}$.

By QDt,

$$\exists \sigma' : Q^{\mathcal{S}} \times Th(\mathcal{T}) \rightarrow \{\text{TRUE}, \text{FALSE}\}.\sigma_Q(q^{\mathcal{S}}, \gamma^{\mathcal{T}}) \equiv \sigma'(q^{\mathcal{S}}, Th(\gamma^{\mathcal{T}}))$$

Thus, σ' satisfies the format of the HMG+ query mapping (i.e., $\pi : Q^{\mathcal{S}} \times Th(\mathcal{T}) \rightarrow \{\text{TRUE}, \text{FALSE}\}$), and by SCq, the state mapping preserves the query mapping, property (i) for the HMG+ correct simulation.

Let $\gamma_0^{\mathcal{S}} \in \Gamma^{\mathcal{S}}, \psi \in \Psi^{\mathcal{S}}$ be an arbitrary state and command in $\mathcal{S}$, and $\gamma_0^{\mathcal{T}} \in \Gamma^{\mathcal{T}}$ a state in $\mathcal{T}$ such that $\sigma_\Gamma(\gamma_0^{\mathcal{S}}) = \gamma_0^{\mathcal{T}}$. Then, if $next(\gamma_0^{\mathcal{S}}, \psi^{\mathcal{S}}) = \gamma_1^{\mathcal{S}}$,

$$\exists \gamma_1^{\mathcal{T}}.(terminal(\gamma_0^{\mathcal{T}}, \sigma_\Psi(\psi, \gamma_0^{\mathcal{T}})) = \gamma_1^{\mathcal{T}} \wedge \gamma_1^{\mathcal{S}} \overset{q}{\sim} \gamma_1^{\mathcal{T}})$$

Thus, the command mapping preserves the state mapping, property (ii) for the HMG+ correct simulation.

These properties satisfy the definition for HMG+ correct simulation, and hence $\mathcal{T}$ admits an HMG+ simulation of $\mathcal{S}$ ($\mathcal{T} \underset{HMG+}{emu} \mathcal{S}$). □

Theorem 34. *HMG+ $\doteqdot \{\mathsf{SCq}, \mathsf{QDt}, \mathsf{R}\rightarrow\}$; that is, the HMG+ parameterized expressiveness simulation (correctness only) decomposes to query correspondence, theory-dependent query, and forward reachability.*

Proof. By Lemma 32, if $\mathcal{T} \underset{HMG+}{emu} \mathcal{S}$, then $\mathcal{S} \leq^{\mathcal{P}} \mathcal{T}$. By Lemma 33, if $\mathcal{S} \leq^{\mathcal{P}} \mathcal{T}$, then $\mathcal{T} \underset{HMG+}{emu} \mathcal{S}$. Thus, $\mathcal{S} \leq^{\mathcal{P}} \mathcal{T}$ if and only if $\mathcal{T} \underset{HMG+}{emu} \mathcal{S}$, and thus the HMG+ simulation decomposes to $\{\mathsf{SCq}, \mathsf{QDt}, \mathsf{R}\rightarrow\}$. □

B.3 AC-PRESERVING HMG+ PARAMETERIZED EXPRESSIVENESS

For the purposes of this proof, let the set of simulation properties $\mathcal{P} = \{\mathsf{SCq}, \mathsf{QDt}, \mathsf{R}\!\!\rightarrow, \mathsf{QPa}\}$.

Lemma 35. *Given access control systems $\mathcal{S}$, $\mathcal{T}$, and $\mathcal{U}$,*

$$\mathcal{T} \underset{HMG+a}{emu} \mathcal{S} \wedge \mathcal{S} \underset{\mathcal{P}}{emu} \mathcal{U} \Rightarrow \mathcal{T} \underset{\mathcal{P}}{emu} \mathcal{U}$$

That is, if $\mathcal{T}$ admits an HMG+ simulation with AC-preservation of $\mathcal{S}$, and $\mathcal{S}$ admits a simulation of $\mathcal{U}$ with properties $\{\mathsf{SCq}, \mathsf{QDt}, \mathsf{R}\!\!\rightarrow, \mathsf{QPa}\}$, then $\mathcal{T}$ admits a simulation of $\mathcal{U}$ with properties $\{\mathsf{SCq}, \mathsf{QDt}, \mathsf{R}\!\!\rightarrow, \mathsf{QPa}\}$.

Proof. The proof of Lemma 32 proves all properties besides QPa with no change. Thus, we now show QPa.

Choose an arbitrary request $r \in \mathcal{R}^{\mathcal{U}}$ and states $\gamma^{\mathcal{S}} \in \Gamma^{\mathcal{S}}, \gamma^{\mathcal{T}} \in \Gamma^{\mathcal{T}}$. Since $\mathcal{S} \underset{\mathcal{P}}{emu} \mathcal{U}$,

$$\sigma_Q\big(r, \gamma^{\mathcal{S}}\big) \equiv \gamma^{\mathcal{S}} \vdash r$$

Since $\mathcal{T} \underset{HMG+a}{emu} \mathcal{S}$,

$$\forall r^{\mathcal{S}} \in \mathcal{R}^{\mathcal{S}} : \sigma_Q\big(r^{\mathcal{S}}, \gamma^{\mathcal{T}}\big) \equiv \gamma^{\mathcal{T}} \vdash r^{\mathcal{S}}$$

Since $\mathcal{S} \underset{\mathcal{P}}{emu} \mathcal{U}$, by SCq, $r \in \mathcal{R}^{\mathcal{S}}$. Thus,

$$\sigma_Q\big(r, \gamma^{\mathcal{T}}\big) \equiv \gamma^{\mathcal{T}} \vdash r$$

Hence, $\mathcal{T} \underset{\{\mathsf{SCq},\mathsf{QDt},\mathsf{R}\rightarrow,\mathsf{QPa}\}}{emu} \mathcal{U}$. □

Lemma 36. *Given access control systems $\mathcal{S}$ and $\mathcal{T}$ and simulation properties $\mathcal{P} = \{\mathsf{SCq}, \mathsf{QDt}, \mathsf{R}\!\!\rightarrow, \mathsf{QPa}\}$, $\mathcal{S} \leq^{\mathcal{P}} \mathcal{T} \Rightarrow \mathcal{T} \underset{HMG+a}{emu} \mathcal{S}$. That is, if $\mathcal{T}$ is at least as expressive as $\mathcal{S}$ with respect to properties $\mathcal{P}$, then $\mathcal{T}$ admits an HMG+ simulation with AC-preservation of $\mathcal{S}$.*

Proof. Let $\mathcal{S}$ and $\mathcal{T}$ be arbitrary access control systems such that $\mathcal{S} \leq^{\mathcal{P}} \mathcal{T}$. Since $\mathcal{S} \leq^{\mathcal{P}} \mathcal{T}$, for any access control system $\mathcal{U}$, if $\mathcal{S} \underset{\mathcal{P}}{emu} \mathcal{U}$, then $\mathcal{T} \underset{\mathcal{P}}{emu} \mathcal{U}$.

Since $\mathcal{S}$ can trivially simulate itself, $\mathcal{S} \underset{\mathcal{P}}{emu} \mathcal{S}$, and thus $\mathcal{T} \underset{\mathcal{P}}{emu} \mathcal{S}$.

The proof of Lemma 33 proves all properties besides AC-preservation. By QPa,

$$\forall r^{\mathcal{S}} \in \mathcal{R}^{\mathcal{S}} : \sigma_Q\big(r^{\mathcal{S}}, \gamma^{\mathcal{T}}\big) \equiv \gamma^{\mathcal{T}} \vdash r^{\mathcal{S}}$$

This satisfies the definition of AC-preservation, and hence $\mathcal{T}$ admits an HMG+ simulation with AC-preservation of $\mathcal{S}$ ($\mathcal{T} \underset{HMG+a}{emu} \mathcal{S}$). □

Theorem 37. *HMG+a $\doteqdot \{\mathsf{SCq}, \mathsf{QDt}, \mathsf{R}\!\!\rightarrow, \mathsf{QPa}\}$; that is, HMG+ parameterized expressiveness with AC-preservation decomposes to query correspondence, theory-dependent query, forward reachability, and authorization preservation.*

Proof. By Lemma 35, if $\mathcal{T} \underset{HMG+a}{emu} \mathcal{S}$, then $\mathcal{S} \leq^{\mathcal{P}} \mathcal{T}$. By Lemma 36, if $\mathcal{S} \leq^{\mathcal{P}} \mathcal{T}$, then $\mathcal{T} \underset{HMG+a}{emu} \mathcal{S}$. Thus, $\mathcal{S} \leq^{\mathcal{P}} \mathcal{T}$ if and only if $\mathcal{T} \underset{HMG+a}{emu} \mathcal{S}$, and thus the HMG+ simulation with AC-preservation decomposes to $\{\mathsf{SCq}, \mathsf{QDt}, \mathsf{R}\!\!\rightarrow, \mathsf{QPa}\}$. □

B.4 MONOTONIC HMG+ PARAMETERIZED EXPRESSIVENESS

For the purposes of this proof, let the set of simulation properties $\mathcal{P} = \{\mathsf{SCq}, \mathsf{QDt}, \mathsf{R}\!\!\rightarrow, \mathsf{CTa}\}$.

Lemma 38. *Given access control systems $\mathcal{S}$, $\mathcal{T}$, and $\mathcal{U}$,*

$$\mathcal{T} \underset{HMG+s}{emu} \mathcal{S} \wedge \mathcal{S} \underset{\mathcal{P}}{emu} \mathcal{U} \Rightarrow \mathcal{T} \underset{\mathcal{P}}{emu} \mathcal{U}$$

That is, if $\mathcal{T}$ admits a monotonic HMG+ simulation of $\mathcal{S}$, and $\mathcal{S}$ admits a simulation of $\mathcal{U}$ with properties $\{\mathsf{SCq}, \mathsf{QDt}, \mathsf{R}\!\!\rightarrow, \mathsf{CTa}\}$, then $\mathcal{T}$ admits a simulation of $\mathcal{U}$ with properties $\{\mathsf{SCq}, \mathsf{QDt}, \mathsf{R}\!\!\rightarrow, \mathsf{CTa}\}$.

Proof. The proof of Lemma 32 proves all properties besides CTa with no change. Thus, we now show CTa.

Choose an arbitrary command $\psi^{\mathcal{U}} \in \Psi^{\mathcal{U}}$ and state $\gamma_0^{\mathcal{S}} \in \Gamma^{\mathcal{S}}$. Let:

- $\langle \psi_1^{\mathcal{S}}, \ldots, \psi_n^{\mathcal{S}} \rangle = \sigma_\Psi\big(\psi^{\mathcal{U}}, \gamma_0^{\mathcal{S}}\big)$
- $\gamma_i^{\mathcal{S}} = terminal\big(\gamma_0^{\mathcal{S}}, \psi_1^{\mathcal{S}} \circ \cdots \circ \psi_i^{\mathcal{S}}\big)$

Since $\mathcal{S} \underset{\mathcal{P}}{emu} \mathcal{U}$, by CTa, this sequence of states $\gamma_i^{\mathcal{S}}$ is monotonic.

Furthermore, let:

- $\gamma_{0,0}^{\mathcal{T}} = \sigma_\Gamma\big(\gamma_0^{\mathcal{S}}\big)$
- $\langle \psi_{i,1}^{\mathcal{T}}, \ldots, \psi_{i,m}^{\mathcal{T}} \rangle = \sigma_\Psi\big(\psi_i^{\mathcal{S}}, \gamma_{i-1,0}^{\mathcal{T}}\big)$
- $\gamma_{i,0}^{\mathcal{T}} = terminal\big(\gamma_{i-1,0}^{\mathcal{T}}, \psi_{i,1}^{\mathcal{T}} \circ \cdots \circ \psi_{i,m}^{\mathcal{T}}\big)$
- $\gamma_{i,j}^{\mathcal{T}} = terminal\big(\gamma_{i,0}^{\mathcal{T}}, \psi_{i+1,1}^{\mathcal{T}} \circ \cdots \circ \psi_{i+1,j}^{\mathcal{T}}\big)$

Put simply, $\psi^{\mathcal{U}}$ is simulated in $\mathcal{T}$ by the following sequence of commands:

$$\psi_{1,1}^{\mathcal{T}}, \ldots, \psi_{1,m}^{\mathcal{T}}, \psi_{2,1}^{\mathcal{T}}, \ldots, \psi_{n,m}^{\mathcal{T}}$$

thus passing through the following trace of states:

$$\gamma_{0,0}^{\mathcal{T}}, \ldots, \gamma_{0,m-1}^{\mathcal{T}}, \gamma_{1,0}^{\mathcal{T}}, \ldots, \gamma_{n-1,m-1}^{\mathcal{T}}, \gamma_{n,0}^{\mathcal{T}}$$

Since $\mathcal{T} \underset{HMG+s}{emu} \mathcal{S}$, each subsequence $\gamma_{i,0}^{\mathcal{T}}, \ldots, \gamma_{i+1,0}^{\mathcal{T}}$ is monotonic.

Since $\mathcal{S} \underset{\mathcal{P}}{emu} \mathcal{U}$, by CTa and SCq, the full sequence must also be monotonic.

Hence, $\mathcal{T} \underset{\{\mathsf{SCq},\mathsf{QDt},\mathsf{R}\rightarrow,\mathsf{CTa}\}}{emu} \mathcal{U}$. □

Lemma 39. *Given access control systems $\mathcal{S}$ and $\mathcal{T}$ and simulation properties $\mathcal{P} = \{\mathsf{SCq}, \mathsf{QDt}, \mathsf{R}\rightarrow, \mathsf{CTa}\}$, $\mathcal{S} \leq^{\mathcal{P}} \mathcal{T} \Rightarrow \mathcal{T} \underset{HMG+s}{emu} \mathcal{S}$. That is, if $\mathcal{T}$ is at least as expressive as $\mathcal{S}$ with respect to properties $\mathcal{P}$, then $\mathcal{T}$ admits a monotonic HMG+ simulation of $\mathcal{S}$.*

Proof. Let $\mathcal{S}$ and $\mathcal{T}$ be arbitrary access control systems such that $\mathcal{S} \leq^{\mathcal{P}} \mathcal{T}$. Since $\mathcal{S} \leq^{\mathcal{P}} \mathcal{T}$, for any access control system $\mathcal{U}$, if $\mathcal{S} \underset{\mathcal{P}}{emu} \mathcal{U}$, then $\mathcal{T} \underset{\mathcal{P}}{emu} \mathcal{U}$.

Since $\mathcal{S}$ can trivially simulate itself, $\mathcal{S} \underset{\mathcal{P}}{emu} \mathcal{S}$, and thus $\mathcal{T} \underset{\mathcal{P}}{emu} \mathcal{S}$.

The proof of Lemma 33 proves all properties besides monotonicity. Since CTa satisfies the definition of monotonicity, we thus have that $\mathcal{T}$ admits a monotonic HMG+ simulation of $\mathcal{S}$ ($\mathcal{T} \underset{HMG+s}{emu} \mathcal{S}$). □

Theorem 40. *HMG+s* $\doteq \{\mathsf{SCq}, \mathsf{QDt}, \mathsf{R}\!\!\rightarrow, \mathsf{CTa}\}$*; that is, HMG+ parameterized expressiveness with monotonicity decomposes to query correspondence, theory-dependent query, forward reachability, and access monotonicity.*

Proof. By Lemma 38, if $\mathcal{T} \underset{HMG+s}{emu} \mathcal{S}$, then $\mathcal{S} \leq^{\mathcal{P}} \mathcal{T}$. By Lemma 39, if $\mathcal{S} \leq^{\mathcal{P}} \mathcal{T}$, then $\mathcal{T} \underset{HMG+s}{emu} \mathcal{S}$. Thus, $\mathcal{S} \leq^{\mathcal{P}} \mathcal{T}$ if and only if $\mathcal{T} \underset{HMG+s}{emu} \mathcal{S}$, and thus the monotonic HMG+ simulation decomposes to $\{\mathsf{SCq}, \mathsf{QDt}, \mathsf{R}\!\!\rightarrow, \mathsf{CTa}\}$. □

B.5 ADMIN-PRESERVING HMG+ PARAMETERIZED EXPRESSIVENESS

For the purposes of this proof, let the set of simulation properties $\mathcal{P} = \{\mathsf{SCq}, \mathsf{QDt}, \mathsf{R}\!\!\rightarrow, \mathsf{CAa}\}$.

Lemma 41. *Given access control systems* $\mathcal{S}$*,* $\mathcal{T}$*, and* $\mathcal{U}$*,*

$$\mathcal{T} \underset{HMG+p}{emu} \mathcal{S} \wedge \mathcal{S} \underset{\mathcal{P}}{emu} \mathcal{U} \Rightarrow \mathcal{T} \underset{\mathcal{P}}{emu} \mathcal{U}$$

That is, if $\mathcal{T}$ *admits an admin-preserving HMG+ simulation of* $\mathcal{S}$*, and* $\mathcal{S}$ *admits a simulation of* $\mathcal{U}$ *with properties* $\{\mathsf{SCq}, \mathsf{QDt}, \mathsf{R}\!\!\rightarrow, \mathsf{CAa}\}$*, then* $\mathcal{T}$ *admits a simulation of* $\mathcal{U}$ *with properties* $\{\mathsf{SCq}, \mathsf{QDt}, \mathsf{R}\!\!\rightarrow, \mathsf{CAa}\}$*.*

Proof. The proof of Lemma 32 proves all properties besides CAa with no change. Thus, we now show CAa.

Consider arbitrary command $\psi^{\mathcal{U}} \in \Psi^{\mathcal{U}}$ and state $\gamma^{\mathcal{S}} \in \Gamma^{\mathcal{S}}$. Since $\mathcal{S} \underset{\mathcal{P}}{emu} \mathcal{U}$, by CAa,

$$\forall \psi^{\mathcal{S}} \in \sigma_{\Psi}\left(\psi^{\mathcal{U}}, \gamma^{\mathcal{S}}\right), \alpha\left(\psi^{\mathcal{S}}\right) \in A \Rightarrow \alpha\left(\psi^{\mathcal{U}}\right) \in A$$

Consider state $\gamma^{\mathcal{T}} \in \Gamma^{\mathcal{T}}$. Since $\mathcal{T} \underset{HMG+p}{emu} \mathcal{S}$, by admin-preservation,

$$\forall \psi^{\mathcal{T}} \in \sigma_{\Psi}\left(\psi^{\mathcal{S}}, \gamma^{\mathcal{T}}\right), \alpha\left(\psi^{\mathcal{T}}\right) \in A \Rightarrow \alpha\left(\psi^{\mathcal{S}}\right) \in A$$

Thus,

$$\forall \psi^{\mathcal{T}} \in \sigma_{\Psi}\left(\psi^{\mathcal{U}}, \gamma^{\mathcal{T}}\right), \alpha\left(\psi^{\mathcal{T}}\right) \in A \Rightarrow \alpha\left(\psi^{\mathcal{U}}\right) \in A$$

Hence, $\mathcal{T} \underset{\{\mathsf{SCq}, \mathsf{QDt}, \mathsf{R}\!\!\rightarrow, \mathsf{CAa}\}}{emu} \mathcal{U}$. □

Lemma 42. *Given access control systems $\mathcal{S}$ and $\mathcal{T}$ and simulation properties $\mathcal{P} = \{\mathsf{SCq}, \mathsf{QDt}, \mathsf{R}\!\rightarrow, \mathsf{CAa}\}$, $\mathcal{S} \leq^{\mathcal{P}} \mathcal{T} \Rightarrow \mathcal{T} \underset{HMG+p}{emu} \mathcal{S}$. That is, if $\mathcal{T}$ is at least as expressive as $\mathcal{S}$ with respect to properties $\mathcal{P}$, then $\mathcal{T}$ admits an admin-preserving HMG+ simulation of $\mathcal{S}$.*

Proof. Let $\mathcal{S}$ and $\mathcal{T}$ be arbitrary access control systems such that $\mathcal{S} \leq^{\mathcal{P}} \mathcal{T}$. Since $\mathcal{S} \leq^{\mathcal{P}} \mathcal{T}$, for any access control system $\mathcal{U}$, if $\mathcal{S} \underset{\mathcal{P}}{emu} \mathcal{U}$, then $\mathcal{T} \underset{\mathcal{P}}{emu} \mathcal{U}$.

Since $\mathcal{S}$ can trivially simulate itself, $\mathcal{S} \underset{\mathcal{P}}{emu} \mathcal{S}$, and thus $\mathcal{T} \underset{\mathcal{P}}{emu} \mathcal{S}$.

The proof of Lemma 33 proves all properties besides admin-preservation. Since CAa satisfies the definition of admin-preservation, we thus have that $\mathcal{T}$ admits an admin-preserving HMG+ simulation of $\mathcal{S}$ ($\mathcal{T} \underset{HMG+p}{emu} \mathcal{S}$). □

Theorem 43. *HMG+p $\doteq \{\mathsf{SCq}, \mathsf{QDt}, \mathsf{R}\!\rightarrow, \mathsf{CAa}\}$; that is, HMG+ parameterized expressiveness with admin-preservation decomposes to query correspondence, theory-dependent query, forward reachability, and administration preservation.*

Proof. By Lemma 41, if $\mathcal{T} \underset{HMG+p}{emu} \mathcal{S}$, then $\mathcal{S} \leq^{\mathcal{P}} \mathcal{T}$. By Lemma 42, if $\mathcal{S} \leq^{\mathcal{P}} \mathcal{T}$, then $\mathcal{T} \underset{HMG+p}{emu} \mathcal{S}$. Thus, $\mathcal{S} \leq^{\mathcal{P}} \mathcal{T}$ if and only if $\mathcal{T} \underset{HMG+p}{emu} \mathcal{S}$, and thus the admin-preserving HMG+ simulation decomposes to $\{\mathsf{SCq}, \mathsf{QDt}, \mathsf{R}\!\rightarrow, \mathsf{CAa}\}$. □

B.6 SMG SIMULATION

For the purposes of this proof, let the set of simulation properties $\mathcal{P} = \{\mathsf{SCa}, \mathsf{R}\!\rightarrow\}$.

Lemma 44. *Given access control systems $\mathcal{S}$, $\mathcal{T}$, and $\mathcal{U}$,*

$$\mathcal{T} \underset{SMG}{emu} \mathcal{S} \wedge \mathcal{S} \underset{\mathcal{P}}{emu} \mathcal{U} \Rightarrow \mathcal{T} \underset{\mathcal{P}}{emu} \mathcal{U}$$

That is, if $\mathcal{T}$ admits an SMG simulation of $\mathcal{S}$, and $\mathcal{S}$ admits a simulation of $\mathcal{U}$ with properties $\{\mathsf{SCa}, \mathsf{R}\!\rightarrow\}$, then $\mathcal{T}$ admits a simulation of $\mathcal{U}$ with properties $\{\mathsf{SCa}, \mathsf{R}\!\rightarrow\}$.

Proof. Let $\mathcal{S}$, $\mathcal{T}$, and $\mathcal{U}$ be arbitrary access control systems such that $\mathcal{T} \underset{SMG}{emu} \mathcal{S}$ and $\mathcal{S} \underset{\mathcal{P}}{emu} \mathcal{U}$. To prove Lemma 44, we must then show that $\mathcal{T} \underset{\mathcal{P}}{emu} \mathcal{U}$.

Choose an arbitrary state $\gamma_0^{\mathcal{U}} \in \Gamma^{\mathcal{U}}$ and command $\psi^{\mathcal{U}} \in \Psi^{\mathcal{U}}$, and let $next\big(\gamma_0^{\mathcal{U}}, \psi^{\mathcal{U}}\big) = \gamma_1^{\mathcal{U}}$. Let $\gamma_0^{\mathcal{S}} \in \Gamma^{\mathcal{S}}$ such that $\gamma_0^{\mathcal{U}} \overset{a}{\sim} \gamma_0^{\mathcal{S}}$. Since $\mathcal{S} \underset{\mathcal{P}}{emu} \mathcal{U}$,

$$\exists \gamma_1^{\mathcal{S}} \in \Gamma^{\mathcal{S}}.(terminal\big(\gamma_0^{\mathcal{S}}, \sigma_\Psi\big(\psi^{\mathcal{U}}, \gamma_0^{\mathcal{S}}\big)\big) = \gamma_1^{\mathcal{S}} \wedge \gamma_1^{\mathcal{U}} \overset{a}{\sim} \gamma_1^{\mathcal{S}})$$

Let $\gamma_0^{\mathcal{T}} \in \Gamma^{\mathcal{T}}$ such that $\gamma_0^{\mathcal{S}} \overset{a}{\sim} \gamma_0^{\mathcal{T}}$. Since $\mathcal{T} \underset{SMG}{emu} \mathcal{S}$, request r can be granted in $\mathcal{S}$ if and only if $\sigma(r)$ can be granted in $\mathcal{T}$. More concretely,

$$\exists \gamma_1^{\mathcal{T}} \in \Gamma^{\mathcal{T}}.(\gamma_0^{\mathcal{T}} \overset{*}{\mapsto} \gamma_1^{\mathcal{T}} \wedge \gamma_1^{\mathcal{S}} \overset{a}{\sim} \gamma_1^{\mathcal{T}})$$

Thus, there exists a sequence of $\mathcal{T}$ commands $\Psi_0^{\mathcal{T}}$ such that $terminal\big(\gamma_0^{\mathcal{T}}, \Psi_0^{\mathcal{T}}\big) = \gamma_1^{\mathcal{T}}$. Define $\sigma_\Psi : \Psi^{\mathcal{U}} \times \Gamma^{\mathcal{T}} \to \big(\Psi^{\mathcal{T}}\big)^*$ such that it returns $\Psi_0^{\mathcal{T}}$ for $\gamma_0^{\mathcal{T}}, \psi^{\mathcal{U}}$. This is formed by concatenating a sequence of sequences of commands: for each command $\psi_i^{\mathcal{S}}$ that $\mathcal{S}$ needs to execute to simulate $\psi^{\mathcal{U}}$, concatenate the commands that $\mathcal{T}$ needs to execute to simulate $\psi_i^{\mathcal{S}}$.

Then, given $\gamma_0^{\mathcal{U}}, \gamma_1^{\mathcal{U}} \in \Gamma^{\mathcal{U}}, \gamma_0^{\mathcal{T}} \in \Gamma^{\mathcal{T}}, \psi^{\mathcal{U}} \in \Psi^{\mathcal{U}}$ such that $next\big(\gamma_0^{\mathcal{U}}, \psi^{\mathcal{U}}\big) = \gamma_1^{\mathcal{U}}$, and $\gamma_0^{\mathcal{U}} \overset{a}{\sim} \gamma_0^{\mathcal{T}}$,

$$\exists \gamma_1^{\mathcal{T}} \in \Gamma^{\mathcal{T}}.(terminal\big(\gamma_0^{\mathcal{T}}, \sigma_\Psi\big(\psi, \gamma_0^{\mathcal{T}}\big)\big) = \gamma_1^{\mathcal{T}} \wedge \gamma_1^{\mathcal{S}} \overset{a}{\sim} \gamma_1^{\mathcal{T}})$$

Hence, $\mathcal{T} \underset{\{\mathsf{SCa},\mathsf{R}\rightarrow\}}{emu} \mathcal{U}$. □

Lemma 45. *Given access control systems $\mathcal{S}$ and $\mathcal{T}$ and simulation properties $\mathcal{P} = \{\mathsf{SCa}, \mathsf{R}\rightarrow\}$, $\mathcal{S} \leq^{\mathcal{P}} \mathcal{T} \Rightarrow \mathcal{T} \underset{SMG}{emu} \mathcal{S}$. That is, if $\mathcal{T}$ is at least as expressive as $\mathcal{S}$ with respect to properties $\mathcal{P}$, then $\mathcal{T}$ admits an SMG simulation of $\mathcal{S}$.*

Proof. Let $\mathcal{S}$ and $\mathcal{T}$ be arbitrary access control systems such that $\mathcal{S} \leq^{\mathcal{P}} \mathcal{T}$. Since $\mathcal{S} \leq^{\mathcal{P}} \mathcal{T}$, for any access control system $\mathcal{U}$, if $\mathcal{S} \underset{\mathcal{P}}{emu} \mathcal{U}$, then $\mathcal{T} \underset{\mathcal{P}}{emu} \mathcal{U}$.

Since $\mathcal{S}$ can trivially simulate itself, $\mathcal{S} \underset{\mathcal{P}}{emu} \mathcal{S}$, and thus $\mathcal{T} \underset{\mathcal{P}}{emu} \mathcal{S}$.

By SCa and QPa, request r can be granted in $\mathcal{S}$ if and only if $\sigma(r)$ can be granted in $\mathcal{T}$. This satisfies the definition of the SMG simulation, and hence $\mathcal{T}$ admits an SMG simulation of $\mathcal{S}$ ($\mathcal{T} \underset{SMG}{emu} \mathcal{S}$). □

Theorem 46. *$SMG \doteqdot \{\mathsf{SCa}, \mathsf{R}\rightarrow\}$; that is, the SMG simulation decomposes to authorization correspondence and forward reachability.*

Proof. By Lemma 44, if $\mathcal{T} \underset{SMG}{emu} \mathcal{S}$, then $\mathcal{S} \leq^{\mathcal{P}} \mathcal{T}$. By Lemma 45, if $\mathcal{S} \leq^{\mathcal{P}} \mathcal{T}$, then $\mathcal{T} \underset{SMG}{emu} \mathcal{S}$. Thus, $\mathcal{S} \leq^{\mathcal{P}} \mathcal{T}$ if and only if $\mathcal{T} \underset{SMG}{emu} \mathcal{S}$, and thus the SMG simulation decomposes to $\{\mathsf{SCa}, \mathsf{R}\rightarrow\}$. □

B.7 GANTA SIMULATION

For the purposes of this proof, let the set of simulation properties $\mathcal{P} = \{\mathsf{SCa}, \mathsf{QPa}, \mathsf{CTs}, \mathsf{R}\leftrightarrow\}$.

Lemma 47. *Given access control systems $\mathcal{S}$, $\mathcal{T}$, and $\mathcal{U}$,*

$$\mathcal{T} \underset{Ganta}{emu} \mathcal{S} \wedge \mathcal{S} \underset{\mathcal{P}}{emu} \mathcal{U} \Rightarrow \mathcal{T} \underset{\mathcal{P}}{emu} \mathcal{U}$$

That is, if $\mathcal{T}$ admits a Ganta simulation of $\mathcal{S}$, and $\mathcal{S}$ admits a simulation of $\mathcal{U}$ with properties $\{\mathsf{SCa}, \mathsf{QPa}, \mathsf{CTs}, \mathsf{R}\leftrightarrow\}$, then $\mathcal{T}$ admits a simulation of $\mathcal{U}$ with properties $\{\mathsf{SCa}, \mathsf{QPa}, \mathsf{CTs}, \mathsf{R}\leftrightarrow\}$.

Proof. Let $\mathcal{S}$, $\mathcal{T}$, and $\mathcal{U}$ be arbitrary access control systems such that $\mathcal{T} \underset{Ganta}{emu} \mathcal{S}$ and $\mathcal{S} \underset{\mathcal{P}}{emu} \mathcal{U}$. To prove Lemma 47, we must then show that $\mathcal{T} \underset{\mathcal{P}}{emu} \mathcal{U}$.

Choose an arbitrary state $\gamma_0^{\mathcal{U}} \in \Gamma^{\mathcal{U}}$ and command $\psi^{\mathcal{U}} \in \Psi^{\mathcal{U}}$, and let $next\big(\gamma_0^{\mathcal{U}}, \psi^{\mathcal{U}}\big) = \gamma_1^{\mathcal{U}}$. Let $\gamma_0^{\mathcal{S}} \in \Gamma^{\mathcal{S}}$ such that $\gamma_0^{\mathcal{U}} \overset{a}{\sim} \gamma_0^{\mathcal{S}}$. Since $\mathcal{S} \underset{\mathcal{P}}{emu} \mathcal{U}$,

$$\exists \gamma_1^{\mathcal{S}} \in \Gamma^{\mathcal{S}}.(terminal\big(\gamma_0^{\mathcal{S}}, \sigma_\Psi\big(\psi^{\mathcal{U}}, \gamma_0^{\mathcal{S}}\big)\big) = \gamma_1^{\mathcal{S}} \wedge \gamma_1^{\mathcal{U}} \overset{a}{\sim} \gamma_1^{\mathcal{S}})$$

Let $\gamma_0^{\mathcal{T}} \in \Gamma^{\mathcal{T}}$ such that $\gamma_0^{\mathcal{S}} \overset{a}{\sim} \gamma_0^{\mathcal{T}}$. Since $\mathcal{T} \underset{Ganta}{emu} \mathcal{S}$, by Property 2 there must exist an equivalent history to $\gamma_0^{\mathcal{S}} \overset{*}{\mapsto} \gamma_1^{\mathcal{S}}$ in $\mathcal{T}$ with an access-correspondent completion state. Thus,

$$\exists \gamma_1^{\mathcal{T}} \in \Gamma^{\mathcal{T}}.(\gamma_0^{\mathcal{T}} \overset{*}{\mapsto} \gamma_1^{\mathcal{T}} \wedge \gamma_1^{\mathcal{S}} \overset{a}{\sim} \gamma_1^{\mathcal{T}})$$

Thus, there exists a sequence of $\mathcal{T}$ commands $\Psi_0^{\mathcal{T}}$ such that $terminal\big(\gamma_0^{\mathcal{T}}, \Psi_0^{\mathcal{T}}\big) = \gamma_1^{\mathcal{T}}$. Define $\sigma_\Psi : \Psi^{\mathcal{U}} \times \Gamma^{\mathcal{T}} \rightarrow \big(\Psi^{\mathcal{T}}\big)^*$ such that it returns $\Psi_0^{\mathcal{T}}$ for $\gamma_0^{\mathcal{T}}, \psi^{\mathcal{U}}$. This is formed by concatenating a sequence of sequences of commands: for each command $\psi_i^{\mathcal{S}}$ that $\mathcal{S}$ needs to execute to simulate $\psi^{\mathcal{U}}$, concatenate the commands that $\mathcal{T}$ needs to execute to simulate $\psi_i^{\mathcal{S}}$.

Then, given $\gamma_0^{\mathcal{U}}, \gamma_1^{\mathcal{U}} \in \Gamma^{\mathcal{U}}, \gamma_0^{\mathcal{T}} \in \Gamma^{\mathcal{T}}, \psi^{\mathcal{U}} \in \Psi^{\mathcal{U}}$ such that $next\big(\gamma_0^{\mathcal{U}}, \psi^{\mathcal{U}}\big) = \gamma_1^{\mathcal{U}}$, and $\gamma_0^{\mathcal{U}} \overset{a}{\sim} \gamma_0^{\mathcal{T}}$,

$$\exists \gamma_1^{\mathcal{T}} \in \Gamma^{\mathcal{T}}.(terminal\big(\gamma_0^{\mathcal{T}}, \sigma_\Psi\big(\psi, \gamma_0^{\mathcal{T}}\big)\big) = \gamma_1^{\mathcal{T}} \wedge \gamma_1^{\mathcal{S}} \overset{a}{\sim} \gamma_1^{\mathcal{T}})$$

Hence, $\mathcal{T} \underset{\{\mathsf{SCa},\mathsf{R}\rightarrow\}}{emu} \mathcal{U}$. Next we show $\mathsf{R}\leftrightarrow$.

Choose some arbitrary states $\gamma_0^{\mathcal{T}}, \gamma_1^{\mathcal{T}} \in \Gamma^{\mathcal{T}}$ such that $\gamma_0^{\mathcal{T}} \mapsto \gamma_1^{\mathcal{T}}$. Let $\gamma_0^{\mathcal{S}} \in \Gamma^{\mathcal{S}}$ such that $\gamma_0^{\mathcal{S}} \overset{q}{\sim} \gamma_0^{\mathcal{T}}$. Since $\mathcal{T} \underset{Ganta}{emu} \mathcal{S}$, by Property 3 there must exist an equivalent history to $\gamma_0^{\mathcal{T}} \mapsto \gamma_1^{\mathcal{T}}$ in $\mathcal{S}$ with an access-correspondent completion state. Thus,

$$\exists \gamma_1^{\mathcal{S}} \in \Gamma^{\mathcal{S}}.(\gamma_0^{\mathcal{S}} \overset{*}{\mapsto} \gamma_1^{\mathcal{S}} \wedge \gamma_1^{\mathcal{S}} \overset{a}{\sim} \gamma_1^{\mathcal{T}})$$

Let $\gamma_0^{\mathcal{U}} \in \Gamma^{\mathcal{U}}$ such that $\gamma_0^{\mathcal{U}} \overset{a}{\sim} \gamma_0^{\mathcal{S}}$. Since $\mathcal{S} \underset{\mathcal{P}}{emu} \mathcal{U}$,

$$\exists \gamma_1^{\mathcal{U}} \in \Gamma^{\mathcal{U}}.(\gamma_0^{\mathcal{U}} \overset{*}{\mapsto} \gamma_1^{\mathcal{U}} \wedge \gamma_1^{\mathcal{U}} \overset{a}{\sim} \gamma_1^{\mathcal{S}})$$

Thus, given $\gamma_0^{\mathcal{T}}, \gamma_1^{\mathcal{T}} \in \Gamma^{\mathcal{T}}, \gamma_0^{\mathcal{U}} \in \Gamma^{\mathcal{U}}$ such that $\gamma_0^{\mathcal{T}} \mapsto \gamma_1^{\mathcal{T}}$ and $\gamma_0^{\mathcal{U}} \overset{a}{\sim} \gamma_0^{\mathcal{T}}$,

$$\exists \gamma_1^{\mathcal{U}} \in \Gamma^{\mathcal{U}}.(\gamma_0^{\mathcal{U}} \overset{*}{\mapsto} \gamma_1^{\mathcal{U}} \wedge \gamma_1^{\mathcal{U}} \overset{a}{\sim} \gamma_1^{\mathcal{T}})$$

Hence, $\mathcal{T} \underset{\{\mathsf{SCa},\mathsf{R}\leftrightarrow\}}{emu} \mathcal{U}$. Next we show QPa.

Choose an arbitrary request $r \in \mathcal{R}^{\mathcal{U}}$ and states $\gamma^{\mathcal{S}} \in \Gamma^{\mathcal{S}}, \gamma^{\mathcal{T}} \in \Gamma^{\mathcal{T}}$. Since $\mathcal{S} \underset{\mathcal{P}}{emu} \mathcal{U}$,

$$\sigma_Q\big(r, \gamma^{\mathcal{S}}\big) \equiv \gamma^{\mathcal{S}} \vdash r$$

Since $\mathcal{T} \underset{Ganta}{emu} \mathcal{S}$, by Property 6,

$$\forall r^{\mathcal{S}} \in \mathcal{R}^{\mathcal{S}} : \sigma_Q\big(r^{\mathcal{S}}, \gamma^{\mathcal{T}}\big) \equiv \gamma^{\mathcal{T}} \vdash r^{\mathcal{S}}$$

Since $\mathcal{S} \underset{\mathcal{P}}{emu} \mathcal{U}$, by SCa, $r \in \mathcal{R}^{\mathcal{S}}$. Thus,

$$\sigma_Q\big(r, \gamma^{\mathcal{T}}\big) \equiv \gamma^{\mathcal{T}} \vdash r$$

Hence, $\mathcal{T} \underset{\{\mathsf{SCa},\mathsf{QPa},\mathsf{R}\leftrightarrow\}}{emu} \mathcal{U}$. Next we show CTs.

Choose an arbitrary command $\psi^{\mathcal{U}} \in \Psi^{\mathcal{U}}$ and state $\gamma_0^{\mathcal{S}} \in \Gamma^{\mathcal{S}}$. Let:

- $\langle \psi_1^{\mathcal{S}}, \ldots, \psi_n^{\mathcal{S}} \rangle = \sigma_\Psi\big(\psi^{\mathcal{U}}, \gamma_0^{\mathcal{S}}\big)$
- $\gamma_i^{\mathcal{S}} = terminal\big(\gamma_0^{\mathcal{S}}, \psi_1^{\mathcal{S}} \circ \cdots \circ \psi_i^{\mathcal{S}}\big)$

Since $\mathcal{S} \underset{\mathcal{P}}{emu} \mathcal{U}$, by CTs,

$$Allowed\big(\gamma_i^{\mathcal{S}}\big) \subseteq Allowed\big(\gamma_0^{\mathcal{S}}\big) \vee Allowed\big(\gamma_i^{\mathcal{S}}\big) \subseteq Allowed\big(\gamma_n^{\mathcal{S}}\big)$$

for any $\gamma_i^{\mathcal{S}}$.

Furthermore, let:

- $\gamma_{0,0}^{\mathcal{T}} = \sigma_\Gamma\big(\gamma_0^{\mathcal{S}}\big)$
- $\langle \psi_{i,1}^{\mathcal{T}}, \ldots, \psi_{i,m}^{\mathcal{T}} \rangle = \sigma_\Psi\big(\psi_i^{\mathcal{S}}, \gamma_{i-1,0}^{\mathcal{T}}\big)$
- $\gamma_{i,0}^{\mathcal{T}} = terminal\big(\gamma_{i-1,0}^{\mathcal{T}}, \psi_{i,1}^{\mathcal{T}} \circ \cdots \circ \psi_{i,m}^{\mathcal{T}}\big)$
- $\gamma_{i,j}^{\mathcal{T}} = terminal\big(\gamma_{i,0}^{\mathcal{T}}, \psi_{i+1,1}^{\mathcal{T}} \circ \cdots \circ \psi_{i+1,j}^{\mathcal{T}}\big)$

Put simply, $\psi^{\mathcal{U}}$ is simulated in $\mathcal{T}$ by the following sequence of commands:

$$\psi_{1,1}^{\mathcal{T}}, \ldots, \psi_{1,m}^{\mathcal{T}}, \psi_{2,1}^{\mathcal{T}}, \ldots, \psi_{n,m}^{\mathcal{T}}$$

thus passing through the following trace of states:

$$\gamma_{0,0}^{\mathcal{T}}, \ldots, \gamma_{0,m-1}^{\mathcal{T}}, \gamma_{1,0}^{\mathcal{T}}, \ldots, \gamma_{n-1,m-1}^{\mathcal{T}}, \gamma_{n,0}^{\mathcal{T}}$$

Consider an arbitrary state $\gamma_{i,j}^{\mathcal{T}}$ in this trace. Since $\mathcal{T} \underset{Ganta}{emu} \mathcal{S}$, by Property 6

$$Allowed\big(\gamma_{i,j}^{\mathcal{T}}\big) \subseteq Allowed\big(\gamma_{i,0}^{\mathcal{T}}\big) \vee Allowed\big(\gamma_{i,j}^{\mathcal{T}}\big) \subseteq Allowed\big(\gamma_{i+1,0}^{\mathcal{T}}\big)$$

Since $\mathcal{S} \underset{\mathcal{P}}{emu} \mathcal{U}$, by CTs and SCa,

$$Allowed\big(\gamma_{i,0}^{\mathcal{T}}\big) \subseteq Allowed\big(\gamma_{0,0}^{\mathcal{T}}\big) \vee Allowed\big(\gamma_{i,0}^{\mathcal{T}}\big) \subseteq Allowed\big(\gamma_{n,0}^{\mathcal{T}}\big)$$

as well as for $\gamma_{i+1,0}^{\mathcal{T}}$ in place of $\gamma_{i,0}^{\mathcal{T}}$.

Thus,

$$Allowed\big(\gamma_{i,j}^{\mathcal{T}}\big) \subseteq Allowed\big(\gamma_{0,0}^{\mathcal{T}}\big) \vee Allowed\big(\gamma_{i,j}^{\mathcal{T}}\big) \subseteq Allowed\big(\gamma_{n,0}^{\mathcal{T}}\big)$$

Hence, $\mathcal{T} \underset{\{\mathsf{SCa},\mathsf{QPa},\mathsf{CTs},\mathsf{R}\leftrightarrow\}}{emu} \mathcal{U}$. □

Lemma 48. *Given access control systems $\mathcal{S}$ and $\mathcal{T}$ and simulation properties $\mathcal{P} = \{\mathsf{SCa}, \mathsf{QPa}, \mathsf{CTs}, \mathsf{R}\leftrightarrow\}$, $\mathcal{S} \leq^{\mathcal{P}} \mathcal{T} \Rightarrow \mathcal{T} \underset{Ganta}{emu} \mathcal{S}$. That is, if $\mathcal{T}$ is at least as expressive as $\mathcal{S}$ with respect to properties $\mathcal{P}$, then $\mathcal{T}$ admits a Ganta simulation of $\mathcal{S}$.*

Proof. Let $\mathcal{S}$ and $\mathcal{T}$ be arbitrary access control systems such that $\mathcal{S} \leq^{\mathcal{P}} \mathcal{T}$. Since $\mathcal{S} \leq^{\mathcal{P}} \mathcal{T}$, for any access control system $\mathcal{U}$, if $\mathcal{S} \underset{\mathcal{P}}{emu} \mathcal{U}$, then $\mathcal{T} \underset{\mathcal{P}}{emu} \mathcal{U}$.

Since $\mathcal{S}$ can trivially simulate itself, $\mathcal{S} \underset{\mathcal{P}}{emu} \mathcal{S}$, and thus $\mathcal{T} \underset{\mathcal{P}}{emu} \mathcal{S}$.

By SCa, QPa, and the definition of σ_Ψ, we have a Ganta *scheme mapping* from $\mathcal{S}$ to $\mathcal{T}$, satisfying Ganta simulation Property 1.

By $\mathsf{R}\!\rightarrow$ and the definition of σ_Ψ, all histories of $\mathcal{S}$ have equivalent serial histories of $\mathcal{T}$, satisfying Property 2.

By $\mathsf{R}\!\leftrightarrow$ and the definition of σ_Ψ, all incomplete histories of $\mathcal{T}$ can be completed serially, and all complete histories of $\mathcal{T}$ have equivalent histories of $\mathcal{S}$, satisfying Properties 3–5.

Finally, by QPa and CTs, completion states are access-correspondent, and intermediate states are non-contaminating, satisfying Property 6.

These properties define the Ganta simulation, and hence $\mathcal{T}$ admits a Ganta simulation of $\mathcal{S}$ ($\mathcal{T} \underset{Ganta}{emu} \mathcal{S}$). □

Theorem 49. *Ganta* $\doteq \{\mathsf{SCa}, \mathsf{QPa}, \mathsf{CTs}, \mathsf{R}\!\leftrightarrow\}$*; that is, the Ganta simulation decomposes to access correspondence, authorization preservation, anti-contamination, and bidirectional reachability.*

Proof. By Lemma 47, if $\mathcal{T} \underset{Ganta}{emu} \mathcal{S}$, then $\mathcal{S} \leq^{\mathcal{P}} \mathcal{T}$. By Lemma 48, if $\mathcal{S} \leq^{\mathcal{P}} \mathcal{T}$, then $\mathcal{T} \underset{Ganta}{emu} \mathcal{S}$. Thus, $\mathcal{S} \leq^{\mathcal{P}} \mathcal{T}$ if and only if $\mathcal{T} \underset{Ganta}{emu} \mathcal{S}$, and thus the Ganta simulation decomposes to $\{\mathsf{SCa}, \mathsf{QPa}, \mathsf{CTs}, \mathsf{R}\!\leftrightarrow\}$. □

B.8 CDM WEAK SIMULATION

For the purposes of this proof, let the set of simulation properties $\mathcal{P} = \{\mathsf{SCa}, \mathsf{QPa}, \mathsf{CDi}, \mathsf{R}\!\rightarrow\}$.

Lemma 50. *Given access control systems* $\mathcal{S}$*,* $\mathcal{T}$*, and* $\mathcal{U}$*,*

$$\mathcal{T} \underset{CDMw}{emu} \mathcal{S} \wedge \mathcal{S} \underset{\mathcal{P}}{emu} \mathcal{U} \Rightarrow \mathcal{T} \underset{\mathcal{P}}{emu} \mathcal{U}$$

That is, if $\mathcal{T}$ admits a CDM weak simulation of $\mathcal{S}$, and $\mathcal{S}$ admits a simulation of $\mathcal{U}$ with properties $\{\mathsf{SCa}, \mathsf{QPa}, \mathsf{CDi}, \mathsf{R}\!\!\rightarrow\!\!\}$, then $\mathcal{T}$ admits a simulation of $\mathcal{U}$ with properties $\{\mathsf{SCa}, \mathsf{QPa}, \mathsf{CDi}, \mathsf{R}\!\!\rightarrow\!\!\}$.

Proof. To prove this lemma, we let $\mathcal{S}$, $\mathcal{T}$, and $\mathcal{U}$ be access control systems such that $\mathcal{T} \underset{CDMw}{emu} \mathcal{S}$ and $\mathcal{S} \underset{\mathcal{P}}{emu} \mathcal{U}$ but are otherwise arbitrary, and we show that $\mathcal{T} \underset{\mathcal{P}}{emu} \mathcal{U}$.

Choose an arbitrary state $\gamma_0^{\mathcal{U}} \in \Gamma^{\mathcal{U}}$ and command $\psi^{\mathcal{U}} \in \Psi^{\mathcal{U}}$, and let $next\big(\gamma_0^{\mathcal{U}}, \psi^{\mathcal{U}}\big) = \gamma_1^{\mathcal{U}}$. Let $\gamma_0^{\mathcal{S}} \in \Gamma^{\mathcal{S}}$ such that $\gamma_0^{\mathcal{U}} \overset{a}{\sim} \gamma_0^{\mathcal{S}}$. Since $\mathcal{S} \underset{\mathcal{P}}{emu} \mathcal{U}$,

$$\exists \gamma_1^{\mathcal{S}} \in \Gamma^{\mathcal{S}}.(terminal\big(\gamma_0^{\mathcal{S}}, \sigma_\Psi\big(\psi^{\mathcal{U}}, \gamma_0^{\mathcal{S}}\big)\big) = \gamma_1^{\mathcal{S}} \wedge \gamma_1^{\mathcal{U}} \overset{a}{\sim} \gamma_1^{\mathcal{S}})$$

Let $\gamma_0^{\mathcal{T}} \in \Gamma^{\mathcal{T}}$ such that $\gamma_0^{\mathcal{S}} \overset{a}{\sim} \gamma_0^{\mathcal{T}}$. Since $\mathcal{T} \underset{CDMw}{emu} \mathcal{S}$,

$$\exists \gamma_1^{\mathcal{T}} \in \Gamma^{\mathcal{T}}.(\gamma_0^{\mathcal{T}} \overset{*}{\mapsto} \gamma_1^{\mathcal{T}} \wedge \gamma_1^{\mathcal{S}} \overset{a}{\sim} \gamma_1^{\mathcal{T}})$$

Thus, there exists a sequence of $\mathcal{T}$ commands $\Psi_0^{\mathcal{T}}$ such that $terminal\big(\gamma_0^{\mathcal{T}}, \Psi_0^{\mathcal{T}}\big) = \gamma_1^{\mathcal{T}}$. Define $\sigma_\Psi : \Psi^{\mathcal{U}} \times \Gamma^{\mathcal{T}} \rightarrow \big(\Psi^{\mathcal{T}}\big)^*$ such that it returns $\Psi_0^{\mathcal{T}}$ for $\gamma_0^{\mathcal{T}}, \psi^{\mathcal{U}}$.

Then, given $\gamma_0^{\mathcal{U}}, \gamma_1^{\mathcal{U}} \in \Gamma^{\mathcal{U}}, \gamma_0^{\mathcal{T}} \in \Gamma^{\mathcal{T}}, \psi^{\mathcal{U}} \in \Psi^{\mathcal{U}}$ such that $next\big(\gamma_0^{\mathcal{U}}, \psi^{\mathcal{U}}\big) = \gamma_1^{\mathcal{U}}$, and $\gamma_0^{\mathcal{U}} \overset{a}{\sim} \gamma_0^{\mathcal{T}}$,

$$\exists \gamma_1^{\mathcal{T}} \in \Gamma^{\mathcal{T}}.(terminal\big(\gamma_0^{\mathcal{T}}, \sigma_\Psi\big(\psi, \gamma_0^{\mathcal{T}}\big)\big) = \gamma_1^{\mathcal{T}} \wedge \gamma_1^{\mathcal{S}} \overset{a}{\sim} \gamma_1^{\mathcal{T}})$$

Hence, $\mathcal{T} \underset{\{\mathsf{SCa},\mathsf{R}\!\!\rightarrow\!\!\}}{emu} \mathcal{U}$. Next, we show QPa.

Choose an arbitrary request $r \in \mathcal{R}^{\mathcal{U}}$ and states $\gamma^{\mathcal{S}} \in \Gamma^{\mathcal{S}}, \gamma^{\mathcal{T}} \in \Gamma^{\mathcal{T}}$. Since $\mathcal{S} \underset{\mathcal{P}}{emu} \mathcal{U}$,

$$\sigma_Q\big(r, \gamma^{\mathcal{S}}\big) \equiv \gamma^{\mathcal{S}} \vdash r$$

Since $\mathcal{T} \underset{CDMw}{emu} \mathcal{S}$,

$$\forall r^{\mathcal{S}} \in \mathcal{R}^{\mathcal{S}} : \sigma_Q\big(r^{\mathcal{S}}, \gamma^{\mathcal{T}}\big) \equiv \gamma^{\mathcal{T}} \vdash r^{\mathcal{S}}$$

Since $\mathcal{S} \underset{\mathcal{P}}{emu} \mathcal{U}$, by SCa, $r \in \mathcal{R}^{\mathcal{S}}$. Thus,

$$\sigma_Q\big(r, \gamma^{\mathcal{T}}\big) \equiv \gamma^{\mathcal{T}} \vdash r$$

Hence, $\mathcal{T} \underset{\{\mathsf{SCa},\mathsf{QPa},\mathsf{R}\!\!\rightarrow\!\!\}}{emu} \mathcal{U}$. Next we show CDi.

Since $\mathcal{S} \underset{\mathcal{P}}{emu} \mathcal{U}$, by CDi, $\exists \sigma^{CDi} : \Psi^{\mathcal{U}} \rightarrow \big(\Psi^{\mathcal{S}}\big)^*.(\sigma_\Psi(\psi, \gamma) \equiv \sigma^{CDi}(\psi))$ Thus, σ_Ψ maps $\mathcal{U}$ commands to $\mathcal{S}$ commands without considering the state in which they will be executed.

Since $\mathcal{T} \underset{CDMw}{emu} \mathcal{S}$, by weak model containment, $\mathcal{S}$ commands are mapped to $\mathcal{T}$ commands without considering the state in which they will be executed. Call this mapping σ^{CDM}.

Thus, let $\sigma' : \Psi^{\mathcal{U}} \to \left(\Psi^{\mathcal{T}}\right)^* = \sigma^{CDM} \circ \sigma^{CDi}$, and say $\sigma_\Psi\left(\psi^{\mathcal{U}}, \gamma^{\mathcal{T}}\right) \equiv \sigma'\left(\psi^{\mathcal{U}}\right)$. This forms a command mapping that satisfies CDi.

Hence, $\mathcal{T} \underset{\mathcal{P}}{emu} \mathcal{U}$. □

Lemma 51. *Given access control systems $\mathcal{S}$ and $\mathcal{T}$ and simulation properties $\mathcal{P} = \{\mathsf{SCa}, \mathsf{QPa}, \mathsf{CDi}, \mathsf{R}\!\!\to\}$, $\mathcal{S} \leq^{\mathcal{P}} \mathcal{T} \Rightarrow \mathcal{T} \underset{CDMw}{emu} \mathcal{S}$. That is, if $\mathcal{T}$ is at least as expressive as $\mathcal{S}$ with respect to properties $\mathcal{P}$, then $\mathcal{T}$ admits a CDM weak simulation of $\mathcal{S}$.*

Proof. To prove this lemma, we let $\mathcal{S}$ and $\mathcal{T}$ be arbitrary access control systems such that $\mathcal{S} \leq^{\mathcal{P}} \mathcal{T}$, and we show that $\mathcal{T} \underset{CDMw}{emu} \mathcal{S}$.

Since $\mathcal{S} \leq^{\mathcal{P}} \mathcal{T}$, for any access control system $\mathcal{U}$, if $\mathcal{S} \underset{\mathcal{P}}{emu} \mathcal{U}$, then $\mathcal{T} \underset{\mathcal{P}}{emu} \mathcal{U}$.

Since $\mathcal{S}$ can trivially simulate itself, $\mathcal{S} \underset{\mathcal{P}}{emu} \mathcal{S}$, and thus $\mathcal{T} \underset{\mathcal{P}}{emu} \mathcal{S}$.

Thus, given $\gamma_0^{\mathcal{S}}, \gamma_1^{\mathcal{S}} \in \Gamma^{\mathcal{S}}, \gamma_0^{\mathcal{T}} \in \Gamma^{\mathcal{T}}$, by SCa and R→, if $\gamma_0^{\mathcal{S}} \overset{a}{\sim} \gamma_0^{\mathcal{T}}$ and $\gamma_0^{\mathcal{S}} \mapsto \gamma_1^{\mathcal{S}}$, then

$$\exists \gamma_1^{\mathcal{T}}.(\gamma_0^{\mathcal{T}} \overset{*}{\mapsto} \gamma_1^{\mathcal{T}} \wedge \gamma_1^{\mathcal{S}} \overset{a}{\sim} \gamma_1^{\mathcal{T}})$$

By CDi,

$$\exists \sigma^{CDi} : \Psi^{\mathcal{S}} \to \left(\Psi^{\mathcal{T}}\right)^*.(\sigma_\Psi(\psi, \gamma) \equiv \sigma^{CDi}(\psi))$$

This relation is thus a weak access-containment relation, which satisfies the definition for a CDM weak simulation, and hence $\mathcal{T}$ admits a CDM weak simulation of $\mathcal{S}$ ($\mathcal{T} \underset{CDMw}{emu} \mathcal{S}$). □

Theorem 52. *$CDMw \doteqdot \{\mathsf{SCa}, \mathsf{QPa}, \mathsf{CDi}, \mathsf{R}\!\!\to\}$; that is, the CDM weak simulation decomposes to authorization correspondence, authorization preservation, independent command mapping, and forward reachability.*

Proof. By Lemma 50, if $\mathcal{T} \underset{CDMw}{emu} \mathcal{S}$, then $\mathcal{S} \leq^{\mathcal{P}} \mathcal{T}$. By Lemma 51, if $\mathcal{S} \leq^{\mathcal{P}} \mathcal{T}$, then $\mathcal{T} \underset{CDMw}{emu} \mathcal{S}$. Thus, $\mathcal{S} \leq^{\mathcal{P}} \mathcal{T}$ if and only if $\mathcal{T} \underset{CDMw}{emu} \mathcal{S}$, and thus the CDM weak simulation decomposes to $\{\mathsf{SCa}, \mathsf{QPa}, \mathsf{CDi}, \mathsf{R}\!\!\to\}$. □

B.9 CDM STRONG SIMULATION

For the purposes of this proof, let the set of simulation properties $\mathcal{P} = \{\mathsf{SCa}, \mathsf{QPa}, \mathsf{CDi}, \mathsf{CS1}, \mathsf{R}\!\rightarrow\}$.

Lemma 53. *Given access control systems $\mathcal{S}$, $\mathcal{T}$, and $\mathcal{U}$,*

$$\mathcal{T} \underset{CDMs}{emu} \mathcal{S} \wedge \mathcal{S} \underset{\mathcal{P}}{emu} \mathcal{U} \Rightarrow \mathcal{T} \underset{\mathcal{P}}{emu} \mathcal{U}$$

That is, if $\mathcal{T}$ admits a CDM strong simulation of $\mathcal{S}$, and $\mathcal{S}$ admits a simulation of $\mathcal{U}$ with properties $\{\mathsf{SCa}, \mathsf{QPa}, \mathsf{CDi}, \mathsf{CS1}, \mathsf{R}\!\rightarrow\}$, then $\mathcal{T}$ admits a simulation of $\mathcal{U}$ with properties $\{\mathsf{SCa}, \mathsf{QPa}, \mathsf{CDi}, \mathsf{CS1}, \mathsf{R}\!\rightarrow\}$.

Proof. To prove this lemma, we let $\mathcal{S}$, $\mathcal{T}$, and $\mathcal{U}$ be access control systems such that $\mathcal{T} \underset{CDMs}{emu} \mathcal{S}$ and $\mathcal{S} \underset{\mathcal{P}}{emu} \mathcal{U}$ but are otherwise arbitrary, and we show that $\mathcal{T} \underset{\mathcal{P}}{emu} \mathcal{U}$.

Choose an arbitrary state $\gamma_0^{\mathcal{U}} \in \Gamma^{\mathcal{U}}$ and command $\psi^{\mathcal{U}} \in \Psi^{\mathcal{U}}$, and let $next\big(\gamma_0^{\mathcal{U}}, \psi^{\mathcal{U}}\big) = \gamma_1^{\mathcal{U}}$. Let $\gamma_0^{\mathcal{S}} \in \Gamma^{\mathcal{S}}$ such that $\gamma_0^{\mathcal{U}} \overset{a}{\sim} \gamma_0^{\mathcal{S}}$. Since $\mathcal{S} \underset{\mathcal{P}}{emu} \mathcal{U}$,

$$\exists \gamma_1^{\mathcal{S}} \in \Gamma^{\mathcal{S}}.(terminal\big(\gamma_0^{\mathcal{S}}, \sigma_\Psi\big(\psi^{\mathcal{U}}, \gamma_0^{\mathcal{S}}\big)\big) = \gamma_1^{\mathcal{S}} \wedge \gamma_1^{\mathcal{U}} \overset{a}{\sim} \gamma_1^{\mathcal{S}})$$

Let $\gamma_0^{\mathcal{T}} \in \Gamma^{\mathcal{T}}$ such that $\gamma_0^{\mathcal{S}} \overset{a}{\sim} \gamma_0^{\mathcal{T}}$. Since $\mathcal{T} \underset{CDMs}{emu} \mathcal{S}$,

$$\exists \gamma_1^{\mathcal{T}} \in \Gamma^{\mathcal{T}}.(\gamma_0^{\mathcal{T}} \mapsto \gamma_1^{\mathcal{T}} \wedge \gamma_1^{\mathcal{S}} \overset{a}{\sim} \gamma_1^{\mathcal{T}})$$

Thus, there exists a $\mathcal{T}$ command $\psi_0^{\mathcal{T}}$ such that $next\big(\gamma_0^{\mathcal{T}}, \psi_0^{\mathcal{T}}\big) = \gamma_1^{\mathcal{T}}$. Define $\sigma_\Psi : \Psi^{\mathcal{U}} \times \Gamma^{\mathcal{T}} \rightarrow \Psi^{\mathcal{T}}$ such that it returns $\psi_0^{\mathcal{T}}$ for $\gamma_0^{\mathcal{T}}, \psi^{\mathcal{U}}$.

Then, given $\gamma_0^{\mathcal{U}}, \gamma_1^{\mathcal{U}} \in \Gamma^{\mathcal{U}}, \gamma_0^{\mathcal{T}} \in \Gamma^{\mathcal{T}}, \psi^{\mathcal{U}} \in \Psi^{\mathcal{U}}$ such that $next\big(\gamma_0^{\mathcal{U}}, \psi^{\mathcal{U}}\big) = \gamma_1^{\mathcal{U}}$, and $\gamma_0^{\mathcal{U}} \overset{a}{\sim} \gamma_0^{\mathcal{T}}$,

$$\exists \gamma_1^{\mathcal{T}} \in \Gamma^{\mathcal{T}}.(next\big(\gamma_0^{\mathcal{T}}, \sigma_\Psi\big(\psi, \gamma_0^{\mathcal{T}}\big)\big) = \gamma_1^{\mathcal{T}} \wedge \gamma_1^{\mathcal{S}} \overset{a}{\sim} \gamma_1^{\mathcal{T}})$$

Hence, $\mathcal{T} \underset{\{\mathsf{SCa},\mathsf{R}\!\rightarrow\}}{emu} \mathcal{U}$. Next, we show QPa.

Choose an arbitrary request $r \in \mathcal{R}^{\mathcal{U}}$ and states $\gamma^{\mathcal{S}} \in \Gamma^{\mathcal{S}}, \gamma^{\mathcal{T}} \in \Gamma^{\mathcal{T}}$. Since $\mathcal{S} \underset{\mathcal{P}}{emu} \mathcal{U}$,

$$\sigma_Q\big(r, \gamma^{\mathcal{S}}\big) \equiv \gamma^{\mathcal{S}} \vdash r$$

Since $\mathcal{T} \underset{CDMs}{emu} \mathcal{S}$,

$$\forall r^{\mathcal{S}} \in \mathcal{R}^{\mathcal{S}} : \sigma_Q\big(r^{\mathcal{S}}, \gamma^{\mathcal{T}}\big) \equiv \gamma^{\mathcal{T}} \vdash r^{\mathcal{S}}$$

Since $\mathcal{S} \underset{\mathcal{P}}{emu} \mathcal{U}$, by SCa, $r \in \mathcal{R}^{\mathcal{S}}$. Thus,

$$\sigma_Q\big(r, \gamma^{\mathcal{T}}\big) \equiv \gamma^{\mathcal{T}} \vdash r$$

Hence, $\mathcal{T} \underset{\{\mathsf{SCa},\mathsf{QPa},\mathsf{R}\rightarrow\}}{emu} \mathcal{U}$. Next we show CDi and CS1.

Since $\mathcal{S} \underset{\mathcal{P}}{emu} \mathcal{U}$, by CDi and CS1, $\exists \sigma^{CDi} : \Psi^{\mathcal{U}} \rightarrow \Psi^{\mathcal{S}}.(\sigma_\Psi(\psi, \gamma) \equiv \sigma^{CDi}(\psi))$ Thus, σ_Ψ maps $\mathcal{U}$ commands to $\mathcal{S}$ commands without considering the state in which they will be executed.

Since $\mathcal{T} \underset{CDMs}{emu} \mathcal{S}$, by strong model containment, $\mathcal{S}$ commands are mapped to single $\mathcal{T}$ commands without considering the state in which they will be executed. Call this mapping σ^{CDM}.

Thus, let $\sigma' : \Psi^{\mathcal{U}} \rightarrow \Psi^{\mathcal{T}} = \sigma^{CDM} \circ \sigma^{CDi}$, and say $\sigma_\Psi\big(\psi^{\mathcal{U}}, \gamma^{\mathcal{T}}\big) \equiv \sigma'\big(\psi^{\mathcal{U}}\big)$. This forms a command mapping that satisfies CDi and CS1.

Hence, $\mathcal{T} \underset{\mathcal{P}}{emu} \mathcal{U}$. □

Lemma 54. *Given access control systems $\mathcal{S}$ and $\mathcal{T}$ and simulation properties $\mathcal{P} = \{\mathsf{SCa}, \mathsf{QPa}, \mathsf{CDi}, \mathsf{CS1}, \mathsf{R}\rightarrow\}$, $\mathcal{S} \leq^{\mathcal{P}} \mathcal{T} \Rightarrow \mathcal{T} \underset{CDMs}{emu} \mathcal{S}$. That is, if $\mathcal{T}$ is at least as expressive as $\mathcal{S}$ with respect to properties $\mathcal{P}$, then $\mathcal{T}$ admits a CDM strong simulation of $\mathcal{S}$.*

Proof. To prove this lemma, we let $\mathcal{S}$ and $\mathcal{T}$ be arbitrary access control systems such that $\mathcal{S} \leq^{\mathcal{P}} \mathcal{T}$, and we show that $\mathcal{T} \underset{CDMs}{emu} \mathcal{S}$.

Since $\mathcal{S} \leq^{\mathcal{P}} \mathcal{T}$, for any access control system $\mathcal{U}$, if $\mathcal{S} \underset{\mathcal{P}}{emu} \mathcal{U}$, then $\mathcal{T} \underset{\mathcal{P}}{emu} \mathcal{U}$.

Since $\mathcal{S}$ can trivially simulate itself, $\mathcal{S} \underset{\mathcal{P}}{emu} \mathcal{S}$, and thus $\mathcal{T} \underset{\mathcal{P}}{emu} \mathcal{S}$.

Thus, given $\gamma_0^{\mathcal{S}}, \gamma_1^{\mathcal{S}} \in \Gamma^{\mathcal{S}}, \gamma_0^{\mathcal{T}} \in \Gamma^{\mathcal{T}}$, by SCa and R→, if $\gamma_0^{\mathcal{S}} \overset{a}{\sim} \gamma_0^{\mathcal{T}}$ and $\gamma_0^{\mathcal{S}} \mapsto \gamma_1^{\mathcal{S}}$, then

$$\exists \gamma_1^{\mathcal{T}}.(\gamma_0^{\mathcal{T}} \mapsto \gamma_1^{\mathcal{T}} \wedge \gamma_1^{\mathcal{S}} \overset{a}{\sim} \gamma_1^{\mathcal{T}})$$

By CDi and CS1,

$$\exists \sigma^{CDi} : \Psi^{\mathcal{S}} \rightarrow \Psi^{\mathcal{T}}.(\sigma_\Psi(\psi, \gamma) \equiv \sigma^{CDi}(\psi))$$

This relation is thus a strong access-containment relation, which satisfies the definition for a CDM strong simulation, and hence $\mathcal{T}$ admits a CDM strong simulation of $\mathcal{S}$ ($\mathcal{T} \underset{CDMs}{emu} \mathcal{S}$). □

Theorem 55. *CDMs* $\doteqdot \{\mathsf{SCa}, \mathsf{QPa}, \mathsf{CDi}, \mathsf{CS1}, \mathsf{R}{\rightarrow}\}$*; that is, the CDM strong simulation decomposes to authorization correspondence, authorization preservation, independent command mapping, lock-step, and forward reachability.*

Proof. By Lemma 53, if $\mathcal{T} \underset{CDMs}{emu} \mathcal{S}$, then $\mathcal{S} \leq^{\mathcal{P}} \mathcal{T}$. By Lemma 54, if $\mathcal{S} \leq^{\mathcal{P}} \mathcal{T}$, then $\mathcal{T} \underset{CDMs}{emu} \mathcal{S}$. Thus, $\mathcal{S} \leq^{\mathcal{P}} \mathcal{T}$ if and only if $\mathcal{T} \underset{CDMs}{emu} \mathcal{S}$, and thus the CDM strong simulation decomposes to $\{\mathsf{SCa}, \mathsf{QPa}, \mathsf{CDi}, \mathsf{CS1}, \mathsf{R}{\rightarrow}\}$. □

BIBLIOGRAPHY

[1] Selim G. Akl and Peter D. Taylor. Cryptographic solution to a problem of access control in a hierarchy. *ACM Transactions on Computer Systems (TOCS)*, 1(3):239–248, 1983.

[2] Paul Ammann, Richard J. Lipton, and Ravi S. Sandhu. The expressive power of multi-parent creation in a monotonic access control model. In *5th IEEE Computer Security Foundations Workshop (CSFW)*, pages 148–156, June 1992.

[3] Paul Ammann, Richard J. Lipton, and Ravi S. Sandhu. The expressive power of multi-parent creation in monotonic access control models. *Journal of Computer Security (JCS)*, 4(2/3):149–166, 1996.

[4] Kay S. Anderson, Joseph P. Bigus, Eric Bouillet, Parijat Dube, Nagui Halim, Zhen Liu, and Dimitrios E. Pendarakis. SWORD: scalable and flexible workload generator for distributed data processing systems. In *Winter Simulation Conference (WSC)*, pages 2109–2116, December 2006.

[5] Ross J. Anderson. *Security engineering - a guide to building dependable distributed systems (2. ed.)*. Wiley, 2008.

[6] Apache Shiro. http://shiro.apache.org.

[7] Mikhail J. Atallah, Marina Blanton, Nelly Fazio, and Keith B. Frikken. Dynamic and efficient key management for access hierarchies. *ACM Transactions on Information and System Security (TISSEC)*, 12(3), 2009.

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

[9] David Elliot Bell and Leonard J. LaPadula. Secure computer system: Mathematical foundations. Technical Report MTR-2547, MITRE Corporation, May 1973.

[10] David Elliot Bell and Leonard J. LaPadula. Secure computer system: Unified exposition and multics interpretation. Technical Report MTR-2997, MITRE Corporation, March 1976.

[11] Elisa Bertino, Barbara Catania, Elena Ferrari, and Paolo Perlasca. A logical framework for reasoning about access control models. *ACM Transactions on Information and System Security (TISSEC)*, 6(1):71–127, 2003.

[12] John Bethencourt, Amit Sahai, and Brent Waters. Ciphertext-policy attribute-based encryption. In *IEEE Symposium on Security and Privacy (S&P)*, pages 321–334, May 2007.

[13] Matt Blaze, Joan Feigenbaum, and Jack Lacy. Decentralized trust management. In *IEEE Symposium on Security and Privacy (S&P)*, pages 164–173, May 1996.

[14] Alexandra Boldyreva, Vipul Goyal, and Virendra Kumar. Identity-based encryption with efficient revocation. In *ACM Conference on Computer and Communications Security (CCS)*, pages 417–426, October 2008.

[15] Dan Boneh, Xavier Boyen, and Eu-Jin Goh. Hierarchical identity based encryption with constant size ciphertext. In *24th Annual International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT)*, pages 440–456, May 2005.

[16] Dan Boneh and Matthew K. Franklin. Identity-based encryption from the Weil pairing. *SIAM Journal on Computing (SICOMP)*, 32(3):586–615, 2003.

[17] Dan Boneh, Kevin Lewi, Hart William Montgomery, and Ananth Raghunathan. Key homomorphic PRFs and their applications. In *33rd Annual International Cryptology Conference (CRYPTO)*, pages 410–428, August 2013.

[18] Eric Bouillet, Parijat Dube, David George, Zhen Liu, Dimitrios E. Pendarakis, and Li Zhang. Distributed multi-layered workload synthesis for testing stream processing systems. In *Winter Simulation Conference (WSC)*, pages 1003–1011, December 2008.

[19] Tony Bourdier, Horatiu Cirstea, Mathieu Jaume, and Hélène Kirchner. Formal specification and validation of security policies. In *4th Canada-France MITACS Workshop on Foundations and Practice of Security*, pages 148–163, May 2011.

[20] Jae Choon Cha and Jung Hee Cheon. An identity-based signature from gap diffie-hellman groups. In *6th International Workshop on Practice and Theory in Public Key Cryptography (PKC)*, pages 18–30, January 2003.

[21] Ajay Chander, Drew Dean, and John C. Mitchell. A state-transition model of trust management and access control. In *14th IEEE Computer Security Foundations Workshop (CSFW)*, pages 27–43, June 2001.

[22] Yuan Cheng, Jaehong Park, and Ravi S. Sandhu. Relationship-based access control for online social networks: Beyond user-to-user relationships. In *4th IEEE International Conference on Privacy, Security, Risk and Trust, (PASSAT)*, pages 646–655, September 2012.

[23] Yuan Cheng, Jaehong Park, and Ravi S. Sandhu. A user-to-user relationship-based access control model for online social networks. In *26th Annual WG 11.3 Conference on Data and Applications Security and Privacy (DBSec)*, pages 8–24, July 2012.

[24] David Cohen, Jason Crampton, Andrei Gagarin, Gregory Gutin, and Mark Jones. Iterative plan construction for the workflow satisfiability problem. *Journal of Artificial Intelligence Research (JAIR)*, 51:555–577, 2014.

[25] Comodo report of incident. https://www.comodo.com/Comodo-Fraud-Incident-2011-03-23.html, March 2011.

[26] Jason Crampton. Cryptographic enforcement of role-based access control. In *7th International Workshop on Formal Aspects of Security and Trust (FAST)*, pages 191–205, September 2010.

[27] Jason Crampton. Practical and efficient cryptographic enforcement of interval-based access control policies. *ACM Transactions on Information and System Security (TISSEC)*, 14(1):14, 2011.

[28] Jason Crampton, Gregory Gutin, and Anders Yeo. On the parameterized complexity and kernelization of the workflow satisfiability problem. *ACM Transactions on Information and System Security (TISSEC)*, 16(1):4, 2013.

[29] Jason Crampton, Keith M. Martin, and Peter R. Wild. On key assignment for hierarchical access control. In *19th IEEE Computer Security Foundations Workshop (CSFW)*, pages 98–111, July 2006.

[30] Sabrina De Capitani di Vimercati, Sara Foresti, Sushil Jajodia, Giovanni Livraga, Stefano Paraboschi, and Pierangela Samarati. Enforcing dynamic write privileges in data outsourcing. *Computers & Security*, 39:47–63, 2013.

[31] Sabrina De Capitani di Vimercati, Sara Foresti, Sushil Jajodia, Stefano Paraboschi, and Pierangela Samarati. Over-encryption: Management of access control evolution on outsourced data. In *33rd International Conference on Very Large Data Bases (VLDB)*, pages 123–134, September 2007.

[32] Sabrina De Capitani di Vimercati, Sara Foresti, Sushil Jajodia, Stefano Paraboschi, and Pierangela Samarati. Encryption policies for regulating access to outsourced data. *ACM Transactions on Database Systems (TODS)*, 35(2), 2010.

[33] David Drummond. A new approach to China. https://googleblog.blogspot.com/2010/01/new-approach-to-china.html, January 2010.

[34] Alina Ene, William G. Horne, Nikola Milosavljevic, Prasad Rao, Robert Schreiber, and Robert Endre Tarjan. Fast exact and heuristic methods for role minimization problems. In *13th ACM Symposium on Access Control Models and Technologies (SACMAT)*, pages 1–10, June 2008.

[35] David F. Ferraiolo, Serban I. Gavrila, Vincent C. Hu, and D. Richard Kuhn. Composing and combining policies under the policy machine. In *10th ACM Symposium on Access Control Models and Technologies (SACMAT)*, pages 11–20, June 2005.

[36] David F. Ferraiolo, Ravi S. Sandhu, Serban I. Gavrila, D. Richard Kuhn, and Ramaswamy Chandramouli. Proposed NIST standard for role-based access control. *ACM Transactions on Information and System Security (TISSEC)*, 4(3):224–274, 2001.

[37] Anna Lisa Ferrara, Georg Fuchsbauer, Bin Liu, and Bogdan Warinschi. Policy privacy in cryptographic access control. In *IEEE 28th Computer Security Foundations Symposium (CSF)*, pages 46–60, July 2015.

[38] Anna Lisa Ferrara, Georg Fuchsbauer, and Bogdan Warinschi. Cryptographically enforced RBAC. In *IEEE 26th Computer Security Foundations Symposium (CSF)*, pages 115–129, June 2013.

[39] Philip W. L. Fong. Relationship-based access control: protection model and policy language. In *First ACM Conference on Data and Application Security and Privacy, (CODASPY)*, pages 191–202, February 2011.

[40] Gregory R. Ganger. Generating representative synthetic workloads: An unsolved problem. In *21st International Computer Measurement Group Conference (CMG)*, pages 1263–1269, December 1995.

[41] Srinivas Ganta. *Expressive Power of Access Control Models Based on Propagation of Rights*. PhD thesis, George Mason University, 1996.

[42] Simson L. Garfinkel, Gene Spafford, and Alan Schwartz. *Practical Unix and internet security*. O'Reilly Media, 3 edition, 2003.

[43] William C. Garrison III and Adam J. Lee. Decomposing, comparing, and synthesizing access control expressiveness simulations. In *IEEE 28th Computer Security Foundations Symposium (CSF)*, pages 18–32, July 2015.

[44] William C. Garrison III and Adam J. Lee. Decomposing, comparing, and synthesizing access control expressiveness simulations (extended version). Technical Report arXiv:1504.07948, Computing Research Repository, April 2015.

[45] William C. Garrison III, Adam J. Lee, and Timothy L. Hinrichs. The need for application-aware access control evaluation. In *The New Security Paradigms Workshop (NSPW)*, pages 115–126, September 2012.

[46] William C. Garrison III, Adam J. Lee, and Timothy L. Hinrichs. An actor-based, application-aware access control evaluation framework. In *19th ACM Symposium on Access Control Models and Technologies (SACMAT)*, pages 199–210, June 2014.

[47] William C. Garrison III, Yechen Qiao, and Adam J. Lee. On the suitability of dissemination-centric access control systems for group-centric sharing: Full proofs. http://www.cs.pitt.edu/~adamlee/pubs/2014/garrison2014proofs.pdf, January 2013.

[48] William C. Garrison III, Yechen Qiao, and Adam J. Lee. On the suitability of dissemination-centric access control systems for group-centric sharing. In *Fourth ACM Conference on Data and Application Security and Privacy (CODASPY)*, pages 1–12, March 2014.

[49] William C. Garrison III, Adam Shull, Steven Myers, and Adam J. Lee. Dynamic and private cryptographic access control for untrusted clouds: Costs and constructions, 2015.

[50] Craig Gentry and Alice Silverberg. Hierarchical id-based cryptography. In *8th International Conference on the Theory and Application of Cryptology and Information Security (ASIACRYPT)*, pages 548–566, December 2002.

[51] Oded Goldreich. *Computational Complexity: A Conceptual Perspective*. Cambridge University Press, 2008.

[52] Vipul Goyal, Abhishek Jain, Omkant Pandey, and Amit Sahai. Bounded ciphertext policy attribute based encryption. In *35th International Colloquium on Automata, Languages and Programming (ICALP)*, pages 579–591, July 2008.

[53] Vipul Goyal, Omkant Pandey, Amit Sahai, and Brent Waters. Attribute-based encryption for fine-grained access control of encrypted data. In *13th ACM Conference on Computer and Communications Security (CCS)*, pages 89–98, October 2006.

[54] Matthew Green and Giuseppe Ateniese. Identity-based proxy re-encryption. In *5th International Conference on Applied Cryptography and Network Security (ACNS)*, pages 288–306, June 2007.

[55] Matthew Green, Susan Hohenberger, and Brent Waters. Outsourcing the decryption of ABE ciphertexts. In *20th USENIX Security Symposium*, August 2011.

[56] Ehud Gudes. The design of a cryptopgraphy based secure file system. *IEEE Transactions on Software Engineering (TSE)*, 6(5):411–420, 1980.

[57] Michael A. Harrison, Walter L. Ruzzo, and Jeffrey D. Ullman. Protection in operating systems. *Communications of the ACM (CACM)*, 19(8):461–471, 1976.

[58] Timothy L. Hinrichs, William C. Garrison III, Adam J. Lee, Skip Saunders, and John C. Mitchell. TBA: A hybrid of logic and extensional access control systems. In *8th International Workshop on Formal Aspects of Security and Trust (FAST)*, pages 198–213, September 2011.

[59] Timothy L. Hinrichs, Diego Martinoia, William C. Garrison III, Adam J. Lee, Alessandro Panebianco, and Lenore D. Zuck. Application-sensitive access control evaluation using

parameterized expressiveness. In *IEEE 26th Computer Security Foundations Symposium (CSF)*, pages 145–160, June 2013.

[60] Horizontal integration: Broader access models for realizing information dominance. Technical Report JSR-04-13, MITRE Corporation JASON Program Office, 2004.

[61] Jeremy Horwitz and Ben Lynn. Toward hierarchical identity-based encryption. In *International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT)*, pages 466–481, May 2002.

[62] Juraj Hromkovic. *Algorithmics for Hard Problems: Introduction to Combinatorial Optimization, Randomization, Approximation, and Heuristics*. Springer, 2001.

[63] Vincent C. Hu, David F. Ferraiolo, and D. Rick Kuhn. Assessment of access control systems. Technical Report NIST-7316, National Institute of Standards and Technolog, September 2006.

[64] Vincent C. Hu, Deborah A. Frincke, and David F. Ferraiolo. The policy machine for security policy management. In *International Conference on Computational Science (ICCS)*, pages 494–506, May 2001.

[65] Luan Ibraimi. *Cryptographically enforced distributed data access control*. PhD thesis, University of Twente, 2011.

[66] Intel software guard extensions programming references. Technical Report 329298–002, Intel, October 2014.

[67] Trevor Jim. SD3: A trust management system with certified evaluation. In *IEEE Symposium on Security and Privacy (S&P)*, pages 106–115, May 2001.

[68] Xin Jin, Ram Krishnan, and Ravi S. Sandhu. A unified attribute-based access control model covering DAC, MAC and RBAC. In *26th Annual WG 11.3 Conference on Data and Applications Security and Privacy (DBSec)*, pages 41–55, July 2012.

[69] Kevin Kane and James C. Browne. On classifying access control implementations for distributed systems. In *11th ACM Symposium on Access Control Models and Technologies (SACMAT)*, pages 29–38, June 2006.

[70] Jonathan Katz, Amit Sahai, and Brent Waters. Predicate encryption supporting disjunctions, polynomial equations, and inner products. *Journal of Cryptology*, 26(2):191–224, 2013.

[71] Ram Krishnan, Jianwei Niu, Ravi S. Sandhu, and William H. Winsborough. Group-centric secure information-sharing models for isolated groups. *ACM Transactions on Information and System Security (TISSEC)*, 14(3):23, 2011.

[72] Ram Krishnan, Ravi S. Sandhu, Jianwei Niu, and William H. Winsborough. A conceptual framework for group-centric secure information sharing. In *ACM Symposium*

on Information, Computer and Communications Security (ASIACCS), pages 384–387, March 2009.

[73] Ram Krishnan, Ravi S. Sandhu, Jianwei Niu, and William H. Winsborough. Foundations for group-centric secure information sharing models. In *14th ACM Symposium on Access Control Models and Technologies (SACMAT)*, pages 115–124, June 2009.

[74] Averill Law. *Simulation Modeling and Analysis*. McGraw-Hill, 5 edition, 2014.

[75] Ninghui Li, John C. Mitchell, and William H. Winsborough. Beyond proof-of-compliance: security analysis in trust management. *Journal of the ACM (JACM)*, 52(3):474–514, 2005.

[76] Ninghui Li and Mahesh V. Tripunitara. On safety in discretionary access control. In *IEEE Symposium on Security and Privacy (S&P)*, pages 96–109, May 2005.

[77] Ninghui Li and Mahesh V. Tripunitara. Security analysis in role-based access control. *ACM Transactions on Information and System Security (TISSEC)*, 9(4):391–420, 2006.

[78] Benoît Libert and Damien Vergnaud. Adaptive-id secure revocable identity-based encryption. In *Topics in Cryptology, The Cryptographers' Track at the RSA Conference (CT-RSA*.

[79] Thomas Milton Liggett. *Continuous Time Markov Processes: An Introduction*. Graduate Studies in Mathematics Series. American Mathematical Society, 2010.

[80] Richard J. Lipton and Lawrence Snyder. A linear time algorithm for deciding subject security. *Journal of the ACM (JACM)*, 24(3):455–464, 1977.

[81] Frank McKeen, Ilya Alexandrovich, Alex Berenzon, Carlos V. Rozas, Hisham Shafi, Vedvyas Shanbhogue, and Uday R. Savagaonkar. Innovative instructions and software model for isolated execution. In *The Second Workshop on Hardware and Architectural Support for Security and Privacy (HASP)*, page 10, June 2013.

[82] Microsoft Web Protection Library. http://wpl.codeplex.com.

[83] Ian Molloy, Hong Chen, Tiancheng Li, Qihua Wang, Ninghui Li, Elisa Bertino, Seraphin B. Calo, and Jorge Lobo. Mining roles with semantic meanings. In *13th ACM Symposium on Access Control Models and Technologies (SACMAT)*, pages 21–30, June 2008.

[84] Sascha Müller and Stefan Katzenbeisser. Hiding the policy in cryptographic access control. In *7th International Workshop on Security and Trust Management (STM)*, pages 90–105, June 2011.

[85] Qamar Munawer and Ravi S. Sandhu. Simulation of the augmented typed access matrix model (ATAM) using roles. In *International Conference on Information Security (INFOSECU)*, 1999.

[86] Prasad Naldurg and Roy H. Campbell. Dynamic access control: preserving safety and trust for network defense operations. In *8th ACM Symposium on Access Control Models and Technologies (SACMAT)*, pages 231–237, June 2003.

[87] Deholo Nali, Carlisle M. Adams, and Ali Miri. Using mediated identity-based cryptography to support role-based access control. In *7th Information Security Conference (ISC)*, pages 245–256, September 2004.

[88] Operation Aurora. https://en.wikipedia.org/wiki/Operation_Aurora.

[89] Sylvia L. Osborn, Ravi S. Sandhu, and Qamar Munawer. Configuring role-based access control to enforce mandatory and discretionary access control policies. *ACM Transactions on Information and System Security (TISSEC)*, 3(2):85–106, 2000.

[90] Rafail Ostrovsky, Amit Sahai, and Brent Waters. Attribute-based encryption with non-monotonic access structures. In *ACM Conference on Computer and Communications Security (CCS)*, pages 195–203, October 2007.

[91] Seunghwan Park, Kwangsu Lee, and Dong Hoon Lee. New constructions of revocable identity-based encryption from multilinear maps. *IEEE Transactions on Information Forensics and Security*, 10(8):1564–1577, 2015.

[92] Bryan D. Payne, Reiner Sailer, Ramón Cáceres, Ronald Perez, and Wenke Lee. A layered approach to simplified access control in virtualized systems. *Operating Systems Review*, 41(4):12–19, 2007.

[93] PlayStation Plus. http://us.playstation.com/psn/playstation-plus.

[94] Tim Ring. Cloud computing hit by celebgate. http://www.scmagazineuk.com/cloud-computing-hit-by-celebgate/article/370815/, 2015.

[95] Amit Sahai and Hakan Seyalioglu. Worry-free encryption: functional encryption with public keys. In *17th ACM Conference on Computer and Communications Security (CCS)*, pages 463–472, October 2010.

[96] Amit Sahai, Hakan Seyalioglu, and Brent Waters. Dynamic credentials and ciphertext delegation for attribute-based encryption. In *32nd Annual International Cryptology Conference (CRYPTO)*, pages 199–217, August 2012.

[97] Amit Sahai and Brent Waters. Fuzzy identity-based encryption. In *24th Annual International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT)*, pages 457–473, May 2005.

[98] Jerome H. Saltzer and Michael D. Schroeder. The protection of information in computer systems. *Proceedings of the IEEE*, 63(9):1278–1308, 1975.

[99] Ravi Sandhu. Attribute-based access control models and beyond. In *10th ACM Symposium on Information, Computer and Communications Security (ASIACCS)*, page 677, April 2015.

[100] Ravi S. Sandhu. The schematic protection model: its definition and analysis for acyclic attenuating schemes. *Journal of the ACM (JACM)*, 35(2):404–432, 1988.

[101] Ravi S. Sandhu. Expressive power of the schematic protection model. *Journal of Computer Security (JCS)*, 1(1):59–98, 1992.

[102] Ravi S. Sandhu. The typed access matrix model. In *IEEE Symposium on Security and Privacy (S&P)*, pages 122–136, July 1992.

[103] Ravi S. Sandhu. Rationale for the RBAC96 family of access control models. In *ACM Workshop on Role-Based Access Control*, 1995.

[104] Ravi S. Sandhu, Edward J. Coyne, Hal L. Feinstein, and Charles E. Youman. Role-based access control models. *IEEE Computer*, 29(2):38–47, 1996.

[105] Ravi S. Sandhu and Srinivas Ganta. On testing for absence of rights in access control models. In *6th IEEE Computer Security Foundations Workshop (CSFW)*, pages 109–118, June 1993.

[106] Ravi S. Sandhu and Srinivas Ganta. On the expressive power of the unary transformation model. In *Third European Symposium on Research in Computer Security (ESORICS)*, pages 301–318, November 1994.

[107] Ravi S. Sandhu and Qamar Munawer. How to do discretionary access control using roles. In *ACM Workshop on Role-Based Access Control*, pages 47–54, 1998.

[108] Andrew Sciberras. Lightweight directory access protocol (LDAP): Schema for user applications. Technical Report RFC 4519, eB2Bcom, 2006.

[109] Jae Hong Seo and Keita Emura. Revocable identity-based cryptosystem revisited: Security models and constructions. *IEEE Transactions on Information Forensics and Security (TIFS)*, 9(7):1193–1205, 2014.

[110] Sara Sinclair, Sean W. Smith, Stephanie Trudeau, M. Eric Johnson, and Anthony Portera. Information risk in financial institutions: Field study and research roadmap. In *3rd International Workshop in Enterprise Applications and Services in the Finance Industry (FinanceCom)*, pages 165–180, December 2007.

[111] Spring Security. http://static.springsource.org/spring-security/site/.

[112] Scott D. Stoller, Ping Yang, C. R. Ramakrishnan, and Mikhail I. Gofman. Efficient policy analysis for administrative role based access control. In *ACM Conference on Computer and Communications Security (CCS)*, pages 445–455, October 2007.

[113] Mahesh V. Tripunitara and Ninghui Li. Comparing the expressive power of access control models. In *11th ACM Conference on Computer and Communications Security (CCS)*, pages 62–71, October 2004.

[114] Mahesh V. Tripunitara and Ninghui Li. A theory for comparing the expressive power of access control models. *Journal of Computer Security (JCS)*, 15(2):231–272, 2007.

[115] U.S. Air Force Scientific Advisory Board. Networking to enable coalition operations. Technical report, MITRE Corporation, 2004.

[116] Lingyu Wang, Duminda Wijesekera, and Sushil Jajodia. A logic-based framework for attribute based access control. In *ACM Workshop on Formal Methods in Security Engineering (FMSE)*, pages 45–55, October 2004.

[117] Qihua Wang and Ninghui Li. Satisfiability and resiliency in workflow authorization systems. *ACM Transactions on Information and System Security (TISSEC)*, 13(4):40, 2010.

[118] William H. Winsborough and Jay Jacobs. Automated trust negotiation in attribute-based access control. In *3rd DARPA Information Survivability Conference and Exposition (DISCEX-III)*, page 252, April 2003.

[119] Dana Zhang, Kotagiri Ramamohanarao, Steven Versteeg, and Rui Zhang. Rolevat: Visual assessment of practical need for role based access control. In *Twenty-Fifth Annual Computer Security Applications Conference (ACSAC)*, pages 13–22, December 2009.