Group-Centric Secure Information-Sharing Models for Isolated Groups

RAM KRISHNAN, JIANWEI NIU, RAVI SANDHU, and WILLIAM H. WINSBOROUGH,
University of Texas at San Antonio

Group-Centric Secure Information Sharing (g-SIS) envisions bringing users and objects together in a group to facilitate agile sharing of information brought in from external sources as well as creation of new information within the group. We expect g-SIS to be orthogonal and complementary to authorization systems deployed within participating organizations. The metaphors "secure meeting room" and "subscription service" characterize the g-SIS approach.

The focus of this article is on developing the foundations of isolated g-SIS models. Groups are *isolated* in the sense that membership of a user or an object in a group does not affect their authorizations in other groups. Present contributions include the following: formal specification of core properties that at once help to characterize the family of g-SIS models and provide a "sanity check" for full policy specifications; informal discussion of policy design decisions that differentiate g-SIS policies from one another with respect to the authorization semantics of group operations; formalization and verification of a specific member of the family of g-SIS models; demonstration that the core properties are logically consistent and mutually independent; and identification of several directions for future extensions.

The formalized specification is highly abstract. Besides certain well-formedness requirements that specify, for instance, a user cannot leave a group unless she is a member, it constrains only whether user-level read and write operations are authorized and it does so solely in terms of the history of group operations; join and leave for users and add, create, and remove for objects. This makes temporal logic one of the few formalisms in which the specification can be clearly and concisely expressed. The specification serves as a reference point that is the first step in deriving authorization-system component specifications from which a programmer with little security expertise could implement a high-assurance enforcement system for the specified policy.

Categories and Subject Descriptors: D.4.6 [**Operating Systems**]: Security and Protection—*Access controls*; K.6.5 [**Management of Computing and Information Systems**]: Security and Protection—*Unauthorized access*

General Terms: Security, Verification

Additional Key Words and Phrases: Access control, groups, information sharing, linear temporal logic, security properties

ACM Reference Format:
Krishnan, R., Niu, J., Sandhu, R., and Winsborough, W. H. 2011. Group-centric secure information-sharing models for isolated groups. ACM Trans. Inf. Syst. Secur. 14, 3, Article 23 (November 2011), 29 pages.
DOI = 10.1145/2043621.2043623 http://doi.acm.org/10.1145/2043621.2043623

The authors are partially supported by NSF grants IIS-0814027, IIS-0814027, CCR-0325951, CCF-0524010, CNS-0964710, and CNS-0716750, AFOSR MURI FA9550-08-1-0265, THECB APR 010115-0037-2007 and grants from Texas Emerging Tech. Fund and Intel Corp.
A preliminary version of some of the results reported in this article first appeared in Krishnan et al. [2009c].
Authors' addresses: R. Krishnan, Department of Electrical and Computer Engineering and Institute for Cyber Security, University of Texas at San Antonio; email: ram.krishnan@utsa.edu; J. Niu, R. Sandhu, and W. H. Winsborough, Department of Computer Science and Institute for Cyber Security, University of Texas at San Antonio; email: niu@cs.utsa.edu, ravi.sandhu@utsa.edu, wwinsborough@acm.org.

DOI 10.1145/2043621.2043623 http://doi.acm.org/10.1145/2043621.2043623

1. INTRODUCTION

The need to *share but protect* is one of the oldest and most challenging problems for trustworthy computing. Saltzer and Schroeder [1975] identified the desirability and difficulty of maintaining "some control over the user of the information even after it has been released." The ensuing three and half decades have only further compounded the technical difficulties. The analog hole [Wikipedia 2009] wherein content is captured at the point it is rendered into human perceptible form and converted back into unprotected digital form highlights the intrinsic limits. At the same time our increasingly information-rich and information-dependent society needs to exploit *secure information sharing* (SIS) to fully benefit from the productivity, social and national security benefits of the ongoing cyber revolution.

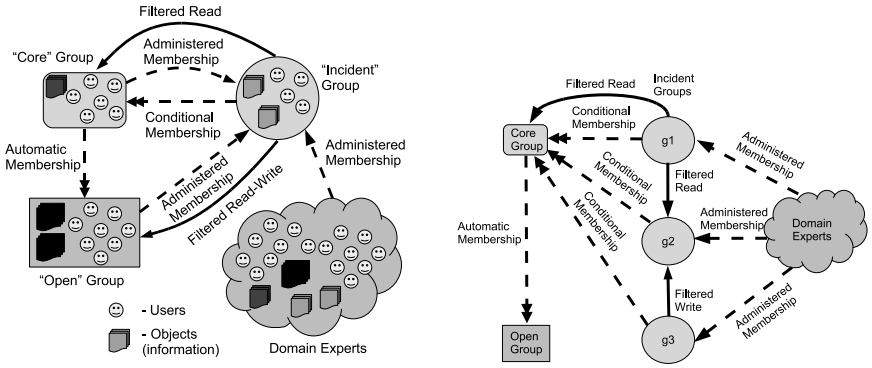
SIS research challenges include the information sharing policy design and enforcement challenge as well as the containment challenge, which we explain just below. The focus of this article is on the *policy challenge* of specifying, analyzing and enforcing SIS policies. A basic premise is that this requires new access control models that can integrate and go beyond earlier ones, have intuitive grounding and rigorous mathematical foundations, are usable by the ordinary citizen and enforceable in distributed systems.

The *containment challenge*, which is outside the scope of this article, is to ensure that protected information is accessible on the recipient's computer only as permitted by policy. This includes preventing the recipient from making unprotected or less-protected copies, as well as more traditional issues such as authentication and technological support for locally restricting access based on authorization decisions (see the confinement problem in Lampson [1973], for example). There is a rich literature on containment including the currently dominant Trusted Platform Module approach [TCG 2007]. This article assumes that adequate assurance for containment is available commensurate with the application.

The article will build upon a novel approach called Group-Centric Secure Information Sharing (g-SIS), recently introduced by the authors [Krishnan et al. 2009a, 2009b, 2009c]. A fundamental philosophy of g-SIS is to bring users and information together in a group to facilitate sharing and collaboration, reinforcing the paradigm shift from a "need to protect" to a "need to share" model. Users gain access to group information by virtue of membership. Likewise information is made available to members by adding it to the group. Using the notion of a group as the basic abstraction of SIS provides many of the same benefits of using roles versus individual users for permission distribution. However, unlike in traditional role based access control systems, the intention in g-SIS is that group users and objects can be affiliated with or originate from multiple organizations.

Two useful metaphors for a g-SIS group are a subscription service and a secure meeting room. Subscription disseminates information to subscribers who participate in blogs and forums accessible to the subscription group. In the second metaphor, a group represents a meeting room that brings people together to share information available in the room, be it brought into the room from external sources or internally created within the room. During the course of the meeting, new information may be created or information already present in the room may be updated to create new versions. The times at which users join and leave and at which objects and its versions are added and removed may affect user authorizations both during and after periods of group membership.

Prior frameworks intended to support group-based policies that are sensitive to the relative times at which group operations, such as user join and object add, have tended to be quite rigid. For instance, in the much studied secure multicast problem [Rafaeli and Hutchison 2003], typically new group members cannot access content added prior to user-join time (backward secrecy) and after leaving the group, users retain access



(a) Cyber-incident life-cycle (b) Dynamic group relationship

Fig. 1. Community cyber security incident management.

to content that was added during the period of their membership. The intention of g-SIS is to be more flexible with respect to this sort of policy decision. For instance, in addition to being able to support the policy enforced by secure multicast, a g-SIS policy might grant members access to information added to the group before they joined (no backward secrecy). Similarly, a g-SIS policy might require that users lose all access to information in the group once they leave. As we will see, the policy elements associated with group operations in g-SIS, and their combined effect on authorization, can be intricate and the space of global g-SIS policies is vast. Cataloging the options and precisely specifying specific policies is a challenging research problem.

The objectives of g-SIS are distinct from those of the prior work in access control of which we are aware. Specifically, the design of g-SIS should be conceptually lightweight; support create, add and remove operations for objects containing information; and enable users to join and leave groups. The intention is that the security models already in use by participating organizations need not be integrated with the g-SIS system. Instead, g-SIS should be orthogonal and complementary to those deployed models. These objectives are intended to support agile responsiveness to shifting circumstances, which occur, for instance, in cyber incident-management life cycle (to be discussed later). To the best of our knowledge, these goals are novel. They are sharply distinguished from those of prior work, such as in the context of dynamic coalitions, which generally focus on integrating security infrastructure across organizational boundaries [Cohen et al. 2002; Freudenthal et al. 2002; Shands et al. 2001].

2. VISION, METHODOLOGY, AND SCOPE

We motivate g-SIS and discuss our long-term research goals using the information sharing domain of community cyber security, where community refers to a county or larger city size unit with a clearly demarcated geographical and/or governance boundary.[1] We believe similar motivation will be applicable in numerous other SIS domains.

2.1. Motivational Scenario

Our scenario is developed in the context of the life cycle of a cyber security incident in a community illustrated in Figure 1(a). A basic premise is that several different organizations need to be involved in recognizing and managing a community-scale cyber

[1]Such scenarios are encountered by the Center for Infrastructure Assurance and Security (CIAS) at the University of Texas at San Antonio which conducts several cyber incident management exercises across the nation. See White and Granado [2009], for example.

security incident. These include local government organizations, for instance, police, fire and health, quasi-government organizations, for instance, energy and power utilities, and private organizations, for instance, telecommunications, internet and financial service providers, and so on. In the steady state consider two "permanent" groups that enable SIS across these organizations. These are the open and core groups in Figure 1(a). The notion of the open group is that it is open to voluntary participation by employees of designated organizations, perhaps subject to some admission criteria such as excluding temporary contractors. Voluntary membership in this group is maintained so long as the admission criteria remains valid. Information shared in the open group is likely to be of relatively low sensitivity. The core group on the other hand enables sharing more sensitive information and its membership is administered. Perhaps, each organization gets to nominate a few participants as appropriate. Members of the core group are automatically enrolled in the open group as a matter of convenience. It is anticipated that over time members of the open group will develop reputation and trust rankings based on their activity and responses from other open group members.

When a cyber incident occurs or is suspected, the initial information sharing may continue to be facilitated via these two groups. At some point it may be appropriate to establish an "incident" group which would focus on this particular incident. Members of the incident group are selected administratively from the core group. The criteria for selection could involve special expertise or representation of a particular sector such as telecommunications. Membership in the incident group is conditional on continued membership in the core group. Additional expertise from outside the core group may be appropriate within the incident group. Two avenues for this purpose are illustrated, from the open group or from a "cloud" of subject area experts such as professors. The incident group is provided a filtered read capability into the core group which enables selective access to core group information. Further the incident group has a filtered read/write capability to the open group. This can allow solicitation of particular expertise or information from that group for example.

Over time the incident group may dissolve or persist for some time depending on circumstances. Figure 1(b) shows the simultaneous existence of multiple incident groups. Our long-term vision for g-SIS is to capture the dynamism of secure information sharing such as in this community-based scenario, and build models with strong mathematical foundations and intuitive appeal. It should be noted that a particular static snapshot of this evolving scenario is likely to be implementable using traditional models such as Bell-LaPadula (BLP) [Bell and La Padula 1975] and DTE [Badger et al. 1995]. The difference is that the latter models are intended for static situations. Their mathematical foundations often assume a static unchanging foundation, such as an assignment of security labels that cannot change (tranquility in BLP). The motivation for g-SIS is agile dynamism and evolution in response to real-world events and information sharing requirements. Agility is enabled by defining appropriate intergroup relationships that facilitate easy administration in response to shifting information sharing requirements. We plan to develop isolated and connected group g-SIS models. In *isolated* g-SIS, multiple groups may exist but user's membership in one group has no implication on her authorizations in other groups. In *connected* g-SIS, membership in one group may have implications on authorizations that are enabled or disabled in other groups.

2.2. Research Methodology

This article represents an initial contribution in the development of a larger research vision, the end goal of which is to provide system component specifications that are

sufficiently concrete that a programmer with little or no security expertise can implement a g-SIS system with high assurance of adherence to the security objectives. This article is concerned primarily with abstract policy capturing those objectives. A basic premise of our research methodology is that specifying and analyzing the application policy should be clearly separated from enforcement policy issues that arise due to the realities and practicalities of a distributed system. Following Sandhu et al. [2006], Krishnan et al. [2007], and Sandhu [2009], we call these respectively P-layer (for application policy) and E-layer (for enforcement policy) concerns. P-layer specifications express a policy that is ideal in that it ignores issues such as distributed authorization state, network latency, caching, and requirements for off-line use. E-layer specifications define authorization decisions that approximate those given by the ideal policy in a manner that provides the desired application-dependent balance between resource availability and timely propagation of authorization-state changes.

To manage complexity, a progression of g-SIS models can be incrementally developed. At the P-layer, our vision is to develop stateless and stateful specifications of the isolated and connected group models. We call a specification *stateless* if a user's authorization to access an object is specified based solely on the history of group operations. Specifications such as this are our current focus. Reliance on the event history makes using temporal logic in the specification a natural choice. In particular, we use first-order linear temporal logic (FOTL), as doing so enables us to support an unbounded number of users, subjects, objects, and groups. By contrast with the stateless specification's reliance on event histories, the *stateful* specification introduces data structures that record sufficient information about the history to support making efficient authorization decisions. The separation of stateless and stateful specification increases assurance that designs meet the intended security objectives. By focusing initially on a stateless specification, we avoid being distracted by issues surrounding mechanism for supporting authorization decisions. Specifying the intended system behavior in two different formalisms supports high assurance of meeting security objectives because one is unlikely to make the same errors in both specifications.

2.3. Article Scope

In this initial work we focus on formal specification of the requirements of an *isolated* g-SIS model and developing *stateless* specifications. (Note that a user may belong to multiple isolated groups, and an object may be made accessible to multiple groups.)

Users may join, leave, and rejoin; external objects may be added, removed, and readded; new objects may be locally (internally) created in the group. Each of these group operations could be of various types (see Sections 3.5 and 3.6). For instance, on *strict leave* the user loses authorization to read all group objects. On *liberal leave*, the user may retain read access to some of the objects authorized during membership period. In general, there may be many variations beyond those explicitly identified in this article. For convenience and readability, we do not include a delete or remove operation for internally created objects, which we anticipate would be a straightforward extension.

We consider read and write permissions. Externally added objects in a group can only be read. Internally created objects can be read and written (to create a new version). The authorization for read and write operations is determined by the interaction between the basic group operations and the particular semantics of different variations of these. For convenience we understand object, version and object version to roughly mean the same thing, unless we need to be absolutely precise. To achieve containment of information within an isolated group we utilize the classic distinction between users and subjects. The term user refers to a representation of a human in the system. The term subject refers to processes that run on behalf of the user. A user may create a

subject in a specific group to read and write versions as authorized. We restrict a subject's read and write access to a single group. We ignore read-only subjects that may read from multiple groups. Read-only subjects are useful for the purpose of aggregation of information from multiple groups (for instance, to view an aggregated calendar or messages from multiple groups) and would be relatively straightforward to support.

Authorization for the basic group operations is out of the current scope, although a complete policy must also specify the authorization for these operations. We expect that administrative models for g-SIS can be adapted from the numerous administrative models in the literature [Harrison et al. 1976; Jaeger and Tidswell 2001; Li et al. 2002; Lipton and Snyder 1977; Sandhu 1988a, 1992; Sandhu et al. 1999].

Our contributions are as follows in correspondence to the outline of the rest of this article. In general, our contributions demonstrate the translation of intuitive aspects of a model, such as the isolated g-SIS, into a formal model. We define a collection of intuitive core properties of isolated g-SIS using a language that is based on first-order linear temporal logic or FOTL (Section 3). The core properties are required of all g-SIS specifications. We propose a number of optional properties that are based on specific semantics of group operations. We develop a stateless g-SIS specification using FOTL, called the π-system, that supports a selected subset of such operation semantics (Section 4). We prove that the π-system entails the core properties and selected optional properties (Section 5). We also prove that the core properties are mutually independent and consistent. Our proof technique involves generalizing a proof obtained for small carriers (with limited number of users, objects, etc.) to large carriers (with countably infinite users, objects, etc.). We use FOTL to specify g-SIS with unbounded carriers and the theorems we state in this article are difficult to prove using completely manual techniques. Therefore, we use a model checker to obtain a proof for the small carrier case and extend that proof to the general unbounded carrier case. Section 6 discusses related work, and Section 7 gives our conclusions.

3. FORMAL SPECIFICATION OF G-SIS

We provide an informal overview of the design features of isolated g-SIS and formalize the model in the following subsections. The specific design choices that have been made are not always as important as the demonstration of the existence of a number of choices. We define the g-SIS language and specify a core set of properties that are required of all g-SIS specifications. We also discuss a few optional properties based on different semantics of some of the group operations. From here on, we understand that the terms g-SIS model and g-SIS specification refer to isolated g-SIS model and specification, respectively.

3.1. Overview

As discussed in Section 2.3, the focus of this article is on the basic group operations of join, leave, create, add, and remove. We follow a simple yet flexible object versioning model where each update operation creates a new version. Note that the update operation can be performed on a specific version (see Table I) of an object. Also, the same version can be updated multiple times creating multiple versions from the same parent version. We assume that the root version of an object has a special version identifier called $\mathbf{v}_{\text{init}}$ that is generated when the object is created. Different versions of an object may be added to the group from an external entity, but group users are not allowed to write to such versions. Objects created locally in the group may be read and updated. Note that it is possible for a subject to copy the content of an externally added object to a locally created object. Thus, once information is brought into a group, strictly speaking it cannot be guaranteed to be removed since it may have been copied

Table I. Summary of Operations

Operation	Explanation
Join (u, g) = join_1 $(u, g) \vee \ldots \vee \text{join}_m$ (u, g)	Join user u to group g.
Leave (u, g) = leave_1 $(u, g) \vee \ldots \vee \text{leave}_n$ (u, g)	Leave user u from group g.
Add (o, v, g) = add_1 $(o, v, g) \vee \ldots \vee \text{add}_p$ (o, v, g)	Add version v of object o to group g.
Remove (o, v, g) = remove_1 $(o, v, g) \vee \ldots \vee$ remove_q (o, v, g)	Remove version v of object o from group g.
CreateO $(o, \mathbf{v}_{\text{init}}, g)$ = createO_1 $(o, \mathbf{v}_{\text{init}}, g) \vee \ldots \vee$ createO_r $(o, \mathbf{v}_{\text{init}}, g)$	Create root version $\mathbf{v}_{\text{init}}$ of object o in group g.
createS (u, s, g)	User u creates a subject s in group g.
kill (u, s, g)	User u kills a subject s that she had created in g.
read (s, o, v, g)	Subject s reads version v of object o in group g.
update (s, o, v_1, v_2, g)	Subject s updates version v_1 of object o in group g resulting in a new version v_2 in g.

into locally created objects. For this article, the crucial information flow property of an isolated group is that information cannot flow out from the group except through human channels. More detailed information flow analysis is deferred to future work (see Section 7).

A summary of operations supported in the model is shown in Table I. As mentioned earlier, group operations in g-SIS may have different semantics. The subscripts in Table I denote these semantics for respective operations. For example, each of join_1 (u, g), ..., join_m (u, g) represent specific join semantics such as strict, liberal, etc. This will be discussed later in this section. The operations shown below the horizontal line, createS, kill, read and update, do not have variations and are carried out by users or subjects explicitly identified in the operation parameters. Operations shown above the horizontal line are carried by administrative action. The administrator who initiates these operations is not explicitly identified in the operation parameters. Identification of the administrative user would become relevant in context of an administrative model controlling administrative authorization. For simplicity we have included CreateO in this set, although arguably it could have been included in the user or subject initiated operations below the line.

3.2. g-SIS Language

We use many sorted, first-order linear temporal logic (FOTL) [Pnueli 1977] to characterize g-SIS. FOTL differs from the familiar propositional linear temporal logic by incorporating predicates with parameters, constants, variables, and quantifiers. We need these to specify systems that contain unbounded numbers of groups, users, subjects, objects, and object versions.

Sorts are syntactic objects that resemble types. For users, subjects, objects, versions, and groups, respectively, we denote variables and sorts by $g : \text{G}$, $u : \text{U}$, $s : \text{S}$, $o : \text{O}$ and $v : \text{V}$. The semantic values over which a variable ranges depend on the variable's sort and are drawn from a set that is called the *carrier* of that sort. For each of the sorts identified above we denote the corresponding carrier by $\mathcal{G}$, $\mathcal{U}$, $\mathcal{S}$, $\mathcal{O}$ and $\mathcal{V}$, respectively. We use the following metavariables to range over individuals: $\mathbf{g} \in \mathcal{G}$, $\mathbf{u} \in \mathcal{U}$, $\mathbf{s} \in \mathcal{S}$, $\mathbf{o} \in \mathcal{O}$ and $\mathbf{v} \in \mathcal{V}$. $\mathcal{V}$ is required to include a distinguished element $\mathbf{v}_{\text{init}}$. When needed, we add subscripts to variables and metavariables. The carriers introduced above are assumed to be countably infinite, enabling us to model systems of unbounded finite size. In later sections, when we discuss using a model checker to verify properties of g-SIS systems, we will also introduce small finite carriers, $\mathcal{G}_{\text{sm}}$, $\mathcal{U}_{\text{sm}}$, $\mathcal{O}_{\text{sm}}$, $\mathcal{V}_{\text{sm}}$, which the

model checker can effectively analyze. In addition, we have p:P with carrier $\mathcal{P} = \{\mathbf{r}, \mathbf{w}\}$ ranging over the permissions to read and write. We denote the collection of carriers by $\mathcal{C} = \langle \mathcal{G}, \mathcal{U}, \mathcal{S}, \mathcal{O}, \mathcal{V}, \mathcal{P} \rangle$ and $\mathcal{C}_{\text{sm}} = \langle \mathcal{G}_{\text{sm}}, \mathcal{U}_{\text{sm}}, \mathcal{S}_{\text{sm}}, \mathcal{O}_{\text{sm}}, \mathcal{V}_{\text{sm}}, \mathcal{P} \rangle$.

Actions are represented by predicates, so the language of formulas depends on the actions that are supported. A *state* is an interpretation, ι, that maps each predicate in the language to a relation over the appropriate carriers. For instance, an action, join_i, is mapped to $[\![\text{join}_i]\!]\iota \subset \mathcal{U} \times \mathcal{G}$. Predicates that represent actions are called *action predicates*. The action predicates we consider are those shown as operations in Table I. In addition, the language has two *authorization predicates* that can be used to construct atomic formulas such as $\text{Authz}(u, o, \upsilon, g, p)$ and $\text{AuthzS}(s, o, \upsilon, g, p)$. We denote the set of states over carriers $\mathcal{C}$ by $\Sigma_{\mathcal{C}}$. A trace is an infinite sequence of states $\sigma \in \Sigma_{\mathcal{C}}^{\omega}$. At a given state σ_i ($i \in \mathbb{N}$), $\langle \mathbf{u}, \mathbf{g} \rangle \in [\![\text{join}_j]\!]\sigma_i$ indicates that user $\mathbf{u}$ joins group $\mathbf{g}$ at position i with the variant of the join operation given by join_j. Similarly, $\langle \mathbf{s}, \mathbf{o}, \mathbf{v}, \mathbf{g}, \mathbf{w} \rangle \in [\![\text{AuthzS}]\!]\sigma_i$ indicates that subject $\mathbf{s}$ is authorized to write/update ($\mathbf{w}$) version $\mathbf{v}$ of object $\mathbf{o}$ in group $\mathbf{g}$ at position i. (Also, $\langle \mathbf{u}, \mathbf{o}, \mathbf{v}, \mathbf{g}, \mathbf{w} \rangle \in [\![\text{Authz}]\!]\sigma_i$ indicates that user $\mathbf{u}$ is authorized to write/update ($\mathbf{w}$) version $\mathbf{v}$ of object $\mathbf{o}$ in group $\mathbf{g}$ at position i.) Notice that if each carrier in $\mathcal{C}_{\text{sm}}$ is a subset of the corresponding carrier in $\mathcal{C}$, then $\sigma \in \Sigma_{\mathcal{C}_{\text{sm}}}^{\omega}$ implies $\sigma \in \Sigma_{\mathcal{C}}^{\omega}$.

FOTL formulas can be classified as either state formulas or temporal formulas. *State formulas* are constructed from atomic formulas, possibly with variables, constant symbols, logical connectives, and quantifiers over variables. Our constant symbols are identified with distinguished elements of the carrier corresponding to the constant's sort. Thus, in particular, $\mathbf{v}_{\text{init}}$, $\mathbf{r}$ and $\mathbf{w}$ can appear in formulas. *Temporal formulas* are constructed from FOTL formulas by applying temporal operators. An intuitive summary of the temporal operators used in this article is given in Table VII in Appendix A. (All appendices of this article are available in the electronic version.)

An *environment* η is a function mapping variables to elements of carriers in which the variable's sort dictates which carrier its value is drawn from. For instance, $\eta(\upsilon) = \mathbf{v} \in \mathcal{V}$ represents the notion of variable υ standing for $\mathbf{v}$ in the current environment. (Environments are needed to provide semantics for formulas containing quantifiers.) For a formula ϕ, ϕ is satisfied by a trace σ, written $\sigma \models \phi$, if $\sigma, 0, \eta \models \phi$ for all environments η. The statement that, at index i, σ satisfies ϕ under environment η, written $\sigma, i, \eta \models \phi$, is defined inductively on the structure of ϕ. The following highlights this inductive construction while suppressing detailed treatment of sorts for simplicity. By $[\![t]\!]\eta$ we denote $\eta(t)$ when t is a variable and the element of the carrier denoted by t, when t is a constant. For n-ary predicate q and terms (*i.e.*, constants or variables) $t_1, \ldots, t_n$ of the appropriate sorts, $\sigma, i, \eta \models q(t_1, \ldots, t_n)$ holds when $\langle [\![t_1]\!]\eta, \ldots, [\![t_n]\!]\eta \rangle \in [\![q]\!]\sigma_i$. Compound state formulas are handled in the usual way. In particular, $\sigma, i, \eta \models \exists x : \tau.\phi$ holds when $\sigma, i, \eta[x \mapsto a] \models \phi$ holds for some a in the carrier for sort τ. ($\eta' = \eta[x \mapsto a]$ is the environment given by $\eta'(y) = a$ if y is the variable x, $\eta(y)$ otherwise.) Temporal operators are handled as follows. We have ("unless") $\sigma, i, \eta \models \phi_1 \mathcal{W} \phi_2$ if there exists $k \geq i$ such that $\sigma, k, \eta \models \phi_2$ and for all j, $i \leq j < k$ implies $\sigma, j, \eta \models \phi_1$, otherwise for all i $\sigma, j, \eta \models \phi_1$. We also have ("since") $\sigma, i, \eta \models \phi_1 \mathcal{S} \phi_2$ when there exists $k \leq i$ such that $\sigma, k, \eta \models \phi_2$ and for all j, $k < j \leq i$ implies $\sigma, j, \eta \models \phi_1$. Finally, ("next") $\sigma, i, \eta \models \bigcirc\phi$ when $\sigma, i+1, \eta \models \phi$ and ("previous") $\sigma, i, \eta \models \ominus\phi$ when $\sigma, i-1, \eta \models \phi$. Other temporal operators can be defined in terms of the above operators. Operators $\ominus$, $\diamondsuit$, and $\mathcal{S}$ are *past operators*; $\bigcirc$, $\Box$, and $\mathcal{W}$ are *future operators*.

Because we are studying the relationship between semantics of various basic group operations, the language of the specification includes several different predicates representing each of the five basic operations join, leave, add, remove and create. In the

following, we often wish to write subformulas that state, for example, some type of join event occurs. It is therefore convenient to introduce the notations Join, Leave, Add, Remove, and CreateO as shown in Table I. In addition, we have two authorization predicates Authz and AuthzS.

3.3. Well-Formed Traces

The well-formed traces constraints specify the syntactic correctness of g-SIS traces. We highlight a few aspects of well-formedness by explaining parts of the formulas below. Other parts should be self-explanatory.

(1) Formula τ_0 specifies operations that cannot simultaneously occur in the same state.

$$\begin{aligned}
&\tau_0(u_1, s_1, s_2, o, v_1, v_2, v_3, g_1) = \\
&\quad \Box\neg(\text{Join}(u_1, g_1) \wedge \text{Leave}(u_1, g_1)) \wedge && (1)\\
&\quad \Box\neg(\text{Add}(o, v_1, g_1) \wedge \text{Remove}(o, v_1, g_1)) \wedge && (2)\\
&\quad \Box\neg(\text{createS}(u_1, s_1, g_1) \wedge (\text{kill}(u_1, s_1, g_1) \vee && (3)\\
&\quad \text{read}(s_1, o, v_3, g_1) \vee \text{update}(s_1, o, v_1, v_2, g_1))) \wedge \\
&\quad \Box\neg(\text{update}(s_1, o, v_1, v_2, g_1) \wedge && (4)\\
&\quad (\text{read}(s_2, o, v_2, g_1) \vee \text{update}(s_2, o, v_2, v_3, g_1))) \wedge \\
&\quad \Box\neg(\text{CreateO}(o, \mathbf{v}_{\text{init}}, g_1) \wedge (\text{update}(s_1, o, \mathbf{v}_{\text{init}}, v_1, g_1))) \wedge && (5)\\
&\quad \Box\neg(\text{Add}(o, v_1, g_1) \wedge \text{read}(s_1, o, v_1, g_1)) && (6)
\end{aligned}$$

It states that a user cannot join and leave in the same state (1). Similarly an object version cannot be added and removed in the same state (2). A subject cannot be created and simultaneously killed. Also the subject cannot perform read or update operations in the same state in which it was created (3). Also note that a version being updated to say v_2 cannot be simultaneously read or further updated by any subject in the same state (4). Finally, an object cannot be created and updated in the same state (5). Similarly, an object may not be added and read in the same state (6). Recall that we assume that objects added from external sources cannot be updated. In general, we require that in the same state an entity cannot be introduced into the system and operated upon.

(2) Formula τ_1 states that two types of the same operation cannot occur in the same state for any given user or object. For instance, for a given user and group, two types of join operation (such as strict and liberal) cannot occur in the same state. Note that subscripts such as i and j are not part of our language but are metavariables used to refer to different types of an operation.

$$\begin{aligned}
&\tau_1(u_1, o, v_1, g_1, g_2) = \\
&\quad \bigwedge\{\Box\neg(\text{join}_i(u_1, g_1) \wedge \text{join}_j(u_1, g_1)) | 1 \le i < j \le m\} \wedge \\
&\quad \bigwedge\{\Box\neg(\text{leave}_i(u_1, g_1) \wedge \text{leave}_j(u_1, g_1)) | 1 \le i < j \le n\} \wedge \\
&\quad \bigwedge\{\Box\neg(\text{add}_i(o, v_1, g_1) \wedge \text{add}_j(o, v_1, g_1)) | 1 \le i < j \le p\} \wedge \\
&\quad \bigwedge\{\Box\neg(\text{remove}_i(o, v_1, g_1) \wedge \text{remove}_j(o, v_1, g_1)) | 1 \le i < j \le q\} \wedge \\
&\quad \bigwedge\{\Box\neg(\text{createO}_i(o, v_1, g_1) \wedge \text{createO}_j(o, v_1, g_2)) | 1 \le i < j \le r\}
\end{aligned}$$

(3) Formula τ_2 specifies the operations that cannot reoccur. Intuitively, these constraints require that anything that is introduced in the system has a unique

identifier of the respective sort. For instance, an object version can never be re-created.

$$\begin{aligned}
&\tau_2(u_1, u_2, s_1, s_2, o, v_1, v_2, v_3, g_1, g_2) = \\
&\quad \Box(\text{CreateO}(o, \mathbf{v}_{\text{init}}, g_1) \rightarrow \bigcirc(\Box\neg\text{CreateO}(o, \mathbf{v}_{\text{init}}, g_2)))\wedge \\
&\quad \Box(\text{createS}(u_1, s_1, g_1) \rightarrow \bigcirc(\Box\neg\text{createS}(u_2, s_1, g_2)))\wedge \\
&\quad \Box(\text{update}(s_1, o, v_1, v_2, g_1) \rightarrow \bigcirc(\Box\neg\text{update}(s_2, o, v_3, v_2, g_1)))\wedge \\
&\quad \Box(\text{CreateO}(o, \mathbf{v}_{\text{init}}, g_1) \rightarrow \bigcirc\Box\neg\text{Add}(o, v, g_2))\wedge \\
&\quad \Box(\text{Add}(o, v, g_1) \rightarrow \bigcirc\Box\neg\text{CreateO}(o, \mathbf{v}_{\text{init}}, g_2))\wedge \\
&\quad \Box(\text{Add}(o, v_1, g_1) \rightarrow \Box\neg\text{update}(s_1, o, v_1, v_2, g_1))\wedge \\
&\quad \Box\neg(\text{CreateO}(o, v_1, g_1) \wedge (v_1 \neq \mathbf{v}_{\text{init}}))\wedge \\
&\quad \Box\neg(\text{update}(s_1, o, v_1, v_2, g_1) \wedge (v_2 = \mathbf{v}_{\text{init}}))
\end{aligned}$$

(4) Formula τ_3 specifies the order in which certain operations of the same sort may occur. For example, once a user joins, she may rejoin only after leaving the group. In order to a leave a group, the user should not have left since she most recently joined.

$$\begin{aligned}
&\tau_3(u_1, s_1, o, v_1, g_1) = \\
&\quad \Box(\text{Join}(u_1, g_1) \rightarrow \bigcirc(\neg\text{Join}(u_1, g_1)\,\mathcal{W}\,\text{Leave}(u_1, g_1)))\wedge \\
&\quad \Box(\text{Leave}(u_1, g_1) \rightarrow \ominus(\neg\text{Leave}(u_1, g_1)\,\mathcal{S}\,\text{Join}(u_1, g_1)))\wedge \\
&\quad \Box(\text{Add}(o, v_1, g_1) \rightarrow \bigcirc(\neg\text{Add}(o, v_1, g_1)\,\mathcal{W}\,\text{Remove}(o, v_1, g_1)))\wedge \\
&\quad \Box(\text{Remove}(o, v_1, g_1) \rightarrow \ominus(\neg\text{Remove}(o, v_1, g_1)\,\mathcal{S}\,\text{Add}(o, v_1, g_1))\wedge \\
&\quad \Box(\text{kill}(u_1, s_1, g_1) \rightarrow \ominus(\neg\text{kill}(u_1, s_1, g_1)\,\mathcal{S}\,\text{createS}(u_1, s_1, g_1)))
\end{aligned}$$

We denote the well-formed constraints collectively as θ.

$$\begin{aligned}
\theta = &\forall u_1, u_2 : \text{U}.\forall s_1, s_2 : \text{S}.\forall o : \text{O}.\forall v_1, v_2, v_3 : \text{V}.\forall g_1, g_2 : \text{G}. \\
&\tau_0(u_1, s_1, s_2, o, v_1, v_2, v_3, g_1) \wedge \tau_1(u_1, o, v_1, g_1, g_2) \wedge \\
&\tau_2(u_1, u_2, s_1, s_2, o, v_1, v_2, v_3, g_1, g_2) \wedge \tau_3(u_1, s_1, o, v_1, g_1)
\end{aligned}$$

3.4. Core Properties

The core properties govern how authorization may change in isolated g-SIS.[2] As will be defined formally later, these properties are required of all isolated g-SIS specifications. In addition to characterizing what we mean by saying that a model is an isolated g-SIS model, core properties provide an opportunity to perform a sort of sanity check on a specification as a whole. Core properties are designed to focus on one global system property at a time. This is helpful because when designing the system specification, it is easy to miss important details. Designing and verifying core properties gives the system designer an opportunity to focus on one system characteristic at a time and to view the specification from various points of view.

The motivation for these core properties is as follows. A group policy in traditional operating systems such as in UNIX is fixed and simple. A user may access an object

[2]Each of these properties signifies an important aspect of a group in g-SIS. We believe that the precise formulation of these properties will change as g-SIS models evolve (for instance, with the introduction of additional basic group operations). However, we expect newer formulations of these properties would still accommodate the intuitive core aspects of g-SIS presented here. Also note that each of these properties defines a *safety* property [Lamport 1977] in the sense that any trace that does not satisfy the property can be recognized as such by examining a finite prefix of the trace.

if both the user and the object are current members of the group. Such a rigid definition is very restrictive in the larger information sharing scenario. Let us consider two distinct application domains to illustrate this assertion. Consider a group managed by an organization whose users include members of the same and some external consultants. In certain scenarios, it is likely that once the consultant leaves the group, she may retain some level of access to objects that she was authorized to access during her membership period. Furthermore, the same consultant may re-visit the group sometime in the near future. At this point, what is an appropriate authorization policy for her? Is she allowed to access every object in the group? If not, what about the objects she had access to during her prior membership period? In contrast to this example, consider a typical group policy of forward and backward secrecy in the secure multicast scenario [Rafaeli and Hutchison 2003]. A joining user may only access information added after join time. Past information is not available to this new user (backward secrecy). A leaving user cannot access new information added to the group after leave time (forward secrecy).

Clearly, the semantics of group operations such as join, leave, create, add and remove (that is the authorizations enabled by such operations) is application dependant and a model for group-centric information sharing should be flexible enough to accommodate different semantics. Specifically, in certain scenarios, a group policy may be very strict in terms of authorization (such as in secure multicast example above), while in others, it may be liberal (such as in the consultant example). The core properties of the g-SIS model specify various aspects of the authorization requirements of a reasonably liberal group. Specifically, the properties allow for different policy specifications such as the examples previously considered to be specified.

(1) *Persistence Properties.* These properties specify the conditions under which authorization may change in g-SIS. In general, authorization value changes in a system only if an authorization changing event occurs.
Authorization Persistence. Formula κ_0 states that when a user is authorized to read a version via group g, she remains authorized unless some membership changing event involving the user or version occurs. For instance, if a user is currently authorized and she leaves the group, she may no longer be authorized. In certain scenarios, a past member of the group may have authorization to read a version but may lose access when she rejoins the group. We call this a *lossy* join. This property states that such events can change the value of authorization predicate for read. We consider some example authorization semantics for group operations in Sections 3.5 and 3.6.

$$\kappa_0 = \forall u : \mathrm{U}.\forall o : \mathrm{O}.\forall v : \mathrm{V}.\forall g : \mathrm{G}.\ \Box(\mathrm{Authz}(u, o, v, g, \mathbf{r}) \rightarrow (\mathrm{Authz}(u, o, v, g, \mathbf{r})\, \mathcal{W}\, (\mathrm{Join}(u, g) \vee \mathrm{Leave}(u, g) \vee \mathrm{Add}(o, v, g) \vee \mathrm{Remove}(o, v, g))))$$

Formula κ_1 states that if a user is authorized to write to a version, she remains authorized unless she leaves. As we will see, the Authorization Provenance core property requires that in order to write any version, the user needs to be a current member. Thus unlike read, authorization to write cannot hold for past members. Further, users cannot write to versions added from external sources. Thus Leave is the only relevant operation.

$$\kappa_1 = \forall u : \mathrm{U}.\forall o : \mathrm{O}.\forall v : \mathrm{V}.\forall g : \mathrm{G}.\ \Box(\mathrm{Authz}(u, o, v, g, \mathbf{w}) \rightarrow (\mathrm{Authz}(u, o, v, g, \mathbf{w})\, \mathcal{W}\, \mathrm{Leave}(u, g)))$$

Revocation Persistence. If a user is not authorized to read a version, she will remain so unless some authorization changing event occurs. For instance, she can

become authorized if she joins the group or if the version is introduced through Add, CreateO or update.

$$\kappa_2 = \forall u : \mathrm{U}.\forall o : \mathrm{O}.\forall v_1 : \mathrm{V}.\forall g : \mathrm{G}.\exists s : \mathrm{S}.\exists v_2 : \mathrm{V}.$$
$$\Box(\neg\mathrm{Authz}(u, o, v_1, g, \mathbf{r}) \rightarrow (\neg\mathrm{Authz}(u, o, v_1, g, \mathbf{r})\, \mathcal{W}\, (\mathrm{Join}(u, g) \vee$$
$$\mathrm{Leave}(u, g) \vee \mathrm{Add}(o, v_1, g) \vee \mathrm{Remove}(o, v_1, g) \vee$$
$$\mathrm{CreateO}(o, v_1, g) \vee \mathrm{update}(s, o, v_2, v_1, g))))$$

Formula κ_3 states that if a user is not authorized to write to a version, she remains so unless she joins the group, or the version is introduced through CreateO or update. Note that externally added objects cannot be updated.

$$\kappa_3 = \forall u : \mathrm{U}.\forall o : \mathrm{O}.\forall v_1 : \mathrm{V}.\forall g : \mathrm{G}.\exists s : \mathrm{S}.\exists v_2 : \mathrm{V}.$$
$$\Box(\neg\mathrm{Authz}(u, o, v_1, g, \mathbf{w}) \rightarrow (\neg\mathrm{Authz}(u, o, v_1, g, \mathbf{w})\, \mathcal{W}\, (\mathrm{Join}(u, g) \vee$$
$$\mathrm{CreateO}(o, v_1, g) \vee \mathrm{update}(s, o, v_2, v_1, g))))$$

(2) *Authorization Provenance.* This property specifies the conditions under which read or write authorization may begin to hold in a given trace. The g-SIS theory recognizes that in certain scenarios past users may retain read access to some of the object versions in the group (such as in the consultant example earlier). Specifically authorization to read a version may hold even during non-membership periods of users. Intuitively, property κ_4 states that a user will not be authorized to read a version until a point in a trace at which both the user and the object version are simultaneously members of the same group. Note that the version may become a member of the group by one of the following means: (a) the version is added to the group from an external source, (b) the version is created in the group or (c) the version is introduced by means of an update operation on some prior version in the group.

$$\kappa_4 = \forall u : \mathrm{U}.\forall o : \mathrm{O}.\forall v_1 : \mathrm{V}.\forall g : \mathrm{G}.\exists s : \mathrm{S}.\exists v_2 : \mathrm{V}.$$
$$(\neg\mathrm{Authz}(u, o, v_1, g, \mathbf{r})\, \mathcal{W}\, (\mathrm{Authz}(u, o, v_1, g, \mathbf{r}) \wedge$$
$$(\neg\mathrm{Leave}(u, g)\, \mathcal{S}\, \mathrm{Join}(u, g)) \wedge ((\neg\mathrm{Remove}(o, v_1, g)\, \mathcal{S}\, \mathrm{Add}(o, v_1, g)) \vee$$
$$((\diamondminus\, \mathrm{CreateO}(o, v_1, g) \vee \diamondminus\, \mathrm{update}(s, o, v_2, v_1, g))))))$$

Two important observations can be made on formula κ_4. First, if authorization to read a version holds in some state, then there was an overlapping period of membership between the user and version at least once either in the current state or in some prior state. Second, authorization to read a version cannot begin to hold for the first time during a user's nonmembership period.

Unlike read, we require that only current users be authorized to write to any existing version as stated in formula κ_5 below. Further, in order to write to a version, the user should be authorized to read that version.[3]

$$\kappa_5 = \forall u : \mathrm{U}.\forall o : \mathrm{O}.\forall v_1 : \mathrm{V}.\forall g : \mathrm{G}.\exists s : \mathrm{S}.\exists v_2 : \mathrm{V}.$$
$$\Box(\mathrm{Authz}(u, o, v_1, g, \mathbf{w}) \rightarrow\ (\mathrm{Authz}(u, o, v_1, g, \mathbf{r}) \wedge (\neg\mathrm{Leave}(u, g)\, \mathcal{S}$$
$$\mathrm{Join}(u, g)) \wedge (\diamondminus\, \mathrm{CreateO}(o, v_1, g) \vee \diamondminus\, \mathrm{update}(s, o, v_2, v_1, g))))$$

(3) *Bounded Authorization.* These properties bound the set of object versions that a user is authorized to read when she leaves a group and the set of users authorized to read a version when it is removed from a group.

[3]As noted earlier, these are design choices that have been made out of the number of other alternatives that exist. For instance, a scenario in which past users are allowed to update versions that they are authorized to read can be easily envisioned. But such specifications are outside the realm of the current g-SIS model.

Bounded User Authorization. Formula κ_6 states that if a user is not authorized to read a version at the time of leave, she will remain so unless she rejoins the group.

$$\begin{aligned}\kappa_6 =& \forall u : \mathrm{U}.\forall o : \mathrm{O}.\forall v : \mathrm{V}.\forall g : \mathrm{G}.\\ & \Box((\mathrm{Leave}(u, g) \wedge \neg\mathrm{Authz}(u, o, v, g, \mathbf{r})) \rightarrow\\ & (\neg\mathrm{Authz}(u, o, v, g, \mathbf{r})\, \mathcal{W}\, \mathrm{Join}(u, g)))\end{aligned}$$

That is if the user is authorized to read an object after leaving the group, the authorization should have held at the time of leave. Note that if authorization to read a version held at the time of leave, it need not hold during the user's nonmembership period.

Bounded Object Authorization. Formula κ_7 states that if a user is not authorized to read a version at the time of remove, she will remain so unless the version is readded.

$$\begin{aligned}\kappa_7 =& \forall u : \mathrm{U}.\forall o : \mathrm{O}.\forall v : \mathrm{V}.\forall g : \mathrm{G}.\\ & \Box((\mathrm{Remove}(o, v, g) \wedge \neg\mathrm{Authz}(u, o, v, g, \mathbf{r})) \rightarrow\\ & (\neg\mathrm{Authz}(u, o, v, g, \mathbf{r})\, \mathcal{W}\, \mathrm{Add}(o, v, g)))\end{aligned}$$

(4) *Version Authorization Uniformity:* This property states that for a locally created object in a group, a current user in the group should be authorized to read or write either all versions of the object or none of them.

$$\begin{aligned}\kappa_8 =& \forall u : \mathrm{U}.\forall o : \mathrm{O}.\forall v_1, v_2, v_3 : \mathrm{V}.\forall g : \mathrm{G}.\forall p : \mathrm{P}.\exists s : \mathrm{S}.\\ & \Box((\neg\mathrm{Leave}(u, g)\, \mathcal{S}\, \mathrm{Join}(u, g)) \wedge \mathrm{Authz}(u, o, v_1, g, p) \wedge\\ & \diamondminus(\mathrm{update}(s, o, v_3, v_2, g) \vee \mathrm{CreateO}(o, v_2, g)) \rightarrow\ \mathrm{Authz}(u, o, v_2, g, p))\end{aligned}$$

This property essentially specifies the authorization dependency between different versions of the object in the group. For example, in a scenario where a user rejoins the group with authorizations to a few versions of an object from prior membership period, this property requires that on rejoin she be authorized to read all versions of that object or none. We expect that this property will be modified as per the requirements of the application. For instance, it may be desirable in some applications that all users are only able to read the most recent version of an object. Therefore, this core property will vary from one versioning model to another.

Definition 3.1 *Semantic Correctness*. A well-formed g-SIS specification γ is semantically correct if:

$$\gamma \wedge \theta \vDash \kappa_0 \wedge \ldots \wedge \kappa_8.$$

That is, a well-formed (θ) g-SIS specification (γ) is semantically correct if it satisfies all of the core properties.

3.5. Membership Properties

In this section and the following, we discuss some example authorization semantics for certain group operations. We formulate such authorization semantics as secondary properties. That is, unlike the core, these properties are not required of all g-SIS specifications. Instead, they define operation semantics that are useful in many application scenarios (e.g., secure multicast) and may be selectively mandated.

Membership Properties characterize the semantics of authorizations enabled when a user joins or a version is added or created and those which are disabled when a user

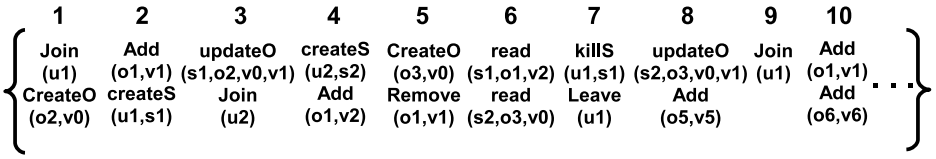


Fig. 2. A sample g-SIS trace.

Table II. Summary of Group Membership Semantics

Operation	Explanation	Property
Strict Join	Versions admitted after join time authorized	$\alpha_0 = \forall u : \mathrm{U}.\forall o : \mathrm{O}.\forall v : \mathrm{V}.\forall g : \mathrm{G}.$ $\Box(\mathrm{Authz}(u, o, v, g, \mathbf{r}) \rightarrow \ \Diamond((\mathrm{Add}(o, v, g) \vee \mathrm{CreateO}(o, \mathbf{v}_{\mathrm{init}}, g)) \wedge (\neg \mathrm{Leave}(u, g)\, \mathcal{S}\, \mathrm{join}_i(u, g))))$
Strict Leave	Lose complete access	$\alpha_1 = \forall u : \mathrm{U}.\forall o : \mathrm{O}.\forall v : \mathrm{V}.\forall g : \mathrm{G}.$ $\Box(\mathrm{Authz}(u, o, v, g, \mathbf{r}) \rightarrow \ (\neg \mathrm{leave}_i(u, g)\, \mathcal{S}\, \mathrm{Join}(u, g)))$
Strict Add/Create	Users who joined before version admission time authorized	$\alpha_2 = \forall u : \mathrm{U}.\forall o : \mathrm{O}.\forall v : \mathrm{V}.\forall g : \mathrm{G}.$ $\Box((\mathrm{add}_i(o, v, g) \wedge \neg\Diamond \mathrm{Join}(u, g)) \rightarrow (\neg \mathrm{Authz}(u, o, v, g, \mathbf{r})\, \mathcal{W}\, \mathrm{Add}(o, v, g))) \wedge$ $\Box((\mathrm{createO}_i(o, \mathbf{v}_{\mathrm{init}}, g) \wedge \neg\Diamond \mathrm{Join}(u, g)) \rightarrow \Box\neg \mathrm{Authz}(u, o, \mathbf{v}_{\mathrm{init}}, g, \mathbf{r}))$
Strict Remove	All users lose access	$\alpha_3 = \forall u : \mathrm{U}.\forall o : \mathrm{O}.\forall v : \mathrm{V}.\forall g : \mathrm{G}.$ $\Box(\mathrm{remove}_i(o, v, g) \rightarrow \ (\neg \mathrm{Authz}(u, o, v, g, \mathbf{r})\, \mathcal{W}\, \mathrm{Add}(o, v, g)))$

leaves or a version is removed from the group. In the following subsection, we consider properties when a user or a version is readmitted. We use the term "admitted" to refer to object versions that are introduced to the group either through an Add or CreateO operation.

Figure 2 shows the first 10 states of a sample well-formed g-SIS trace. We omit the group parameter in all of these operations since they are all on the same group. Also we denote version v of object o as (o,v). Consider user $u1$ who joins in state 1 and creates a subject $s1$ in state 2. In state 3, $s1$ updates $(o2,v0)$, created in state 1, to a new version $v1$. In state 6, $s1$ also reads $(o1,v2)$ which was added in state 4. User u then kills $s1$ in state 7 and leaves the group and subsequently re-joins in state 9. In the mean time, in state 8, user $u2$ who joined in state 3 updates $(o3,v0)$ to $v1$ using her subject $s2$.

*Strict Join (*SJ*) vs Liberal Join (*LJ*).* In SJ, the joining user may only read some or all of the versions admitted after join time. LJ additionally allows the user to read some or all of the versions that were admitted prior to join time. Suppose that in Figure 2, $u2$ joins with SJ in state 3. Then $u2$ would not be authorized to read $(o1,v1)$ added in state 2. Furthermore created objects and their derivative versions introduced by update operations are also not authorized. For instance, $u2$ would not be authorized to read $(o2,v0)$, created in state 1, and $(o2,v1)$, introduced by update in state 3. This can be formalized by requiring that join_i, a type of Join, has the semantics of SJ only if it satisfies α_0 stated in Table II. In a g-SIS specification that supports only joins of type

LJ, there exists at least one well-formed trace that does not satisfy α_0. Evidently, SJ has the same semantics as backward secrecy in secure multicast applications (when applied to all objects) while LJ has the same join semantics of groups in traditional operating systems.

Strict Leave (SL) vs. Liberal Leave (LL). In SL, the leaving user loses read access to all versions. In LL, the leaving user may retain read access to some or all of the versions authorized prior to Leave time. In Figure 2, on SL in state 7, $u1$ loses read access to all versions. In case of LL, $u1$ may retain read access to ($o2$,$v0$), ($o2$,$v1$), ($o1$,$v2$) and ($o3$,$v0$) during non-membership period (states 7 and 8). (Also ($o1$,$v1$) if the remove operation in state 5 has appropriate semantics.) Note that $u2$ would not be authorized to read ($o3$,$v1$) introduced by update after leave time by $s2$ in state 8. A type of Leave, leave$_i$, has the semantics of SL only if it satisfies α_1 stated in Table II. In a g-SIS specification that only supports leaves of type LL, there exists at least one well-formed trace that does not satisfy α_1. Again, the LL operation has the same semantics as forward secrecy in secure multicast applications (when applied to all objects) while SL has the same leave semantics of traditional groups.

*Strict Add/Create (*SA/SC*) vs. Liberal Add/Create (*LA/LC*).* On SA/SC, some or all of the users who joined prior to admission time of the version may read. On LA/LC, the admitted version may also be read by some or all of the users that join (e.g., LJ) later. In figure 2, if ($o2$,$v0$) was created with SC in state 1, $u2$ joining later in state 3 would not be authorized to read. Note that $u2$ would also be not authorized to read all derivative versions of ($o2$,$v0$) such as ($o2$,$v1$). In case of LC of ($o2$,$v0$), $u2$ may be authorized to read ($o2$,$v0$) depending on the semantics of join. If $u2$ joins with SJ, she would not be authorized to read ($o2$,$v0$) and its derivative versions. A type of Add (CreateO), add$_i$ (createO$_i$), would be admitted as SA (SC) only if it satisfies α_2 stated in Table II. In a g-SIS specification that only supports adds (creates) of type LA (LC), there exists at least one well-formed trace that does not satisfy α_2.

Traditionally, add semantics for objects have been liberal. In certain information sharing scenarios, it is desirable that the owner of the information who is sharing an object with the group may need to specify which users in the group may access that information in the future. In the consultant example earlier, it may be desirable that the objects being brought into the group by the organization be accessible only to existing consultants in the group. The strict add semantics automatically disqualifies consultants joining the group in the future from accessing that object. If consultants joining in the future need access to that object, a different add semantics may facilitate that process. Thus strict add and create semantics enrich previously held notions of authorization semantics for object operations.

*Strict Remove (*SR*) vs. Liberal Remove (*LR*).* In SR, the removed version cannot be accessed by any user. In LR, some or all of the users who had read access to the version at Remove time may retain access. (Of course, users joining later are not allowed to read the removed object—this respects the Authorization Provenance core property.) In figure 2, if ($o1$,$v1$) is removed with SR in state 5, all users lose access. In case of LR, $u1$ who had read access to ($o1$,$v1$) between states 2 to 5 may retain read access even after the version is removed. A type of Remove, remove$_i$, would be admitted as SR only if it satisfies α_3 stated in Table II. In a g-SIS specification that only supports removes of type LR, there exists at least one well-formed trace that does not satisfy α_3. Traditionally, the semantics of remove have been that of SR. We augment the remove semantics with LR. In a subscription scenario for instance, LR allows a leaving user to retain access to objects that she has paid money for.

Table III. Summary of Group Membership Renewal Semantics

Operation	Explanation	Property
Lossless join	Authorizations prior to join time not lost	$\beta_0 = \forall u : \mathrm{U}.\forall o : \mathrm{O}.\forall v : \mathrm{V}.\forall g : \mathrm{G}.$ $\Box((\mathrm{Join}(u,g) \wedge \neg \mathrm{Remove}(o,v,g)) \wedge$ $\ominus \mathrm{Authz}(u,o,v,g,\mathbf{r})) \rightarrow \mathrm{Authz}(u,o,v,g,\mathbf{r}))$
Nonrestorative join	Authorizations from past membership not explicitly restored	$\beta_1 = \forall u_1, u_2 : \mathrm{U}.\forall o : \mathrm{O}.\forall v : \mathrm{V}.\forall g : \mathrm{G}.$ $\bigcirc\Box(\mathrm{join}_1(u_1,g) \wedge \mathrm{join}_1(u_2,g) \wedge (\exists v : \mathrm{V}.\mathrm{Authz}(u_1,o,v,g,\mathbf{r}) \wedge$ $\neg\exists v : \mathrm{V}.\mathrm{Authz}(u_2,o,v,g,\mathbf{r})) \rightarrow$ $\ominus(\exists v : \mathrm{V}.\mathrm{Authz}(u_1,o,v,g,\mathbf{r}) \wedge \neg\exists v : \mathrm{V}.\mathrm{Authz}(u_2,o,v,g,\mathbf{r})))$
Gainless leave	Authorizations that never held before leave not gained	$\beta_2 = \forall u : \mathrm{U}.\forall o : \mathrm{O}.\forall v : \mathrm{V}.\forall g : \mathrm{G}.$ $\Box(\mathrm{Leave}(u,g) \wedge \mathrm{Authz}(u,o,v,g,\mathbf{r}) \rightarrow \ominus(\neg \mathrm{Leave}(u,g)\, \mathcal{S}$ $(\mathrm{Authz}(u,o,v,g,\mathbf{r}) \wedge (\neg \mathrm{Leave}(u,g)\, \mathcal{S}\, \mathrm{Join}(u,g)))))$

3.6. Membership Renewal Properties

Membership Renewal Properties characterize what, if any, authorizations from previous membership period(s) are enabled or disabled when users rejoin and subsequently leave the group. In the meeting room metaphor, Alice may leave the room and rejoin later. These properties are concerned with her authorizations from her previous sessions in the room and its continuity when she leaves the room again. As the name implies, these properties are applicable only to returning users. We discuss a few renewal properties for users to illustrate that a number of choices exist and to build towards a complete specification of a g-SIS model in the next section. Similar properties for objects could be contemplated.

Lossless vs. Lossy Join. In lossless join, a rejoining user does not lose authorizations held immediately prior to rejoin time. A join operation that causes a user to lose some or all prior authorizations is called lossy. Suppose in Figure 2, $u1$ leaves with a liberal leave in state 7 and rejoins with a join of type lossless in state 9. By rejoining the group, $u1$ does not lose read access to versions that were authorized at the time of liberal leave in state 7. Note that as per Version Authorization Uniformity property, since $u1$ had access to ($o3$,$v0$) prior to rejoin time, she would be authorized to read ($o3$,$v1$) even if she rejoins in state 9 with strict join. In a join of type lossy, $u1$ may lose access to ($o3$,$v0$) and/or ($o1$,$v2$) at rejoin time. A Join operation is of type lossless if it satisfies formula β_0 in Table III. A specification that only supports lossy join does not satisfy this property.

A lossy join is useful in scenarios when authorizations from past membership and those from the new membership are in conflict of interest. For example, if a student registers for a course, drops after the midterm and reregisters the following semester, he/she may be required to relinquish access to exercise solutions and other materials from past enrollment. The student may be readmitted with a lossy join in this scenario.

Nonrestorative vs. Restorative Join. In a nonrestorative join, authorizations from past membership periods may not be explicitly restored at the time of rejoin. On the other hand, a Restorative join explicitly restores authorizations from past membership period. Suppose in Figure 2 $u1$ leaves with a SL in state 7 and rejoins with a restorative join in state 9. In this scenario, $u1$'s access to ($o2$,$v0$), ($o1$,$v1$), ($o2$,$v1$), ($o1$,$v2$) and ($o3$,$v0$) would be explicitly restored in state 9 even if the join type is strict. In nonrestorative join, $u1$ may not be guaranteed access to those versions after rejoin time.

Formalizing nonrestorative join is not straightforward because we want our characterization to be independent of the exact semantics of the join operation in question. Intuitively, we want to require that the nonrestorative join does not add any authorizations that it would not have added if the user had a different history. However, FOTL does not enable one to compare different traces. The solution we take is to consider two different users within a single trace. Because the two users can have different histories with the same trace, this strategy enables us to formalize the property.

Formula β_1 in Table III formalizes nonrestorative join. It states that if two users $u1$ and $u2$ join in the same state with the same join type and $u1$ is authorized to read some version of an object and $u2$ is not authorized to read any version of that object, then it should also be the case in the state prior to join. Intuitively, since the join is of same type, any difference in authorization between the two users may arise only because they had different history of joins and leaves. Restorative join operations does not satisfy β_1.

A restorative join is applicable in scenarios where an incentive is provided for a user to rejoin the group. On the other hand, in subscription service scenarios, a restorative and nonrestorative join may be priced differently, which may decide what prior authorizations to their past subscription materials will be restored.

Gainless vs. Gainful Leave. After rejoining the group, a subsequent leave could either be gainless or gainful. In gainless leave, authorizations that never held during current membership period cannot be obtained by leaving the group. A gainful leave allows new authorizations to be granted at the time of leave. Suppose that in state 9 in Figure 2, u rejoins with a lossless SJ after leaving with SL in state 7. Because of SJ, only $(o1,v1)$ and $(o6,v6)$ can be read in state 10. If $u1$ leaves the group with LL, say in state 11, a gainless LL will not grant any new authorizations other than that to $(o1,v1)$ and $(o6,v6)$. A gainful LL, may additionally grant access to, for example, $(o5,v5)$. A Gainful leave is useful in scenarios where an incentive is provided for a user to leave the group.

In restorative leave, a type of gainful leave, authorizations that the user had prior to joining the group are explicitly restored at leave time. Suppose in Figure 2 $u1$ left the group with LL in state 7 and rejoins with lossy SJ in state 9. In this case, $u1$ possibly loses access to $(o2,v0)$, $(o1,v1)$, $(o2,v1)$, $(o1,v2)$ and $(o3,v0)$ at rejoin time. Later on, if $u1$ leaves with gainful LL, a Restorative leave will allow $u1$ to regain access to those versions at the time of leave. In the meeting room metaphor, suppose Alice is reinvited as a consultant on demand and is required to relinquish her past authorizations due to a conflict of interest with new authorizations that will be enabled by current membership. After Alice performs her duties and leaves the group, it is natural to expect that she will regain access to objects for which she lost authorization when joining the group.

Formula β_2 in Table III characterizes gainless leave. It states that if the user is authorized to read an object after leave, it should have been authorized during the immediate membership period. A specification that only supports gainful leave does not satisfy β_2.

4. THE π-SYSTEM SPECIFICATION

We have specified core properties that are required of any g-SIS policy and then discussed example authorization semantics of several group-operation variants. Methodologically, we want to show that it is feasible to develop a g-SIS specification satisfying all of the core properties. In this section, we develop a stateless g-SIS specification called the π-system. The π-system g-SIS specification has real practical significance

in group-centric application scenarios and also serves as a proof-of-concept. In the next section we show that the π-system satisfies all of the core properties.

The π-system supports both strict and liberal variants of the group operations discussed earlier. Thus, different users and objects may be admitted with different types of respective operations. For example, SJ for $u1$, LJ for $u2$, SL for $u1$ and possibly LL for $u2$, and similarly for objects. For Membership Renewal operations, we confine our scope to join operation variants that are lossless and nonrestorative, and leave operations that are Gainless. These specific types of renewal operations are the most basic since they do not require us to treat past membership authorizations explicitly. Further, the semantics of other renewal operations are likely application dependant. For example, what exact authorizations will be disabled at join time in lossy join depends on the application in question (similarly for restorative join, and gainful leave). In summary, the π-system supports strict and liberal lossless, non-restorative join operations, strict and liberal gainless leave operations, and strict and liberal add, create and remove operations. For notational convenience, we assume that SJ and LJ refer to operations of type lossless and nonrestorative. Similarly, SL and LL refer to gainless leave operations.

Finally, we will follow a specific interpretation of strict and liberal operations which effectively replaces "some or all" in section 3.5 with "all" for authorizations. For instance, in π-system, an LJ allows the user to access *all* versions admitted before and after join time, while SJ allows the user to access *all* versions admitted after Join time.

4.1. Motivation for the π-system

The π-system serves two important purposes. First, the core properties by themselves constrain, but do not fully specify the authorization semantics of g-SIS systems. In contrast, the π-system decides authorization of any user action in the context of any group-operation history. Second, the π-system has real practical significance in group-centric application sharing scenarios. As mentioned earlier, the policy requirements of information sharing is more varied than the specific rigid semantics supported in traditional group policies. For instance, the join and leave semantics of secure multicast are SJ and LL respectively, while that of groups in traditional operating systems are LJ and SL respectively. The add and remove semantics of secure multicast and groups in traditional operating systems is not clear in the literature largely because the focus has been on the user and not on the information being shared. We believe that the intended add and remove semantics of secure multicast are LA and LR respectively while that of groups in operating systems are LA and SR respectively.

By contrast, we believe that information sharing scenarios such as the community cyber incident management require much richer semantics. For instance, in Figure 1(a), domain experts are brought into the incident group with a SJ. That is, they are not authorized to access objects added prior to their join time because, for example, those objects contain sensitive information on how the decision to admit them was arrived at. However, users in the core group are brought into the incident group with LJ. Similarly, when domain experts leave the group, they may be issued a SL so they cannot access incident group objects once they leave. However, core group users may be issued a LL when they leave. Certain objects from the core group may be added to the incident group with SA so as to ensure that only existing incident group users would be authorized to access them. Other non-sensitive objects may be added with LA making them accessible to future users joining the incident group. Similarly, objects may be removed with SR or LR as applicable.

Membership renewal semantics applies in this context as well. For example, past authorizations may be restored to a core group user rejoining the incident group.

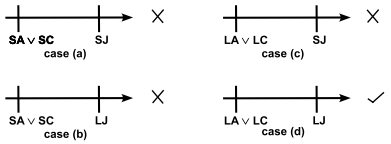
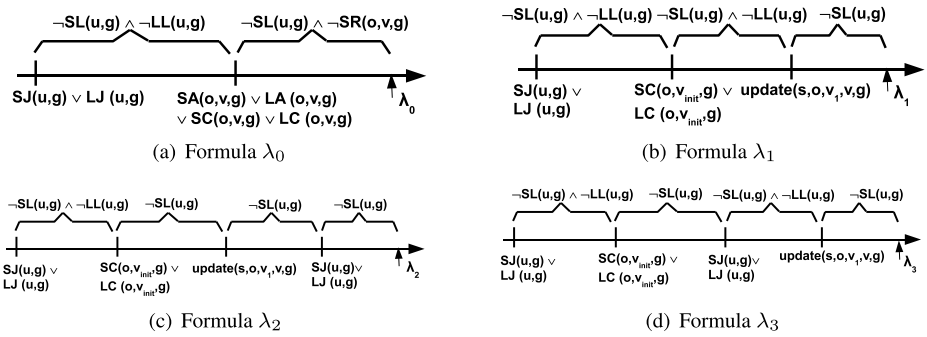


Fig. 3. Cases when Add or CreateO occur prior to join.



(a) Formula λ_0

(b) Formula λ_1

(c) Formula λ_2

(d) Formula λ_3

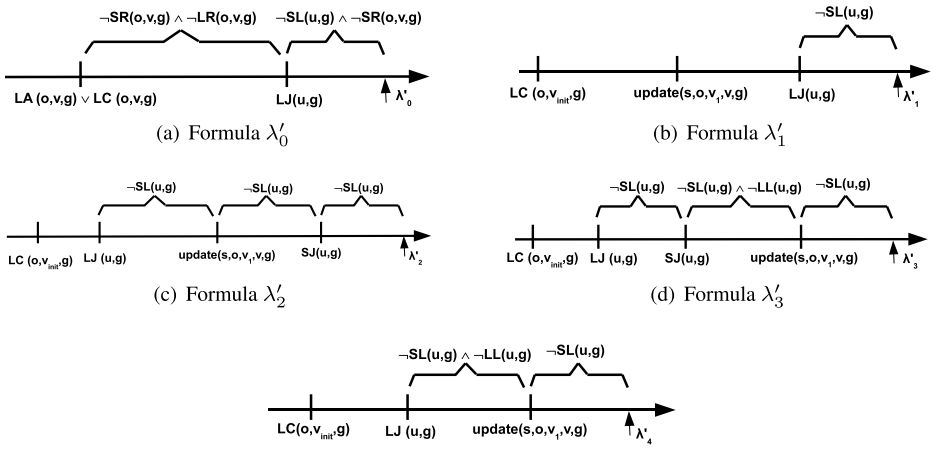
Fig. 4. Add or CreateO after join.

This is a restorative join. In contrast, a rejoining domain expert may be given non-restorative join. Recognizing the simultaneous need for varied operation semantics in different application context, the π-system augments and generalizes authorization policies typically supported in secure multicast and traditional operating system groups. This is achieved by simultaneously supporting (the authorization semantics of) multiple group-operation variants. (The π-system supports a subset of membership renewal variants as mentioned earlier.)

4.2. Construction

There are two scenarios to consider when a user requests access to an object version: (i) the Join event occurred prior to Add or CreateO event or (ii) the Add or CreateO event occurred prior to Join event. In scenario (i), authorization decision is straightforward. As per the semantics of SJ and LJ, the user is authorized to read object versions admitted after join time. Scenario (ii) where an Add or CreateO occurs prior to Join is more interesting. As shown in Figure 3, there are four possible cases. Authz holds only in case (d) where both the join and add/create are of type liberal. Authz does not hold in cases (a) and (b) since the version is introduced with a strict operation prior to join time of the user (see section 3.5). Further, Authz does not hold in case (c) since the join type is strict. (In our design, a strict operation dominates liberal).

In Figures 4 and 5, we consider these two scenarios in further detail. Locally created objects may be updated and the user may have access to some or all of the updated versions of the created object depending on her membership state. For instance, if she is a past member, she may not read versions introduced by update operation after her leave time (this respects Authorization Provenance). On the other hand, if she rejoins, she must be granted read and write access to all or none of those versions as per Version Authorization Uniformity property. Since π-system supports Lossless Join, if the user rejoins with some prior read access to one of the created object versions (for instance due to a liberal leave in the past), she must be authorized to read all versions at join time.



(a) Formula λ'_0 (b) Formula λ'_1

(c) Formula λ'_2 (d) Formula λ'_3

(e) Formula λ'_4

Fig. 5. Add or CreateO before join.

Table IV. FOTL Formulas for Figure 4

$$\lambda_0(u,o,v,g) = \{\neg\mathrm{SL}(u,g) \wedge \neg\mathrm{SR}(o,v,g)\}\,\mathcal{S}\,\{[\mathrm{SA}(o,v,g) \vee \mathrm{LA}(o,v,g) \vee \mathrm{SC}(o,v,g) \vee \mathrm{LC}(o,v,g)] \wedge [(\neg\mathrm{SL}(u,g) \wedge \neg\mathrm{LL}(u,g))\,\mathcal{S}\,(\mathrm{SJ}(u,g) \vee \mathrm{LJ}(u,g))]\}$$

$$\lambda_1(u,s,o,v_1,v,g) = \neg\mathrm{SL}(u,g)\,\mathcal{S}\,\{\mathrm{update}(s,o,v_1,v,g) \wedge ([\neg\mathrm{SL}(u,g) \wedge \neg\mathrm{LL}(u,g)]\,\mathcal{S}\,([\mathrm{SC}(o,\mathbf{v}_{\mathrm{init}},g) \vee \mathrm{LC}(o,\mathbf{v}_{\mathrm{init}},g)] \wedge ([\neg\mathrm{SL}(u,g) \wedge \neg\mathrm{LL}(u,g)]\,\mathcal{S}\,[\mathrm{SJ}(u,g) \vee \mathrm{LJ}(u,g)]))))\}$$

$$\lambda_2(u,s,o,v_1,v,g) = \neg\mathrm{SL}(u,g)\,\mathcal{S}\,\{[\mathrm{SJ}(u,g) \vee \mathrm{LJ}(u,g)] \wedge [\neg\mathrm{SL}(u,g)\,\mathcal{S}\,(\mathrm{update}(s,o,v_1,v,g) \wedge [\neg\mathrm{SL}(u,g)\,\mathcal{S}\,([\mathrm{SC}(o,\mathbf{v}_{\mathrm{init}},g) \vee \mathrm{LC}(o,\mathbf{v}_{\mathrm{init}},g)] \wedge ([\neg\mathrm{SL}(u,g) \wedge \neg\mathrm{LL}(u,g)]\,\mathcal{S}\,[\mathrm{SJ}(u,g) \vee \mathrm{LJ}(u,g)]))])]\}$$

$$\lambda_3(u,s,o,v_1,v,g) = \neg\mathrm{SL}(u,g)\,\mathcal{S}\,\{\mathrm{update}(s,o,v_1,v,g) \wedge [(\neg\mathrm{SL}(u,g) \wedge \neg\mathrm{LL}(u,g))\,\mathcal{S}\,([\mathrm{SJ}(u,g) \vee \mathrm{LJ}(u,g)] \wedge (\neg\mathrm{SL}(u,g)\,\mathcal{S}\,\{(\mathrm{SC}(o,\mathbf{v}_{\mathrm{init}},g) \vee \mathrm{LC}(o,\mathbf{v}_{\mathrm{init}},g)) \wedge ([\neg\mathrm{SL}(u,g) \wedge \neg\mathrm{LL}(u,g)]\,\mathcal{S}\,[\mathrm{SJ}(u,g) \vee \mathrm{LJ}(u,g)])\}))]\}$$

Observe that Figures 4(a) to 4(d) for the Add/CreateO after Join scenario is symmetric to the respective Figures 5(a) to 5(d) in the Add/CreateO before Join scenario. Furthermore, formal specification of these figures using FOTL is also similar as can be observed in Tables IV and V respectively. Since the Add or CreateO before Join scenarios in Figure 5 include one additional case, we discuss this scenario in detail and omit discussion of Figure 4. Figure 5(a) considers the case when read authorization needs to be ascertained for a user u for version v of an object o introduced either through Add or CreateO. Formula λ'_0 in Table V holds in the indicated state in Figure 5(a) if there has not been a strict leave or strict remove since the user liberally joined. In the case v is a created version, $\neg\mathrm{SR}(o,v,g)$ would be true since locally created object versions cannot be removed. At the time user joined, v should exist in the group. Moreover, if it is an added version, it should not have been removed since it was added. Note that where λ'_0 holds, the user would be authorized to read v if she had left the group with a liberal leave.

The remaining formulas consider the cases when an authorization decision needs to be made on updated versions of a locally created object as opposed to its root version or some added version. For instance, formula λ'_1 is similar to λ'_0 except v is an updated version of a locally created object. Formulas λ'_2 and λ'_3 consider the cases when the

Table V. FOTL Formulas for Figure 5

$$\lambda'_0(u,o,v,g) = \{\neg \mathrm{SL}(u,g) \wedge \neg \mathrm{SR}(o,v,g)\}\, \mathcal{S}\, \{\mathrm{LJ}(u,g) \wedge [(\neg \mathrm{SR}(o,v,g) \wedge \neg \mathrm{LR}(o,v,g))\, \mathcal{S}\, (\mathrm{LA}(o,v,g) \vee \mathrm{LC}(o,v,g))]\}$$

$$\lambda'_1(u,s,o,v_1,v,g) = \neg \mathrm{SL}(u,g)\, \mathcal{S}\, \{\mathrm{LJ}(u,g) \wedge \diamond[\mathrm{update}(s,o,v_1,v,g) \wedge \diamond \mathrm{LC}(o,\mathbf{v}_{\mathrm{init}},g)]\}$$

$$\lambda'_2(u,s,o,v_1,v,g) = \neg \mathrm{SL}(u,g)\, \mathcal{S}\, \{\mathrm{SJ}(u,g) \wedge [\neg \mathrm{SL}(u,g)\, \mathcal{S}\, [\mathrm{update}(s,o,v_1,v,g) \wedge (\neg \mathrm{SL}(u,g)\, \mathcal{S}\, (\mathrm{LJ}(u,g) \wedge \diamond \mathrm{LC}(o,\mathbf{v}_{\mathrm{init}},g)))]]\}$$

$$\lambda'_3(u,s,o,v_1,v,g) = \neg \mathrm{SL}(u,g)\, \mathcal{S}\, \{\mathrm{update}(s,o,v_1,v,g) \wedge [(\neg \mathrm{SL}(u,g) \wedge \neg \mathrm{LL}(u,g))\, \mathcal{S}\, (\mathrm{SJ}(u,g) \wedge [\neg \mathrm{SL}(u,g)\, \mathcal{S}\, (\mathrm{LJ}(u,g) \wedge \diamond \mathrm{LC}(o,\mathbf{v}_{\mathrm{init}},g))]]\}$$

$$\lambda'_4(u,s,o,v_1,v,g) = \neg \mathrm{SL}(u,g)\, \mathcal{S}\, \{\mathrm{update}(s,o,v_1,v,g) \wedge [(\neg \mathrm{SL}(u,g) \wedge \neg \mathrm{LL}(u,g))\, \mathcal{S}\, (\mathrm{LJ}(u,g) \wedge \diamond \mathrm{LC}(o,\mathbf{v}_{\mathrm{init}},g))]\}$$

user rejoins with SJ after multiple LJ's in the past. As shown in the respective figures, the two formulas differ with respect to the time at which the update occurred. For example, in Figure 5(c), u had access to the root version of the object in question at the time of liberal join. User u then left the group with a liberal leave and rejoins with SJ. In this scenario, since joins in π-system are Lossless, u would be authorized to read v (introduced by update) at join time since she had access to $\mathbf{v}_{\mathrm{init}}$ prior to join time (uniformity property). In contrast, in Figure 5(d), the update operation occurs after SJ. Observe that the four cases discussed so far are symmetric to those in Figure 4. Formula λ'_4 is unique to this scenario where the user should be authorized in case the object was created prior to join time but updated after. Note that u would not be authorized if this join type is strict.

We now specify the precise conditions under which authorization can hold in the π-system (Table VI) as per our design considerations discussed in Section 3.1.

Definition 4.1 π-system. The π-system is given by:

$$\pi = \chi_0 \wedge \chi_1 \wedge \chi_2 \wedge \chi_3 \wedge \chi_4 \wedge \chi_5 \wedge \chi_6$$

That is, a user is authorized to read a version if and only if one of the scenarios in Figure 4 or 5 is satisfied (χ_0). A user is authorized to write to a version if and only if she is authorized to read the version (χ_1). Note that the version was introduced by means of create or a subsequent update in the past. A user is allowed to create a subject in a group if the user had joined the group at least once in the past (χ_2). This allows the users who liberally left the group to read authorized objects. Existing subjects may read and write if and only if the user who created the subject is authorized to read and write respectively (χ_3 and χ_4). The read and update operations may occur only if the subject is authorized to read and write respectively (χ_5). Thus, the π system does not address all aspects of subject management. For instance, it would be entirely consistent with π to prevent users from having subjects in groups in which the user has no access rights. This would amount to permitting only a subset of the traces permitted by the π-system. As such, all the results in the next section would hold in the more restricted semantics just as they are shown to hold for all traces that satisfy π. Finally, the π-system should be well-formed (χ_6).

5. FORMAL ANALYSIS

In this section, we show that the π-system is a semantically correct g-SIS specification. We also show that the core properties are both independent and consistent. Both of these results are based on proofs that are mechanically generated by using model checking. Model checking is an automated approach for verifying finite systems,

Table VI. The π-system

$$\chi_0 = \forall u : \mathrm{U}.\forall o : \mathrm{O}.\forall v : \mathrm{V}.\forall g : \mathrm{G}.$$
$$\Box(\mathrm{Authz}(u, o, v, g, \mathbf{r}) \leftrightarrow \exists v_1 : \mathrm{V}.\exists s : \mathrm{S}.(\lambda_0(u, o, v, g) \vee \ldots \vee \lambda_3(u, s, o, v_1, v, g) \vee$$
$$\lambda'_0(u, o, v, g) \vee \ldots \vee \lambda'_4(u, s, o, v_1, v, g)))$$

$$\chi_1 = \forall u : \mathrm{U}.\forall o : \mathrm{O}.\forall v : \mathrm{V}.\forall g : \mathrm{G}.$$
$$\Box(\mathrm{Authz}(u, o, v, g, \mathbf{w}) \leftrightarrow \mathrm{Authz}(u, o, v, g, \mathbf{r}) \wedge (\neg \mathrm{Leave}(u, g)\, \mathcal{S}\, \mathrm{Join}(u, g)) \wedge$$
$$(\exists v_1 : \mathrm{V}.\exists s : \mathrm{S}.\diamondminus \mathrm{update}(s, o, v_1, v, g) \vee \diamondminus(\mathrm{LC}(o, v, g) \vee \mathrm{SC}(o, v, g))))$$

$$\chi_2 = \forall u : \mathrm{U}.\forall s : \mathrm{S}.\forall g : \mathrm{G}.$$
$$\Box(\mathrm{createS}(u, s, g) \rightarrow \diamondminus \mathrm{Join}(u, g))$$

$$\chi_3 = \forall s : \mathrm{S}.\forall o : \mathrm{O}.\forall v : \mathrm{V}.\forall g : \mathrm{G}.$$
$$\Box(\mathrm{AuthzS}(s, o, v, g, \mathbf{r}) \leftrightarrow \exists u : \mathrm{U}.(\mathrm{Authz}(u, o, v, g, \mathbf{r}) \wedge$$
$$(\neg \mathrm{kill}(u, s, g)\, \mathcal{S}\, \mathrm{createS}(u, s, g))))$$

$$\chi_4 = \forall s : \mathrm{S}.\forall o : \mathrm{O}.\forall v : \mathrm{V}.\forall g : \mathrm{G}.$$
$$\Box(\mathrm{AuthzS}(s, o, v, g, \mathbf{w}) \leftrightarrow \exists u : \mathrm{U}.(\mathrm{Authz}(u, o, v, g, \mathbf{w}) \wedge$$
$$(\neg \mathrm{kill}(u, s, g)\, \mathcal{S}\, \mathrm{createS}(u, s, g))))$$

$$\chi_5 = \forall s : \mathrm{S}.\forall o : \mathrm{O}.\forall v_1, v_2 : \mathrm{V}.\forall g : \mathrm{G}.$$
$$\Box(\mathrm{read}(s, o, v_1, g) \rightarrow \mathrm{AuthzS}(s, o, v_1, g, \mathbf{r})) \wedge$$
$$\Box(\mathrm{update}(s, o, v_1, v_2, g) \rightarrow \mathrm{AuthzS}(s, o, v_1, g, \mathbf{w}))$$

$$\chi_6 = \forall u_1, u_2 : \mathrm{U}.\forall s_1, s_2 : \mathrm{S}.\forall o : \mathrm{O}.\forall v_1, v_2, v_3 : \mathrm{V}.\forall g_1, g_2 : \mathrm{G}.$$
$$\tau_0(u_1, s_1, s_2, o, v_1, v_2, v_3, g_1) \wedge \ldots \wedge \tau_3(u_1, s_1, o, v_1, g_1)$$

modeled as finite state machines. Model checking exhaustively explores the reachable state space based on the transition relation to determine if a given property holds. In the case that a property fails to hold, a model checker produces a counterexample consisting of a trace that shows how the failure can arise, which can be used to correct the model or the property specification. Our use of the model checker is nonstandard: the model we use nondeterministically specifies all possible traces involving g-SIS events and authorizations. We call this the *most general model*. The (LTL) formulas that are shown to hold for all these traces express the required relationships between core properties and the system specification. We choose to use model checker NuMSV [Cimatti et al. 2000] because of the size of the specification formula and because of the large number of cases that have to be considered. NuSMV is a BDD-based (symbolic), highly optimized tool that can handle a relatively larger state space than an explicit model checker does. In addition, it supports propositional temporal logic formulas with both past and future temporal operators, which are used in g-SIS.

Automation of the proof process provides a level of assurance of validity that is essentially impossible to achieve by manual means. By using this approach, we discovered several subtle errors in our own preliminary designs. It also enabled efficient verification following design revisions that were made for reasons other than correctness. Unfortunately, the theorems cannot be proven entirely by using existing model-checker methods. In practice the NuSMV model checker we use can handle only finite g-SIS systems. (In our case, the carriers we used each contains just one to three elements. Based on the values in these small carriers, we convert FOTL formulas into propositional LTL formulas that NuSMV can handle.) So proving the theorems given in this section requires showing that the results obtained by using the model checker can be generalized to countably infinite carriers, which are needed to support g-SIS systems of unbounded size.

5.1. Semantic Correctness of π-system

We wish to show that the π-system satisfies the core properties, $\kappa_0, \ldots, \kappa_8$. We have verified [Krishnan et al. 2010] the following result by using the NuSMV model checker.

LEMMA 5.1 SMALL-CARRIER SEMANTIC CORRECTNESS OF π. *Let* $\mathcal{U}_{\text{sm}} = \{\mathbf{u}_1, \mathbf{u}_2\}$, $\mathcal{S}_{\text{sm}} = \{\mathbf{s}_1, \mathbf{s}_2\}$, $\mathcal{O}_{\text{sm}} = \{\mathbf{o}\}$, $\mathcal{V}_{\text{sm}} = \{\mathbf{v}_{\text{init}}, \mathbf{v}_1, \mathbf{v}_2\}$, $\mathcal{G}_{\text{sm}} = \{\mathbf{g}\}$ *and* $\mathcal{C}_{\text{sm}} = \langle \mathcal{U}_{\text{sm}}, \mathcal{S}_{\text{sm}}, \mathcal{O}_{\text{sm}}, \mathcal{V}_{\text{sm}}, \mathcal{G}_{\text{sm}}, \mathcal{P} \rangle$. *Then for all traces over these carriers,* $\sigma \in \Sigma^{\omega}_{\mathcal{C}_{\text{sm}}}$, *if* $\sigma \models \pi$, *then* $\sigma \models \bigwedge_{0 \le q \le 8} \kappa_q$.

Intuitively, the generalization of the model checker result makes use of two characteristics of the π-system. First, authorization of one user to access versions of one object is independent of authorizations of other users and on other objects. Second, authorization to access an object version that is introduced by an update operation is independent of the object version upon which that update was applied. So, for instance, if the operation update($\mathbf{s}, \mathbf{o}, \mathbf{v}_1, \mathbf{v}_2, \mathbf{g}$) is performed, a subject's authorization to access $\mathbf{v}_2$ does not depend on which version $\mathbf{v}_1$ happens to be. This latter point is helpful in limiting the number of object versions that must be considered by the model checker.

We deal with the second of these issues in the next lemma, which shows that, given any trace that satisfies π, we obtain another trace that satisfies π if we change each update operation so that it is applied to the initial version of the object in question. Recall that a trace is a sequence of states. Each state is a mapping that provides for each predicate in the language the relation over the appropriate carriers that is used to interpret the predicate in that state. These relations model the actions that take place in the transition to the current state, as well as the actions each user or subject is authorized to perform in that state. Given a trace σ, let $\widehat{\sigma}$ be obtained from σ as follows. For each $i \in \mathbb{N}$, if $\langle \mathbf{s}, \mathbf{o}, \mathbf{v}_1, \mathbf{v}_2, \mathbf{g} \rangle \in [\![\text{update}]\!]\sigma_i$, then this tuple is replaced by $\langle \mathbf{s}, \mathbf{o}, \mathbf{v}_{\text{init}}, \mathbf{v}_2, \mathbf{g} \rangle \in [\![\text{update}]\!]\widehat{\sigma_i}$; all other relations in $\widehat{\sigma_i}$ are identical to those in σ_i. In particular, for all $i \in \mathbb{N}$, $[\![\text{Authz}]\!]\sigma_i = [\![\text{Authz}]\!]\widehat{\sigma_i}$ and $[\![\text{AuthzS}]\!]\sigma_i = [\![\text{AuthzS}]\!]\widehat{\sigma_i}$. The next lemma says that changing the update relation does not prevent the resulting trace $\widehat{\sigma}$ from satisfying π. Proof for this lemma is discussed in Appendix B.

LEMMA 5.2. *Let* $\sigma \in \Sigma^{\omega}_{\mathcal{C}}$ *be any trace. Then* $\sigma \models \pi$ *implies* $\widehat{\sigma} \models \pi$.

We denote the projection of a trace σ onto a collection of small carriers, $\mathcal{C}_{\text{sm}}$, by $\sigma{\downarrow_{\mathcal{C}_{\text{sm}}}}$. We obtain $\sigma{\downarrow_{\mathcal{C}_{\text{sm}}}}$ from σ by modifying each state in σ so that the relation given as the interpretation of each predicate becomes a relation over carrier elements in $\mathcal{C}_{\text{sm}}$. (Done by removing from each relation in each state of σ all tuples that contain elements not in $\mathcal{C}_{\text{sm}}$).

We denote the projection of a trace σ onto a collection of small carriers, $\mathcal{C}_{\text{sm}}$, by $\sigma{\downarrow_{\mathcal{C}_{\text{sm}}}}$. We obtain $\sigma{\downarrow_{\mathcal{C}_{\text{sm}}}}$ from σ by modifying each state in σ so that the relation given as the interpretation of each predicate becomes a relation over carrier elements in $\mathcal{C}_{\text{sm}}$. (Done by removing from each relation in each state of σ all tuples that contain elements not in $\mathcal{C}_{\text{sm}}$).

THEOREM 5.3 SEMANTIC CORRECTNESS OF π. *The π-system entails the g-SIS core.*

$$\pi \models \bigwedge_{0 \le q \le 8} \kappa_q$$

The proof for this theorem in given in Appendix B.

5.2. Entailment Theorems for Secondary Properties

In this section, we outline the proofs for entailment of secondary properties discussed in Sections 3.5 and 3.6. We follow the approach that we have taken in proving Theorem 5.3.

THEOREM 5.4 MEMBERSHIP RENEWAL ENTAILMENT OF π. *The* π-system *entails the Membership Renewal Properties (β_0 to β_2):*

$$\pi \vDash \bigwedge_{0 \leq r \leq 2} \beta_r$$

We have proved this theorem for the small carrier case using model checking [Krishnan et al. 2010] as in Lemma 5.1. The generalization to large carrier case follows.

Since the π-system supports membership operations of type both strict and liberal, it is not feasible to verify the membership properties. We fix the operation types in the π-system to be strict to obtain a μ-system. In the μ-system, the only operations allowed are strict versions of join, leave, add, create and remove operations. This corresponds to the typical secure multicast policies of forward and backward secrecy [Rafaeli and Hutchison 2003]. The μ-system can be easily derived from π-system by substituting the liberal operations in λ_0 to λ_3 and λ'_0 to λ'_4 with false. We denote each χ that we obtain by means of substitution as χ'.

Definition 5.5. The *μ-system* is given by:

$$\mu = \chi'_0 \wedge \chi'_1 \wedge \chi'_2 \wedge \chi'_3 \wedge \chi'_4 \wedge \chi'_5 \wedge \chi'_6$$

We can now verify the membership properties against the μ-system.

THEOREM 5.6 MEMBERSHIP ENTAILMENT OF μ. *The* μ-system *entails the Membership Properties (α_0 to α_3):*

$$\mu \vDash \bigwedge_{0 \leq r \leq 3} \alpha_r$$

Again, we have proved this theorem for the small carrier case [Krishnan et al. 2010]. The large carrier case follows as before. Note that if the μ-system admitted any liberal operations, the above entailment would fail.

5.3. Independence and Consistency

We prove that the g-SIS core properties are both mutually independent and consistent.

5.3.1. Independence Theorem. The independence theorem states that no core property, nor its negation, is logically entailed by the conjunction of the other core properties.

THEOREM 5.7 INDEPENDENCE. *The g-SIS core properties are mutually independent. That is the following two assertions hold:*

*For each $i \in [0..8]$, it is **not** the case that for each $\sigma \in \Sigma^{\omega}_{\mathcal{C}}$,*

$$\sigma \vDash ((\theta \wedge \bigwedge_{i \neq j, j \in [0..8]} \kappa_j) \rightarrow \kappa_i) \tag{7}$$

*For each $i \in [0..8]$, it is **not** the case that for each $\sigma \in \Sigma^{\omega}_{\mathcal{C}}$,*

$$\sigma \vDash ((\theta \wedge \bigwedge_{i \neq j, j \in [0..8]} \kappa_j) \rightarrow \neg\kappa_i) \tag{8}$$

PROOF. We present the proof of assertion (7). Assertion (8) follows by a similar argument. Using the most-general model (see Section 5), we used the model checker to prove the above formula for the small carrier ($\mathcal{C}_{\text{sm}}$) case in which

$$\mathcal{U}_{\text{sm}} = \{\mathbf{u}\}, \mathcal{S}_{\text{sm}} = \{\mathbf{s}\}, \mathcal{O}_{\text{sm}} = \{\mathbf{o}\}, \mathcal{V}_{\text{sm}} = \{\mathbf{v}_{\text{init}}, \mathbf{v}_1, \mathbf{v}_2\}, \mathcal{G}_{\text{sm}} = \{\mathbf{g}\}, \mathcal{P}_{\text{sm}} = \{\mathcal{P}\}$$

In the model checker, we assert that $((\theta \wedge \bigwedge_{i \neq j, j \in [0..8]} \kappa_j) \rightarrow \kappa_i)$ holds for the most-general model. The model checker finds a counter example $\sigma \in \Sigma^{\omega}_{\mathcal{C}_{\text{sm}}}$. Because it follows that $\sigma \in \Sigma^{\omega}_{\mathcal{C}}$, this shows independence in the general case ($\mathcal{C}$). □

Note that for this theorem, we just had to show the existence of a single counter example, in contrast with the results in the previous section, where we had to show something holds for all traces in $\Sigma^{\omega}_{\mathcal{C}}$. NuSMV code proving the independence of core properties for the small carrier case can be found at Krishnan et al. [2010].

5.3.2. Consistency Theorem. The consistency theorem states that there exists a well-formed trace that satisfies all the core properties. Note that theorem 5.3 does not prove the consistency of the core properties unless we know that there is a trace σ that satisfies π.

THEOREM 5.8 CONSISTENCY. *The g-SIS core properties are satisfiable. That is:*

$$\textit{There exists a } \sigma \in \Sigma^{\omega}_{\mathcal{C}} \textit{ such that } \sigma \models \theta \wedge \bigwedge_{i \in [0..8]} \kappa_i$$

PROOF. As in the proof of the independence of the core properties, here we need only to show the existence of a single trace σ. We have done this by using the model checker. And since any trace in $\Sigma^{\omega}_{\mathcal{C}_{\text{sm}}}$ is also a trace in $\Sigma^{\omega}_{\mathcal{C}}$, this proves the result in the general case. □

NuSMV code proving the consistency of core properties for small carriers can be found at Krishnan et al. [2010].

6. RELATED WORK

The traditional approach to information sharing, which we characterize as dissemination-centric, focuses on attaching attributes and policies to an object as it is disseminated from producers to consumers in a system. These policies are sometimes described as being "sticky" [Bandhakavi et al. 2006; Chadwick and Lievens 2008; Mont et al. 2003]. As an object is disseminated further down a supply chain the policies may get modified, such modification itself being controlled by existing policies. This mode of information sharing goes back to early discussions on originator-control systems [Abrams et al. 1991; Graubart 1989; McCollum et al. 1990] in the 1980's and Digital Rights Management in the 1990's and 2000's. XrML [XrML 2001] and ODRL [ODRL 2005] are recent examples of policy languages developed for this purpose. Group-centric sharing differs in that it advocates bringing users and objects together to facilitate sharing. We envision that dissemination and group-centric sharing will coexist in a mutually supportive manner. For example, objects could be added with "sticky" policies in a group-centric model. At a pragmatic level, we believe group-centric and dissemination-centric are significantly different approaches to information sharing.

FOTL has been previously used to specify security policies such as in the privacy domain [Barth et al. 2006]. At the policy level, the closest to the candidate g-SIS specification discussed in this article, the π-system, is in the domain of secure multicast. As discussed earlier, the π-system subsumes policies typically considered in secure multicast and traditional groups in operating systems. Enforcement aspects of secure

group communication such as key management and agreement and efficient and scalable re-keying with change in user membership are well studied in the literature [Ballardie 1996; Ballardie and Crowcroft 1995; Berkovits 1991; Chiou and Chen 1989; Fiat and Naor 1994; Gong 1996; Harney et al. 1997; Kim et al. 2000; Mittra 1997; Stinson 1997; Wong et al. 1998]. Also, Rafaeli and Hutchison [2003] provide a survey of key management for secure group communication.

The work presented in this article builds upon our preliminary research on read only models for g-SIS discussed in Krishnan et al. [2009a, 2009c]. (Although not explicitly identified, the groups considered in our prior work were by definition isolated). The use of groups in access control goes back to earliest OS's [Saltzer 1974] and is now commonplace in modern OS's and directory services such as LDAP [Zeilenga et al. 2006]. Administration of user membership in groups has been well-studied. Sandhu and Share [1986] discuss user authorization schemes for dynamically defining membership in groups. They demonstrate a variety of administrative policy choices for defining group membership. For instance, a user may create a group whose members are derived from groups that she has access to. Sandhu [1988b] discusses group membership policies using special kind of group hierarchies based on a subgroup partial order. In hierarchical groups, membership in a group is automatic if the user is a member of a dominant group. The modern concept of Role-Based Access Control [Sandhu et al. 1996] simplifies administration in organizations marrying concepts such as hierarchies and constraints. (See Sandhu [1996] for a discussion on roles versus groups). Temporal aspects of access control have been previously studied in different contexts. Bertino et al. [1994] provide a temporal authorization model for discretionary access control. Specifically, the model extends traditional authorization with the notion of temporal interval of its validity (such as expiry of a right to access an object) and temporal dependency (such as contingency of a user's authorization on that of another). Further, role-based access control models have been extended to accommodate dynamic aspects of a role such as restrictions on the time period during which a role may be invoked [Bertino et al. 2001; Joshi et al. 2005]. Ahn et al. [2007] use a role-based delegation framework for specifying policies for resource and information sharing within and across organizations.

Achieving security in dynamic coalitions has been extensively studied [Atluri and Warner 2004; Cohen et al. 2002; Freudenthal et al. 2002; Khurana and Gligor 2004; Khurana et al. 2002; Li et al. 2002; Phillips et al. 2002; Shands et al. 2001; Warner et al. 2007]. The intention of g-SIS is that the security models already in use by participating organizations need not be necessarily integrated with the g-SIS system. Instead, g-SIS should be orthogonal and complementary to those deployed models. This is sharply distinguished from those of prior work, such as in the context of dynamic coalitions, which generally focus on integrating security infrastructure across organizational boundaries. For instance, the work of Shands et al. [2001] on secure virtual enclaves, focusses specifically on building a middleware infrastructure amongst participating organizations while being agnostic to the security policy. Cohen et al. [2002] have proposed a complex role-based model for integrating security infrastructure of multiple organizations in a coalition. Furthermore, prior work on dynamic coalition focuses on sharing static resources such as a service or a computing facility. g-SIS focuses explicitly on information or object sharing, the life-cycle of which is highly dynamic. Information flow is also a major concern in such a scenario. g-SIS models explicitly handle information flow issues by distinguishing users (people) from software subjects to restrict permissions, thereby containing unpredictable information flow. The work of Li et al. (*RT* [Li et al. 2002]) falls in the domain of dynamic coalitions. We believe that, depending on the application, *RT* can be a powerful administrative model for g-SIS.

7. CONCLUSION AND FUTURE WORK

The work presented here on g-SIS serves as a foundation for systematic study and extensions in several directions. FOTL supported our research methodology and allowed us to specify a *stateless* g-SIS model without getting distracted about the structure of states in g-SIS. As a first step towards realizing stateless g-SIS policies specified in this article, we plan to develop a *stateful* model involving data structures that efficiently keep track of various aspects of the model such as the membership states of users and objects in a group. Furthermore, we plan to develop a proof technique to ensure that the stateless and stateful models are authorization equivalent. That is, a stateful model will authorize a user to read or write an object if and only if authorization holds in the stateless model. Due to the expected distributed nature of the stateful specification, we believe that some approximations may be necessary to achieve equivalence.

Our current work focussed on isolated groups. Specifically, export of objects from groups and direct interaction and dependencies between groups were not allowed. We plan to investigate a generalized g-SIS model with different groups with common users and objects interacting with each other. We refer to this as the connected g-SIS model in which groups could engage in relationships with various authorization semantics. We have investigated such semantics in Sandhu et al. [2010]. We expect that the connected g-SIS model would require additional group operations such as suspending and resuming locally created object versions, delegation, user substitution, etc. A major research area here is to study information flow from one group to another through subjects. We plan to investigate and specify static information flow properties that would govern expected flow similar to the core properties we discussed for isolated groups. Finally, our future work also involves investigating synergy between g-SIS and classic models such as DAC, LBAC and RBAC.

REFERENCES

ABRAMS, M., HEANEY, J., KING, O., LAPADULA, L., LAZEAR, M., AND OLSON, I. 1991. Generalized framework for access control: Towards prototyping the ORGCON Policy. In *Proceedings of the National Computer Security Conference*.

AHN, G.-J., MOHAN, B., AND HONG, S.-P. 2007. Towards secure information sharing using role-based delegation. *J. Netw. Comput. Appl. 30*, 1, 42–59.

ATLURI, V. AND WARNER, J. 2004. Automatic enforcement of access control policies among dynamic coalitions. In *Proceedings of the International Conference on Distributed Computing and Internet Technology*.

BADGER, L., STERNE, D. F., SHERMAN, D. L., WALKER, K. M., AND HAGHIGHAT, S. A. 1995. Practical domain and type enforcement for UNIX. In *Proceedings of the IEEE Symposium on Security and Privacy (SP'95)*. IEEE Computer Society, Los Alamitos, CA, 66.

BALLARDIE, A. 1996. Scalable multicast key distribution. http://rsync.tools.ietf.org/html.rfc1949.

BALLARDIE, T. AND CROWCROFT, J. 1995. Multicast-specific security threats and counter-measures. In *Proceedings of the Symposium on Network and Distributed System Security*.

BANDHAKAVI, S., ZHANG, C. C., AND WINSLETT, M. 2006. Super-sticky and declassifiable release policies for flexible information dissemination control. In *Proceedings of the ACM Workshop on Privacy in Electronic Society*. 51–58.

BARTH, A., DATTA, A., MITCHELL, J. C., AND NISSENBAUM, H. 2006. Privacy and contextual integrity: Framework and applications. In *Proceedings of the IEEE Symposium on Security and Privacy*. 184–198.

BELL, D. AND LA PADULA, L. 1975. Secure computer systems: Unified exposition and multics interpretation. Tech. rep. ESD-TR-75-306, MITRE Corp.

BERKOVITS, S. 1991. How to broadcast a secret. In *Proceedings of the EUROCRYPT'91*. 535–541.

BERTINO, E., BETTINI, C., AND SAMARATI, P. 1994. A temporal authorization model. In *Proceedings of the 2nd ACM Conference on Computer and Communications Security*.

BERTINO, E., BONATTI, P., AND FERRARI, E. 2001. TRBAC: A temporal role-based access control model. *ACM Trans. Info. Syst. Sec. 4*, 3, 191–233.

CHADWICK, D. W. AND LIEVENS, S. F. 2008. Enforcing "sticky" security policies throughout a distributed application. In *Proceedings of the Workshop on Middleware Security (MidSec'08)*. 1–6.

CHIOU, G. AND CHEN, W. 1989. Secure broadcasting using the secure lock. *IEEE Trans. Softw. Engin. 15*, 8, 929–934.

CIMATTI, A., CLARKE, E., GIUNCHIGLIA, F., AND ROVERI, M. 2000. NuSMV: A new symbolic model checker. *J. Softw. Tools Tech. Transfer*, 410–425.

COHEN, E., THOMAS, R. K., WINSBOROUGH, W., AND SHANDS, D. 2002. Models for coalition-based access control (cbac). In *Proceedings of the 7th ACM Symposium on Access Control Models and Technologies (SACMAT'02)*. ACM, New York, NY, 97–106.

FIAT, A. AND NAOR, M. 1994. Broadcast Encryption. In *Proceedings of Crypto'93*. 480–491.

FREUDENTHAL, E., PESIN, T., PORT, L., KEENAN, E., AND KARAMCHETI, V. 2002. drbac: Distributed role-based access control for dynamic coalition environments. In *Proceedings of the 22nd International Conference on Distributed Computing Systems (ICDCS'02)*. IEEE Computer Society, Los Alamitos, CA, 411.

GONG, L. 1996. Enclaves: Enabling secure collaboration over the internet. In *Proceedings of the USENIX Security Symposium*.

GRAUBART, R. 1989. On the need for a third form of access control. In *Proceedings of the 12th National Computer Security Conference*. 296–304.

HARNEY, H., MUCKENHIRN, C., AND RIVERS, T. 1997. Group key management protocol (GKMP) architecture. Tech. rep., RFC 2094, SPARTA Inc.

HARRISON, M., RUZZO, W., AND ULLMAN, J. 1976. Protection in operating systems. *Comm. ACM*.

JAEGER, T. AND TIDSWELL, J. E. 2001. Practical safety in flexible access control models. *ACM Trans. Info. Syst. Sec. 4*, 2, 158–190.

JOSHI, J. B. D., BERTINO, E., LATIF, U., AND GHAFOOR, A. 2005. A generalized temporal role-based access control model. *IEEE Trans. Knowl. Data Engin. 17*, 1, 4–23.

KHURANA, H. AND GLIGOR, V. D. 2004. A model for access negotiations in dynamic coalitions. In *Proceedings of the IEEE International Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises*.

KHURANA, H., GLIGOR, V., AND LINN, J. 2002. Reasoning about joint administration of access policies for coalition resources. In *Proceedings of the 22nd International Conference on Distributed Computing Systems*. 429.

KIM, Y., PERRIG, A., AND TSUDIK, G. 2000. Simple and fault-tolerant key agreement for dynamic collaborative groups. In *Proceedings of the 7th ACM Conference on Computer and Communications Security*. 235–244.

KRISHNAN, R., SANDHU, R., AND RANGANATHAN, K. 2007. PEI models towards scalable, usable and high-assurance information sharing. In *Proceedings of the Symposium on Access Control Models and Technologies (SACMAT'07)*. ACM, 145–150.

KRISHNAN, R., SANDHU, R., NIU, J., AND WINSBOROUGH, W. 2009a. A conceptual framework for group-centric secure information sharing. In *Proceedings of the ACM Symposium on Information, Computer and Communications Security*.

KRISHNAN, R., SANDHU, R., NIU, J., AND WINSBOROUGH, W. 2009b. Towards a framework for group-centric secure collaboration. In *Proceedings of IEEE International Conference on Collaborative Computing*.

KRISHNAN, R., SANDHU, R., NIU, J., AND WINSBOROUGH, W. H. 2009c. Foundations for group-centric secure information sharing models. In *Proceedings of the ACM Symposium on Access Control Models and Technologies*.

KRISHNAN, R., NIU, J., SANDHU, R., AND WINSBOROUGH, W. 2010. Model checking code for proofs of small carrier cases. http://profsandhu.com/ram_krishnan/tissec_sacmat/index.html.

LAMPORT, L. 1977. Proving the correctness of multiprocess programs. *IEEE Trans. Softw. Engin. 3*, 2, 125–143.

LAMPSON, B. W. 1973. A note on the confinement problem. *Comm. ACM 16*, 10, 613–615.

LI, N., MITCHELL, J. C., AND WINSBOROUGH, W. H. 2002. Design of a role-based trust-management framework. In *Proceedings of the IEEE Symposium on Security and Privacy*.

LIPTON, R. AND SNYDER, L. 1977. A linear time algorithm for deciding subject security. *J. ACM. 24*, 3.

MCCOLLUM, C., MESSING, J., AND NOTARGIACOMO, L. 1990. Beyond the pale of MAC and DAC: Defining new forms of access control. In *Proceedings of the IEEE Symposium on Security and Privacy*. 190–200.

MITTRA, S. 1997. Iolus: A framework for scalable secure multicasting. *ACM SIGCOMM Comp. Comm. Rev.*

MONT, M. C., PEARSON, S., AND BRAMHALL, P. 2003. Towards accountable management of identity and privacy: Sticky policies and enforceable tracing services. In *Proceedings of the International Workshop on Databases and Expert System Applications*.

ODRL. 2005. The open digital rights language initiative. www.odrl.net.

PHILLIPS JR., C. E., TING, T., AND DEMURJIAN, S. A. 2002. Information sharing and security in dynamic coalitions. In *Proceedings of the ACM Symposium on Access Control Models and Technologies*.

PNUELI, A. 1977. The temporal logic of programs. In *Proceedings of the 18th IEEE Symposium on Foundations of Computer Science*. 46–67.

RAFAELI, S. AND HUTCHISON, D. 2003. A survey of key management for secure group communication. *ACM Comput. Surv.* 309–329.

SALTZER, J. AND SCHROEDER, M. 1975. The protection of information in computer systems. *Proc. IEEE 63*, 9, 1278–1308.

SALTZER, J. H. 1974. Protection and the control of information sharing in multics. *Comm. ACM 17*, 7, 388–402.

SANDHU, R. 1988a. The schematic protection model: Its definition and analysis for acyclic attenuating schemes. *J. ACM 35*, 2, 404–432.

SANDHU, R. 1988b. The ntree: A two dimension partial order for protection groups. *ACM Trans. Comput. Syst. 6*, 2, 197–222.

SANDHU, R. 1992. The typed access matrix model. In *Proceedings of the IEEE Symposium on Security and Privacy*. 122.

SANDHU, R. 1996. Roles versus groups. In *Proceedings of the ACM Workshop on Role Based Access Control*.

SANDHU, R. 2009. The PEI framework for application-centric security. In *Proceedings of 5th International Conference on Collaborative Computing: Networking, Applications and Worksharing*.

SANDHU, R. S. AND SHARE, M. E. 1986. Some owner based schemes with dynamic groups in the schematic protection model. In *Proceedings of the IEEE Symposium on Security and Privacy*.

SANDHU, R., COYNE, E., FEINSTEIN, H., AND YOUMAN, C. 1996. Role-based access control models. *IEEE Comput.* 38–47.

SANDHU, R., BHAMIDIPATI, V., AND MUNAWER, Q. 1999. The ARBAC97 model for role-based administration of roles. *ACM Trans. Info. Syst. Sec. 2*, 1, 105–135.

SANDHU, R., RANGANATHAN, K., AND ZHANG, X. 2006. Secure information sharing enabled by trusted computing and PEI models. In *Proceedings of the ACM Symposium on Information, Computer and Communications Security*. 2–12.

SANDHU, R., KRISHNAN, R., NIU, J., AND WINSBOROUGH, W. 2010. Group-centric models for secure and agile information sharing. In *Proceedings of the 5th International Conference, on Mathematical Methods, Models, and Architectures for Computer Network Security (MMM-ACNS'10)*. Springer.

SHANDS, D., YEE, R., JACOBS, J., AND SEBES, E. 2001. Secure virtual enclaves: Supporting coalition use of distributed application technologies. *ACM Trans. Info. Syst. Sec. 4,* 2, 103–133.

STINSON, D. 1997. On some methods for unconditionally secure key distribution and broadcast encryption. *Des. Codes Cryptog. 12*, 3, 215–243.

TCG. 2007. TCG specification architecture overview. http://www.trustedcomputinggroup.org.

WARNER, J., ATLURI, V., MUKKAMALA, R., AND VAIDYA, J. 2007. Using semantics for automatic enforcement of access control policies among dynamic coalitions. In *Proceedings of the ACM Symposium on Access Control Models and Technologies (SACMAT'07)*. 235–244.

WHITE, G. AND GRANADO, N. 2009. Developing a community cyber security incident response capability. In *Proceedings of the 42nd Hawaii International Conference on System Sciences (HICSS'09)*. 1–9.

WIKIPEDIA. 2009. Analog hole.

WONG, C. K., GOUDA, M., AND LAM, S. S. 1998. Secure group communications using key graphs. *IEEE/ACM Trans. Netw*. 68–79.

XRML. 2001. eXtensible rights Markup Language. www.xrml.org.

ZEILENGA, K., ED. ET AL. 2006. Lightweight Directory Access Protocol (LDAP): Tech. Spec. Road Map. http://www.potaroo.net/ietf/idref/draft-zeilenga-ldap-assert/.

Received March 2010; revised October 2010; accepted February 2011