A Trust Management Approach for Flexible Policy Management in Security-Typed Languages

Sruthi Bandhakavi
Dept. of Computer Science
Univ. of Illinois at Urbana Champaign
sbandha2@uiuc.edu
and
William Winsborough
Dept. of Computer Science
Univ. of Texas at San Antonio
wwinsborough@acm.org
and
Marianne Winslett
Dept. of Computer Science
Univ. of Illinois at Urbana Champaign
winslett@cs.uiuc.edu

Early work on security-typed languages required that legal information flows be defined statically. More recently, techniques have been introduced that relax these assumptions and allow policies to change at run-time. For example, the Rx language uses a policy language based on RT, a trust management framework for representing authorization policies. While Rx made significant strides toward the goal of allowing policy updates in security-typed languages, in this paper we observe that certain design choices of Rx violate the privacy and autonomy requirements of principals in trust management systems, thus making decentralized control over information difficult. To address these problems, we propose RTI, a new security-typed language. In addition to avoiding prior pitfalls, RTI's most distinguishing characteristic is that it supports fine-grained specification of security for dynamic policy. We also provide a proof of noninterference for RTI.

Categories and Subject Descriptors: D.3.3 [**Programming Languages**]: Language Constructs and Features; F.3.1 [**Theory of Computation**]: Specifying and Verifying and Reasoning about programs

General Terms: Design, Languages, Security, Theory

1. INTRODUCTION

Increasingly, applications must operate on data that is subject to regulatory and corporate requirements regarding confidentiality and integrity. From medical data to customer profiles, the exact ways in which the data can be used and modified depends on the stated wishes of the client and the terms under which the data were collected. Even the client's decisions on how the data can be used may be sensitive and subject to data integrity concerns and regulatory oversight. Furthermore, the wishes of the client may change over time, requiring corresponding changes in the usage policy associated with the data. Operating-system-based security can ensure that an application does not read high security data and then write to low security channels. However, this granularity is too coarse for individual

This research has been partly funded by SBC BYU 03-0114 and by NSF award CCF-0524010.
UIUC Techreport UIUCDCS-R-2008-2961

applications that must work with data having many different usage policies, such as sets of medical records or customer profiles.

EXAMPLE 1 (MEDICAL RECORDS MANAGEMENT). *A hospital patient Alice may have a policy allowing only the doctors in the hospital and her adult daughter to view her medical record. She may also prefer her stay in the hospital to be kept private. If Alice wants a specialist from another hospital to look at her record, she should be able to specify a new policy to this effect.*

These applications typically process data belonging to several distinct individuals or organizations with different privacy and authority requirements. Within limits posed by legal requirements, individuals need to have autonomy in defining portions of policy that control how their data is used. To accommodate decentralized control over information, we also need to minimize the amount of coordination required between different individuals or organizations as they work to define the policies. The individuals may also choose to keep their information private and we need to provide flexible ways to define such policies.

EXAMPLE 2 (REQUIREMENTS FOR A POLICY MANAGER). *Within certain limits imposed by regulations, Alice should be able to specify her own policies for how her data can be used. She may choose to delegate to the hospital some authority to control who can view and modify her data. For example, Alice can set up her policies so that when a new doctor joins the hospital, Alice's records can be shown to him without any change to her policies. However, Alice's policy should be sufficiently well protected that a medical-supplies manufacturer cannot view or modify her policy.*

To program such an application, we need a way to effectively define and enforce such policies in the program. Trust management frameworks provide a way to define policies and credentials for distributed authorization, and are a good candidate for defining the policies for applications such as those mentioned above. Language-based security research seeks to provide flexible and efficient methods to automatically enforce fine-grained confidentiality and integrity policies during program execution. For the past decade, researchers in this area have worked on the problems of how to specify a desired policy in a programming language, how to enforce the policy efficiently and, more recently, how to deal with changes in policy.

Security-typed languages have been one of the most active branches of language-based security research over the last decade. In this approach, the data types of program variables are annotated with security labels, and a lattice label ordering is used to specify the legal information flows among labeled variables. The type-checking rules verify that information flow in the program adheres to the label ordering and type-checking thereby enforces the security policy. If the program type-checks with respect to a policy, it does not have illegal information flows. This enforces very fine-grained (variable-level) information flow control.

While early work in this area required that legal information flows be defined quite statically, in recent years new techniques have relaxed these assumptions, enabling policies to change [Broberg and Sands 2006; Hicks et al. 2005; Swamy et al. 2006; Tse and Zdancewic 2004; Zheng and Myers 2004]. Two of these works introduced techniques whereby the interpretation of the information flow policies depends on the principals interacting with the system [Tse and Zdancewic 2004; Zheng and Myers 2004]. Two others introduced methods for updating the policies that define legal information flows within the program [Hicks

et al. 2005; Swamy et al. 2006]. One of the most recent works, Rx [Swamy et al. 2006], used a policy language based on RT, a trust management framework for representing authorization policies [Li et al. 2002]. Although Rx makes several important innovations, as we shall see, its design makes several mistakes with respect to subtle, but essential aspects of the integration of security-typed languages and trust management.

In Rx, the security type of a variable x is given by two RT roles, one that specifies x's confidentiality policy (i.e., who can read the value of x), and one for x's integrity policy (i.e., who trusts the integrity of the value of x). A program's security policy is a set of RT policy statements. Rx employs static type-checking rules and also novel dynamic semantics. At run-time, the executing program has access to a set of policy statements. These statements form the dynamic policy and can be used to determine the control flow of the program. Rx also includes constructs to add and delete policy statements dynamically during program execution. The policy changes affect the semantics of a role and therefore change the ways in which program variables can legally be used. In a sense, using roles as the basis of security labels introduces a level of indirection that enables policy changes to change program behavior without modifying the program. While there are several security policy models that are based on roles, notably RBAC [Sandhu et al. 1996], control over policy definition in these models is usually highly centralized, which is contrary to our design goals.

EXAMPLE 3 (RX POLICY UPDATES). *Suppose that Alice wants specialist Bob from another hospital to view her record. In an Rx program, she can add the new statement Alice.record ← Bob on the fly. If Bob specializes in a disease that Alice does not want everyone to know that she has, then Alice will want to keep this policy statement private. (Though, of course, if it is to have any effect, the policy statement cannot be kept private from Bob.)*

The previous example shows that security policies themselves can potentially become inappropriate conduits of information. To avoid this problem, Rx gives roles the same kinds of protection as provided for program variables, in the form of a pair of metapolicies $(C(\rho), I(\rho))$ associated with each role ρ. Confidentiality policy $C(\rho)$ identifies principals authorized to know the members of ρ, and integrity policy $I(\rho)$ identifies principals that trust that the membership of ρ is defined correctly. The latter is significant principally when values given to output channels[1] depend on how that membership is defined. In that case, everyone that trusts the data integrity of the value of the output channel has to trust the way the membership of ρ is defined. As explained in section 8, this would mean that Rx allows inappropriate principals to influence the definitions of roles.

Also, in the kinds of applications trust management is intended for, the choice of roles as the unit that is protected has undesirable ramifications owing simply to the coarseness the unit. For example, every member of an Rx role is entitled to learn who all the other members of that role are. In the real world, role membership is often sensitive. But in Rx, not only is every role member authorized to know that one is in the role, but also, because of information-flow requirements, to know *why* one is in the role. If one gets a certain service because one is HIV positive, one probably does not want everyone else authorized for the service to know this.

[1] We use the term *output channel* to refer to observable effects of program execution, such as changes to variable values or the security policy.

To address these problems, we propose RTI, a new security-typed language that builds on Rx's approach to the use of RT roles as security types, while addressing the privacy and autonomy concerns of individual principals[2]. RTI uses role-based labels in programs, and uses trust management principles to allow policy creators to control who can access and update each individual policy statement. A principal needs to be authorized to know the existence of policy statements that prove his membership in each of his roles; he does not need to know the other members in the role. RTI also introduces language constructs for querying and modifying policy statements that are persistent run-time values protected according to security labels that are themselves run-time values. These constructs are designed to control information flow via these policy statements in a manner that interacts cleanly with the rest of the language constructs, in which information flow is controlled by purely static means. The main contributions of this paper are:

—We observe that privacy of role membership and autonomy over defining ones own policy are important requirements for flexible policy management in a decentralized environment.

—We present the RTI programming language, which embodies design choices that satisfy these requirements.

—RTI introduces language constructs that are designed to control information flow via policy statements that are persistent run-time values and that are protected according to security labels that are themselves run-time values.

—We present a noninterference theorem for RTI.

—We present a comprehensive study of the design choices in Rx and identify their shortcomings with respect to the above requirements.

The rest of the paper is organized as follows. Section 2 reviews RT. Section 3 presents the key design features of RTI that distinguish it from prior work. Section 4 introduces the RTI language. Section 5 presents recommended idioms for uniformly specifying labels for policy statements and variables. Section 6 gives an example program written in RTI that illustrates how the dynamic policy can be changed in RTI and how doing so affects the program execution. The example is used to illustrate the static and dynamic semantics, as well as the recommended security-label idiom. Section 7 presents the RTI noninterference theorem. Section 8 describes the problems with Rx and how RTI resolves them. Section 9 describes related work. We conclude in section 10. Our noninterference theorem and proof follow that of Rx and therefore we omit that from the main paper due to lack of space.

2. BACKGROUND: RT OVERVIEW

RT is a family of languages [Li et al. 2002; Li and Mitchell 2003] whose simplest member has been called $\mathsf{RT}[\,]$ because it contains no optional features (which, if they existed, would be listed between the brackets) [Li et al. 2005]. The basic constructs in RT are *principals* and *role names*. We use A, B, D, and P, often with subscripts, to denote principals and r, u, and w to denote role names. A *role* is given by a principal followed by a role name, separated by a dot, e.g., $A.r$ and $B.r_1$; we use ρ to denote roles. A role defines a set of principals that are its members. Each principal A has the sole authority to designate the members of each role of the form $A.r$, which it does by signing policy statements.

[2]RTI stands for Role-based Trust management for Information flow.

Each statement has the form $A.r \leftarrow e$, in which e is a principal or a role. We read "$\leftarrow$" as "includes", and say the policy statement *defines* $A.r$.

— *Simple Membership*: $A.r \leftarrow D$
This statement means that A asserts that D is a member of A's r role.

— *Simple Inclusion*: $A.r \leftarrow B.r_1$
This statement means that A asserts that its r role includes (all members of) B's r_1 role. This represents a delegation from A to B, as B may add principals to become members of the role $A.r$ by issuing statements defining $B.r_1$.

A *policy* π is a set of policy statements. The semantics of roles is given by translating the statements in π into a set $SP(\pi)$ of definite Horn clauses that obey the Datalog restrictions shown below. We use x, y, and z to denote Datalog variables, which in this case range over principals.

For each $A.r \leftarrow D$ in π, add to $SP(\pi)$
$$m(A, r, D) \qquad (m1)$$
For each $A.r \leftarrow B.r_1$ in π, add to $SP(\pi)$
$$m(A, r, x) :- m(B, r_1, x) \quad (m2)$$

The semantics of a Datalog program such as $SP(\pi)$ can be defined through several equivalent approaches. Viewing $SP(\pi)$ as a set of first-order sentences, we write $SP(\pi) \models m(A, r, D)$ when $m(A, r, D)$ is logically entailed by $SP(\pi)$[3]. Given a policy π, the semantics of a role $A.r$ is given by $[\![A.r]\!]_{SP(\pi)} = \{x \mid SP(\pi) \models m(A, r, x)\}$.

3. RTI DESIGN CHOICES

In this section we motivate the manner in which RT[ ] is incorporated into RTI by appealing to established trust-management principles.

Role owners should have autonomy. A readers-oriented view of integrity says that "if the assignment $x := y$ is to be allowed, every principal who trusts the integrity of the value of x must also trust the integrity of y". A writers-oriented view says that "every principal that is trusted to define the value of y must also be trusted to define the value of x." In most contexts, a readers-oriented view of integrity is essentially interchangeable with a writers-oriented view, which says that everyone who is allowed to define the value of y must also be allowed to define the value of x [Li et al. 2003; Tse and Zdancewic 2004]. However when RT roles are used to define these sets of principals, the readers-oriented view, used in prior work in the area [Swamy et al. 2006], runs counter to the need that control over the use of information be retained by the information's owner. We illustrate this conflict with the following example.

EXAMPLE 4 (READERS-ORIENTED INTEGRITY). *Alice should allow the doctors at her hospital to write entries in her medical record. Assume the integrity labels of the updated record y and Alice's permanent record x are Hospital.doctor and Alice.record, respectively. If the assignment $x := y$ is to be allowed, then every member of Hospital.doctor must also be a member of Alice.record. This is achieved in* RT[ ] *if the policy includes the statement Hospital.doctor ← Alice.record. Because the role in the head of this*

[3]I.e., all models of $SP(\pi)$ are models of $m(A, r, D)$. Note that for atomic formulas L, $SP(\pi) \models L$ just in case L is in the least Herbrand model [Lloyd 1993]. However, this only holds for atomic formulas (i.e., formulas with no logical connectives or quantifiers).

rule belongs to the hospital, according to RT's design, the hospital controls whether or not this statement is in the policy, so Alice has no control over whether the doctor is authorized to modify her record.

A writers-oriented view of integrity fits RT much better.

EXAMPLE 5 (WRITERS-ORIENTED INTEGRITY). *Under a writers-oriented approach to integrity, once Alice adds the policy statement* $Alice.record \leftarrow Hospital.doctor$*, the value of a variable* y *having type Hospital.doctor can be assigned to a variable* x *having type Alice.record. With writers-oriented integrity, Alice controls who can change the information in her medical record.*

These observations give a clear preference for the writers-oriented interpretation when working in a trust management-style policy language such as RT[ ]. RTI integrity labels take the writers-oriented approach for this reason.

Privacy of role-members should be preserved. Focusing on confidentiality for the moment and ignoring integrity, if the confidentiality label of variable x consists of role ρ, then members of ρ must be permitted to know that they are members of ρ. Otherwise their membership in ρ cannot permit them to view variable x as intended. To see this, assume the principal in question knows that x has confidentiality ρ. Then, if the principal can see the value x, he knows he is in ρ. If the latter is not permitted, he cannot be permitted to see x.

Prior work using RT[ ] roles as labels [Swamy et al. 2006] has protected policy at the role level. This means that one cannot be permitted to know that one is a member without also being permitted to know of every other member's membership. That is to say, it is not possible to differentiate between a role member's right to know about his own membership and his right to know that the other members of ρ are members of ρ. This can be quite undesirable: suppose $Hospital.HIVpatients$ contains all HIV patients at the hospital; if role-level protection is used, every member of this role needs to have permission to know of every other member's membership.

To avoid this and related problems, RTI protects individual policy statements rather than memberships of entire roles. This obviates the need for each member of a role to be authorized to know all the role's members. Instead, each member of $[\![A.r]\!]_{SP(\Pi)}$ need only be able to know about each statement that is needed to prove his own membership. The implication of this is that for each credential of the form $A.r \leftarrow D$, D must be among the principals that are permitted to know this statement exists. Similarly, each member of $[\![B.r_1]\!]_{SP(\Pi)}$ must be permitted to know that $A.r \leftarrow B.r_1$ exists.

New techniques are needed to manage control dependence. Because the labels of policy statements are dynamic values in RTI, a type-based solution to avoiding illegal flows requires new language features. The approach we use is to establish statically a bound on the security level of policy statements that will be considered when the policy is consulted at run-time. This bound is used to ensure that low security assignments and policy updates will not have control dependence on high security policy statements. In fact, suitable values for these bounds are easily inferred by using the type system presented in the next section.

4. THE RTI LANGUAGE

RTI is a simple imperative language with additional features that support (1) role-based security labels on variables (static labels) and on policy statements (run-time labels), (2) dynamic updates to security policy, and (3) conditions based on queries against that policy. In this section, we present RTI syntax, issues related to protecting policy statements with dynamic security labels, RTI's label ordering, its operational semantics, and its static semantics (i.e., type-checking rules).

4.1 RTI Syntax

principal	P		
roles	ρ	::=	$P.r$
atomic labels	κ	::=	$\rho \mid P$
policy statements	s	::=	$\rho \leftarrow \kappa$
queries	q	::=	$\kappa_1 \sqsubseteq \kappa_2$
one-property confidentiality labels	CL	::=	$\kappa \mid CL_1 \sqcup CL_2$
one-property integrity labels	IL	::=	$\kappa \mid IL_1 \sqcap IL_2$
two-property labels	ℓ	::=	(CL, IL)
one-property confidentiality bounds	CB	::=	$CL \mid CB_1 \sqcap CB_2$
one-property integrity bounds	IB	::=	$IL \mid IB_1 \sqcup IB_2$
two-property bound	b	::=	(CB, IB)
types	t	::=	$\ldots \mid \mathrm{pol}$
security types	τ	::=	t_ℓ
policy	Π	::=	$\{\langle s_1, \ell_1\rangle, \ldots, \langle s_m, \ell_m\rangle\}$
policy context	Q	::=	$\{\langle q_1, b_1\rangle, \ldots, \langle q_n, b_n\rangle\}$
update	δ	::=	$\texttt{add} \mid \texttt{del}$
updates	Δ	::=	$\delta(\langle s, \ell\rangle) \mid \delta(\langle s, \ell\rangle), \Delta$
expressions	E	::=	$\texttt{true} \mid \texttt{false} \mid x \mid E \oplus E$
statements	S	::=	$\texttt{skip} \mid S; S \mid \texttt{try}_{(Q,\mathrm{pc})} S \mid x := E \mid \texttt{while}\,(E)\,S \mid$ $\texttt{if}\,(E)\,S\,S \mid \texttt{if}\,(\langle q, b\rangle)\,S\,S \mid \texttt{update}\ \Delta$

Fig. 1. Syntax of RTI.

statements	S	::=	$\texttt{skip}$	$\mid \texttt{if}\,(E)\,S\,S$	$\mid \texttt{if}\,(\langle q, b\rangle)\,S\,S$	$\mid \texttt{update}\ \Delta$
	S'	::=	$x := E$	$\mid \texttt{skip}$	$\mid S'; S'$	$\mid \texttt{if}\,(\langle q, b\rangle)\,S'\,S'$
			$\mid \texttt{if}\,(E)\,S'\,S'$	$\mid \texttt{while}\,(E)\,S'$		
	S''	::=	S'	$\mid \texttt{try}_Q\ S$	$\mid S''; S''$	

Fig. 2. Alternate statement syntax of RTI.

Figure 1 presents a grammar for RTI syntax. Like most of the previous approaches for security-typed languages, RTI adds a static security label ℓ to each each program variable's data type t, forming t_ℓ. As is commonly done, a security label ℓ in RTI is actually a pair of labels $\langle CL, IL\rangle$. A join of principals and roles, CL represents the confidentiality level; IL is a meet of principals and roles, and represents the integrity level of the associated variable. CL denotes all the principals that can view the information bearing that label. IL

denotes all the principals that can modify the contents of the variable—the writers-oriented interpretation of integrity.

In RTI, policy statements also have this kind of two-part labels, though in this case, the labels are dynamic objects as well as static. As we discuss below in section 4.5, the type-checking rules also use security bounds, b, to annotate the statements of the program. Security bounds resemble security labels, but have a slightly more general syntactic form, as explained below in section 4.3. Apart from the normal data-types in the programming language, the RTI language also has a special type 'pol' for typing a set of updates.

A policy statement, s, is an $\mathsf{RT}[\,]$ statement. In the context of confidentiality, $\rho_1 \leftarrow \kappa$ means that the value of a variable with label ρ_1 can flow to a variable labeled κ. In the context of integrity, $\rho_1 \leftarrow \kappa$ means that a variable labeled ρ_1 can be written to by a variable labeled κ.

The policy, Π, is a run-time structure that consists of a set of (policy statement, label) pairs $\langle s, \ell \rangle \in \Pi$. The presence of $\langle s, \ell \rangle \in \Pi$ indicates that observers that have a security level dominating ℓ are permitted see that s is in Π. Note that if any observer sees s as being in Π, then s is in fact in Π. It is impossible for the view of a low observer to show s is in Π, while a high observer sees that s is not in Π. The statements s in Π define the label ordering.

The RTI programming language consists of three new statements in addition to a simple imperative programming language: a conditional statement that queries the dynamic policy, a policy update statement, and a try statement that is used to safely encapsulate policy updates. A policy query, q, tests whether two atomic labels are ordered. The then-branch of the conditional containing q is compiled under the assumption that this relationship holds. In this way, a static approximation of the dynamic policy is built up within nested conditionals. This approximation Q, consisting of a set of order relationships, is used by the type system to determine the security of information flows.

The policy update statement supports addition and deletion of policy statements during the execution of the program. Policy updates must be embedded with in a try statement, $\mathtt{try}_Q\ S$, which specifies a policy context Q containing all the policy queries used in its body S.

This illustrates that fact that programs must conform to the syntax shown in figure 2, which is more restrictive than the one given in figure 1. In a try statement, the statement S must consist of conditionals, query conditionals and update statements only. (In particular, sequential composition of statements is not permitted.) While one could enforce these restrictions syntactically, as shown in figure 2, it simplifies the proofs if we reduce the number of syntactic categories by using the syntax shown in figure 1, and instead enforce these restrictions in the static typing rules. We show how this is done below in section 4.5. The try statement acts as a placeholder, enabling the syntactic restrictions to ensure that should an update statement change the policy in such a way as to violate the assumptions under which the executing code segment was type-checked, there are no further statements to execute in that code segment, so there is no opportunity for information-flow restrictions to be violated. Specifically, the syntactic restrictions ensure that after the update occurs, no other changes to the configuration can be made prior to exiting all policy-query conditionals.

4.2 Supporting Fine-grained Metapolicy

In RTI, each policy query q is annotated with a bound $b = (CB, IB)$ that determines which policy statements are used when evaluating the query at run-time. A statement $\langle s, \ell \rangle \in \Pi$ is used if $\ell \sqsubseteq_2 b$. This ensures that any policy statement that is used is less confidential that CB, and that its integrity must be at least as great as IB. The purpose of using only such policy statements is to ensure that subsequent modifications to public or high integrity output channels (variables or policy statements) does not depend on sensitive or unreliable policy statements used to evaluate the query.

The query bound b is inferred using RTI's static semantics in a manner that ensures it is dominated by the labels of all output channels that are modified in either branch of the query conditional. In this inference process, RTI's static semantics defines the bound on a policy query to be the meet of the labels of the output channels that can be modified within the scope of the policy-query conditional. This has the required effect of ensuring that no information flow restrictions will be violated because of a low security output channel depending on a high security policy statement.

As mentioned in the previous section, RTI's static semantics uses policy-query conditionals to derive a static approximation of the dynamic policy that is in force when a given code region is reached. In the then-branch, the query is assumed to hold. However, within the else-branch one cannot assume that the query does not hold, since in general not all policy statements are used to evaluate the query. In this sense, query evaluation is *incomplete* in RTI.

4.3 Label Ordering

The incompleteness of query evaluation identified above places a significant requirement on the order relation $\sqsubseteq$. It needs to be monotonic in the sense that adding statements to the policy can cause a false query to become true, but not a true query to become false. This is necessary because the then-branch of a conditional that tests whether a query is true is type-checked under the assumption that the ordering relationship given by the query in fact holds. With a non-monotonic order relation, it would be possible for the query to evaluate to true because only low security policy statements can be considered in the context where the query happens to occur, while it would fail to hold if the full policy were used. Clearly this would make it unsound to type-check the then-branch under the assumption that the query holds.

Unfortunately, the ordering relation used in Rx, the prior work using RT[ ]-based security labels, is not monotonic. In Rx, given a dynamic policy π, the judgment $\pi \vdash \rho_1 \sqsubseteq_{Rx} \rho_2$ is defined to hold if $[\![\rho_1]\!]_{SP(\pi)} \supseteq [\![\rho_2]\!]_{SP(\pi)}$. The following example illustrates that this ordering is not monotonic.

EXAMPLE 6. *Consider the policies* $\pi_1 = \{A.r \leftarrow D, B.r_1 \leftarrow D\}$ *and* $\pi_2 = \pi_1 \cup \{B.r_1 \leftarrow E\}$. *We have* $\pi_1 \vdash A.r \sqsubseteq_{Rx} B.r_1$, *but* $\pi_2 \nvdash A.r \sqsubseteq_{Rx} B.r_1$.

To achieve monotonicity, RTI uses a stronger ordering relation that is based on logical entailment, rather than on subset ordering. In RTI, $\pi \vdash A.r \sqsubseteq_{RTI} B.r_1$ if $SP(\pi) \models \forall x.m(B, r_1, x) \Rightarrow m(A, r, x)$. Henceforth in this paper we omit the subscript, writing simply $\sqsubseteq$ for $\sqsubseteq_{RTI}$. This ordering requires that in all models of π, the implication holds—not only in the minimal model.

EXAMPLE 7. *Referring again to the policies above, we have* $\pi_1 \nvdash A.r \sqsubseteq B.r_1$, *and*

also $\pi_2 \not\vdash A.r \sqsubseteq B.r_1$. *In particular,* $\pi_1 \not\vdash A.r \sqsubseteq B.r_1$ *because there are models of* $SP(\pi)$ *that, for example, make* $m(B, r_1, F)$ *hold without making* $m(A, r, F)$ *hold. However, taking* $\pi_3 = \{A.r \leftarrow B.r_1, B.r_1 \leftarrow D\}$ *and* $\pi_4 = \pi_3 \cup \{B.r_1 \leftarrow E\}$, *we have* $\pi_3 \vdash A.r \sqsubseteq B.r_1$, *and* $\pi_4 \vdash A.r \sqsubseteq B.r_1$.

The policy π is a set of policy statements $\{s_1, \ldots, s_n\}$. It should be thought of consisting of the policy statements that are visible at some security level. We use a bound function, given by the notation $\Pi \lfloor_b$, to calculate such a π at run-time. The expression $\Pi \lfloor_b$ denotes the subset of Π consisting of statements that are visible at the security level given by b. As we must take care not to leak information about high security policy statements that could help prove that some ℓ' is dominated by b, we define $\Pi \lfloor_b$ to be the greatest fixpoint of a monotonic function $f_b : \wp(\Pi) \to \wp(\Pi)$, projected onto the statements:

$$\Pi \lfloor_b = \{s \mid \exists \ell'.\langle s, \ell' \rangle \in f_b^{\Pi} \downarrow^{\omega}\}$$
$$\Pi \lfloor_{\top} = \{s \mid \exists \ell'.\langle s, \ell' \rangle \in \Pi\}$$
$$f_b^{\Pi} \downarrow^0 = \Pi \quad f_b^{\Pi} \downarrow^{i+1} = f_b^{\Pi}(f_b^{\Pi} \downarrow^i) \quad f_b^{\Pi} \downarrow^{\omega} = \sqcap_{i<\omega} f_b^{\Pi} \downarrow^i$$
$$f_b^{\Pi}(\widehat{\Pi}) = \{\langle s, \ell' \rangle \in \Pi \mid \widehat{\Pi} \lfloor_{\top} \vdash \ell' \sqsubseteq_2 b\}$$

EXAMPLE 8. *Let* $\Pi = \{\langle A.r_1 \leftarrow B.r_2, \langle C.r_3, A\rangle\rangle, \langle D.r_4 \leftarrow E.r_5, \langle A.r_1, D\rangle\rangle\}$ *and let* $\ell = (B.r_2, A \sqcap D)$. *Then* $f_\ell^{\Pi} \downarrow^0 = \Pi$, $f_\ell^{\Pi} \downarrow^1 = \{\langle D.r_4 \leftarrow E.r_5, A.r_1\rangle\}$, *and* $f_\ell^{\Pi} \downarrow^2 = \emptyset$.

As shown in figure 1, one-property confidentiality and integrity labels are, respectively, joins and meets of individual atomic labels. One-property confidentiality and integrity bounds are, respectively, meets of confidentiality labels and joins of integrity labels. If we take equivalence classes of labels with respect to logical equivalence, this gives us semi-lattices (specifically, a join-semi-lattice for confidentiality labels and meet-semi-lattice for integrity labels.) Taking equivalence classes of each kind of bound yields a full lattice. Fortunately it is not necessary to compare arbitrary elements, as doing so is in general intractable. We show presently that all the comparisons that might occur during the execution of the program can be evaluated by straight-forward decomposition.

Tractable label ordering. We extend $\sqsubseteq$ to single-property labels and bounds as follows. Let $L ::= CL \mid IL \mid CB \mid IB$.

$$\pi \vdash L_1 \sqsubseteq L_2 \text{ if } SP(\pi) \models \forall x.\varphi_{L_1}(x) \Leftarrow \varphi_{L_2}(x)$$

in which

$$\begin{aligned}
\varphi_P(x) &\equiv x = P \\
\varphi_{P.r}(x) &\equiv m(P, r, x) \\
\varphi_{L_1 \sqcup L_2}(x) &\equiv \varphi_{L_1}(x) \wedge \varphi_{L_2}(x) \\
\varphi_{L_1 \sqcap L_2}(x) &\equiv \varphi_{L_1}(x) \vee \varphi_{L_2}(x)
\end{aligned}$$

This order relation is easily computed as follows (whenever it is obvious from the context, we elide $\pi \vdash$ in the following computations):

$$\begin{aligned}
\kappa_1 \sqsubseteq \kappa_2 &\equiv \kappa_1 = \kappa_2 \vee \exists \rho_3.(\rho_3 \leftarrow \kappa_2 \in \pi \wedge \pi \vdash \kappa_1 \sqsubseteq \rho_3)) \\
\kappa \sqsubseteq CL_1 \sqcup CL_2 &\equiv \kappa \sqsubseteq CL_1 \vee \kappa \sqsubseteq CL_2 \\
CL_1 \sqcup CL_2 \sqsubseteq CL_3 &\equiv CL_1 \sqsubseteq CL_3 \wedge CL_2 \sqsubseteq CL_3 \\
CL \sqsubseteq CB_1 \sqcap CB_2 &\equiv CL \sqsubseteq CB_1 \wedge CL \sqsubseteq CB_2 \\
IL_1 \sqcap IL_2 \sqsubseteq \kappa &\equiv IL_1 \sqsubseteq \kappa \vee IL_2 \sqsubseteq \kappa \\
IL_1 \sqsubseteq IL_2 \sqcap IL_3 &\equiv IL_1 \sqsubseteq IL_2 \wedge IL_1 \sqsubseteq IL_3 \\
IB_1 \sqcup IB_2 \sqsubseteq IL &\equiv IB_1 \sqsubseteq IL \wedge IB_2 \sqsubseteq IL
\end{aligned}$$

Ordering relationships can be evaluated efficiently because we never have need to evaluate relationships of the form $CB_1 \sqcap CB_2 \sqsubseteq CB_3$ or $IB_1 \sqsubseteq IB_2 \sqcup IB_3$.

We use $\sqsubseteq_2$ for the order over two-property labels and bounds. Two-property labels are ordered by

$$(CL, IL) \sqsubseteq_2 (CB, IB) \equiv CL \sqsubseteq CB \wedge IB \sqsubseteq IL$$

Notice the contra-variance with the integrity part of the label; $\ell_1 \sqsubseteq_2 \ell_2$ means that data labeled ℓ_1 has lower confidentiality but higher integrity than data labeled ℓ_2. This is the circumstance in which information flow from data labeled ℓ_1 to data labeled ℓ_2.

We use the following projection operators, which extract the one-property labels and bounds represented by a two-property label and bounds.

$$\begin{array}{ll} |(CL, IL)|_C = CL & |(CL, IL)|_I = IL \\ |(CB, IB)|_C = CB & |(CB, IB)|_I = IB \end{array}$$

The join and meet of two-property labels and bounds are as follows:

$$\begin{array}{l} \ell_1 \sqcup \ell_2 = (|\ell_1|_C \sqcup |\ell_2|_C, |\ell_1|_I \sqcap |\ell_2|_I) \\ b_1 \sqcap b_2 = (|b_1|_C \sqcap |b_2|_C, |b_1|_I \sqcup |b_2|_I) \end{array}$$

4.4 Operational Semantics

$$\frac{\mathcal{E}.\Psi = \cdot \quad \Psi' = (\texttt{try}_Q S)}{\mathcal{E}, \texttt{try}_Q S \longrightarrow \mathcal{E}[\Psi = \Psi'], \texttt{try}_Q S} \text{ (E-Tr1)} \qquad \frac{\mathcal{E}.\Psi \neq \cdot \quad \mathcal{E}, S \longrightarrow \mathcal{E}', S_1}{\mathcal{E}, \texttt{try}_Q S \longrightarrow \mathcal{E}', \texttt{try}_Q S_1} \text{ (E-Tr2)}$$

$$\frac{\mathcal{E}.\Psi \neq \cdot}{\mathcal{E}, \texttt{try}_Q \texttt{skip} \longrightarrow \mathcal{E}[\Psi = \cdot], \texttt{skip}} \text{ (E-Tr3)} \qquad \frac{\Pi' = \mathit{update}(\mathcal{E}.\Pi, \Delta) \quad \mathcal{E}.\Psi = (\texttt{try}_Q S)}{\mathcal{E}, \texttt{update}\ \Delta \longrightarrow \mathcal{E}[\Pi = \Pi'], \texttt{skip}} \text{ (E-Up)}$$

$$\frac{(\mathcal{E}.\Pi)\lfloor_b \vdash q \Rightarrow j = 1 \quad (\mathcal{E}.\Pi)\lfloor_b \nvdash q \Rightarrow j = 2}{\mathcal{E}, \texttt{if}\ (\langle q, b\rangle)\ S_1\ S_2 \longrightarrow \mathcal{E}, S_j} \text{ (E-IfQ)}$$

In which $\mathit{update}(\Pi, \{\texttt{add}(\langle s, \ell\rangle)\} \cup \Delta) = \mathit{update}(\Pi \cup \{\langle s, \ell\rangle\}, \Delta)$
$\mathit{update}(\Pi, \{\texttt{del}(\langle s, \ell\rangle)\} \cup \Delta) = \mathit{update}(\Pi \backslash \{\langle s, \ell'\rangle \mid \langle s, \ell'\rangle \in \Pi \wedge \Pi\lfloor_{\ell'} \vdash \ell \sqsubseteq_2 \ell'\}, \Delta)$
$\mathit{update}(\Pi, \emptyset) = \Pi$

Fig. 3. Operational semantics

The operational semantics of RTI is shown in figure 3. Each rule in it contributes to the definition of the relation $\mathcal{E}, S \longrightarrow \mathcal{E}', S'$, which defines one step of execution of the program. The configuration $\mathcal{E}$ is defined as follows:

$$\begin{array}{rl} \textbf{execution configuration} & \mathcal{E} ::= (\Pi; M; \Psi) \\ \textbf{dynamic snapshot} & \Psi ::= (S'')|\cdot \end{array}$$

An execution configuration $\mathcal{E}$ consists of the current dynamic policy, Π, the memory state, M, and a dynamic snapshot, Ψ, consisting of the $\texttt{try}_Q$ statement in which the current statement being executed is nested, if any. Ψ is used to determine whether the execution is inside a `try` block.

The simple imperative programming language constructs follow the standard one-step semantics and therefore are omitted here. The first four rules in figure 3 are used to evaluate

the statements inside the try. (E-Tr1) represents the state transition when a try statement is reached. Before a try is entered, the try statement is preserved in the dynamic snapshot. (E-Tr2) is used when the statement inside the `try` takes a step. (E-Tr3) marks the end of the try. The dynamic snapshot is reverted to an empty state when the try is exited.

(E-IfQ) states that if the dynamic policy visible to observers at level b prove query q, then the then-branch of the conditional is executed, otherwise the else-branch is executed.

(E-Up) defines the execution of a policy update statement. Since a policy update can happen only inside a try, (E-Up) rule requires $\mathcal{E}.\Psi$ to be non-empty. As part of an update, the result of any query in Q could change as a result of the policy update. If so, it is possible that continuing to execute normally would execute code that was type-checked under assumptions about the policy that no longer hold after the policy update. Doing so would be unsafe. Therefore, we prevent any other memory or policy update statement to occur inside the same conditional branches as the update statement using our static typing rules in section 4.5.

In (E-Up), the function *update*(Π, Δ) (see figure 3) updates the policy. Deletes are an interesting case of policy updates, which remove only those occurrences of a statement that are at least as protected as the label in the update. If the parameters to a `del` are $\langle s, \ell' \rangle$, then any $\langle s, \ell \rangle \in \Pi$ will be removed just in case $\ell \sqsubseteq_2 \ell'$.

4.5 Static Semantics

The policy context, Ω, which is used to type-check the program, can be summarized as follows:

$$\begin{array}{rrcl} \textbf{typing context} & \Omega & ::= & (\Gamma; \mathrm{pc}; Q; \Phi) \\ \textbf{syntactic context} & \Phi & ::= & try \mid \cdot \mid * \end{array}$$

Ω is the static typing context, which includes a type binding for variables, Γ, the current program counter, pc, the query context, Q, and the syntactic context, Φ. In addition to labeling each program statement with a bound, RTI uses pc to keep track of the security level of the context. The pc value that is used to type check a given program statement is the least upper bound of the security labels of data inspected in conditionals that determine whether that statement is reached by execution. It is used to ensure that there is no unauthorized information flow from those conditionals to assignments or policy updates occurring in the statement. Similarly, the policy context, Q, keeps track of all the queries encountered to reach a particular program point.

A syntactic context is a flag having value *try*, $\cdot$ or $*$. It is used to encode whether the current context is inside a try statement (when $\Phi = try$), outside a try or a query conditional statement (when $\Phi = \cdot$), or inside a query conditional but not inside a try (when $\Phi = *$). This information is then used to enable the type rules to enforce the syntactic restrictions that are present in the syntax shown in figure 2, but absent from that in figure 1. Specifically, the try statement must not occur inside a query conditional, the assignment statement must not occur inside a try statement, a sequence of statements must not occur inside a try, and the update statement occurs only inside a try.

The static typing rules are given in figure 4. The static semantics of RTI are defined by the typing relations $\Omega \vdash E : t_\ell$ and $\Omega \vdash S'' : b\,\mathtt{cmd}$. An expression, E, has type t_ℓ, where t denotes the normal data type of the expression and ℓ is a two-property security label. The type of the expression is the least upper bound of the labels of all its sub-expressions. In $\Omega \vdash S'' : b\,\mathtt{cmd}$, b will be the greatest lower bound of the labels of all output channels

$$\frac{}{Q \vdash \kappa \sqsubseteq \kappa} \qquad \frac{Q \vdash \kappa_1 \sqsubseteq \kappa_2 \quad (\kappa_2 \sqsubseteq \kappa_3, b) \in Q}{Q \vdash \kappa_1 \sqsubseteq \kappa_3} \qquad \frac{Q \vdash \kappa \sqsubseteq CL_1}{Q \vdash \kappa \sqsubseteq CL_1 \sqcup CL_2}$$

$$\frac{Q \vdash \kappa \sqsubseteq CL_2 \sqcup CL_1}{Q \vdash \kappa \sqsubseteq CL_1 \sqcup CL_2} \qquad \frac{Q \vdash IL_1 \sqsubseteq \kappa}{Q \vdash IL_1 \sqcap IL_2 \sqsubseteq \kappa} \qquad \frac{Q \vdash IL_2 \sqsubseteq \kappa}{Q \vdash IL_1 \sqcap IL_2 \sqsubseteq \kappa}$$

$$\frac{Q \vdash CL_2 \sqsubseteq CL_3 \quad Q \vdash CL_1 \sqsubseteq CL_3}{Q \vdash CL_1 \sqcup CL_2 \sqsubseteq CL_3} \qquad \frac{Q \vdash CL \sqsubseteq CB_1 \quad Q \vdash CL \sqsubseteq CB_2}{Q \vdash CL \sqsubseteq CB_1 \sqcap CB_2}$$

$$\frac{Q \vdash IL_1 \sqsubseteq IL_2 \quad Q \vdash IL_1 \sqsubseteq IL_3}{Q \vdash IL_1 \sqsubseteq IL_2 \sqcap IL_3} \qquad \frac{Q \vdash IB_1 \sqsubseteq IL \quad Q \vdash IB_2 \sqsubseteq IL}{Q \vdash IB_1 \sqcup IB_2 \sqsubseteq IL}$$

$$\frac{Q \vdash |\ell_1|_C \sqsubseteq |\ell_2|_C \quad Q \vdash |\ell_2|_I \sqsubseteq |\ell_1|_I}{Q \vdash \ell_1 \sqsubseteq_2 \ell_2} \qquad \frac{Q \vdash CL_1 \sqcup CL_2 \sqsubseteq CL_3 \quad Q \vdash IL_3 \sqsubseteq IL_1 \sqcap IL_2}{Q \vdash (CL_1, IL_1) \sqcup (CL_2, IL_2) \sqsubseteq_2 (CL_3, IL_3)}$$

$$\frac{Q \vdash CL_1 \sqsubseteq CL_2 \sqcup CL_3 \quad Q \vdash IL_2 \sqcap IL_3 \sqsubseteq IL_1}{Q \vdash (CL_1, IL_1) \sqsubseteq_2 (CL_2, IL_2) \sqcup (CL_3, IL_3)} \qquad \frac{Q \vdash CL \sqsubseteq CB_1 \sqcap CB_2 \quad Q \vdash IB_1 \sqcup IB_2 \sqsubseteq IL}{Q \vdash (CL, IL) \sqsubseteq_2 (CB_1, IB_1) \sqcap (CB_2, IB_2)}$$

$$\frac{\Omega.Q \vdash \Omega.\mathrm{pc} \sqsubseteq_2 \ell}{\Omega \vdash \delta(\langle s, \ell \rangle) : \mathrm{pol}_\ell} \text{ (T-AddDel1)} \qquad \frac{\Omega \vdash \Delta : \mathrm{pol}_{b'} \quad \Omega.Q \vdash \Omega.\mathrm{pc} \sqsubseteq_2 \ell}{\Omega \vdash (\delta(\langle s, \ell \rangle), \Delta) : \mathrm{pol}_{\ell \sqcap b'}} \text{ (T-AddDel2)}$$

$$\frac{\begin{array}{c}\Omega \vdash x : t_{\ell_1} \quad \Omega \vdash E : t_{\ell_2} \quad \Omega.\Phi \in \{\cdot, *\} \\ \Omega.Q \vdash \ell_2 \sqsubseteq \ell_1 \quad \Omega.Q \vdash \Omega.\mathrm{pc} \sqsubseteq_2 \ell_1\end{array}}{\Omega \vdash x := E : \ell_1\ \mathtt{cmd}} \text{ (T-Assign)} \qquad \frac{\begin{array}{c}\Omega.\Phi \in \{\cdot, *\} \\ \Omega \vdash S_1 : b_1\ \mathtt{cmd} \quad \Omega \vdash S_2 : b_2\ \mathtt{cmd}\end{array}}{\Omega \vdash S_1; S_2 : (b_1 \sqcap b_2)\ \mathtt{cmd}} \text{ (T-Seq)}$$

$$\frac{\begin{array}{c}\Gamma; \mathrm{pc}; Q; \Phi \vdash E : t_\ell \\ \Gamma; \mathrm{pc} \sqcup \ell; Q; \Phi \vdash S_1 : b_1\ \mathtt{cmd} \quad \Gamma; \mathrm{pc} \sqcup \ell; Q; \Phi \vdash S_2 : b_2\ \mathtt{cmd}\end{array}}{\Gamma; \mathrm{pc}; Q; \Phi \vdash \mathtt{if}(E)\ S_1\ S_2 : b_1 \sqcap b_2\ \mathtt{cmd}} \text{ (T-IfE)}$$

$$\frac{\begin{array}{c}\Phi = try \Rightarrow \Phi' = try \quad \Phi \in \{\cdot, *\} \Rightarrow \Phi' = * \\ \Gamma; \mathrm{pc}; Q \cup \{\langle q, b \rangle\}; \Phi' \vdash S_1 : b_1\ \mathtt{cmd} \\ \Gamma; \mathrm{pc}; Q; \Phi' \vdash S_2 : b_2\ \mathtt{cmd} \quad b = b_1 \sqcap b_2\end{array}}{\Gamma; \mathrm{pc}; Q; \Phi \vdash \mathtt{if}(\langle q, b \rangle)\ S_1\ S_2\ :\ b\ \mathtt{cmd}} \text{ (T-IfQ)}$$

$$\frac{\Gamma; \mathrm{pc}; \emptyset; try \vdash S : b\ \mathtt{cmd}}{\Gamma; \mathrm{pc}; \emptyset; \cdot \vdash \mathtt{try}_{Q'} S : b\ \mathtt{cmd}} \text{ (T-Try)} \qquad \frac{\Gamma; \mathrm{pc}; Q; try \vdash \Delta : \mathrm{pol}_b}{\Gamma; \mathrm{pc}; Q; try \vdash \mathtt{update}(\Delta) : b\ \mathtt{cmd}} \text{ (T-Up)}$$

Fig. 4. Static type-checking rules

that can be modified inside S. This bound is important because policy statements that have labels not dominated by it cannot safely be used to evaluate policy queries that govern conditional execution of S. The impact of this will be seen in our discussion of the (T-IfQ) rule below.

The first set of rules in figure 4 specify the label ordering in RTI. The interpretation of the label ordering under the static policy context Q corresponds to the interpretation of the

ordering with respect to the dynamic policy Π as defined in Section 4.3.

The (T-AddDel1) rule types the expression with the policy type and the security level associated with the policy statement. The (T-AddDel2) rule specifies the security level of a set of add or del commands as the greatest lower bound of all the levels of individual add or del commands. Note that (T-AddDel1) and (T-AddDel2) verify that the labels of statements that are added or removed from the policy are at least as restrictive as the pc.

The (T-Assign), (T-Seq), and (T-IfE) give the typing rules for the normal assignment, sequential statements and the conditionals, respectively. For an assignment statement to type-check, the level of the output variable should dominate the level of the input expression, to disallow illegal direct flows. In addition, to prevent illegal control flows, the level of the output variable should dominate the pc. Additionally, since the level of the output channel is ℓ_1, the type of the assignment statement is ℓ_1 `cmd`. Checking whether the syntactic context is either $\cdot$ or * ensures that the assignment statement does not occur inside a try, where the context would be *try*. The same is the case in (T-Seq). The type rules for (T-Seq) and (T-IfE) are typed similar to those in previous approaches.

The (T-IfQ) rule defines the conditions for the query conditional statement to type-check. The value of b in the query can actually be inferred by using this rule, based on the values of b_1 and b_2 in the premise of this rule. The query $\langle q, b\rangle$ is added to Q, the policy context, for the purpose of type-checking the then-branch of the conditional. If both then- and else-branches type-check, they yield bounds b_1 and b_2 that correspond to the greatest lower bound of all the output channels in each of the respective branches. The bound $b = b_1 \sqcap b_2$ on the whole conditional is again a greatest lower bound, per the third premise. Notice that there is no need to modify the pc when descending into the branches of the conditional. This is because the operational semantics prevents information from policy statements that are used to evaluate the query q from flowing illegally to to output channels inside the conditional. The operational semantics does this by ensuring that q is evaluated by using only statements that are visible at security level b. Notice that the two branches of the conditional are type-checked under a syntactic context Φ', which is either *try* or *. If $\Phi' = try$, the query conditional is already inside a try statement, thereby disallowing other try statements inside it. Φ' is set to * when the query conditional is not inside a try; the * will prevent a try occurring inside this query conditional.

The (T-Try) rule says that if the statements inside the try block type-check, the try statement type-checks too. We also ensure that the try statement does not occur inside another try statement or a query conditional by requiring that it is type-checked under $\Phi = \cdot$. The (T-Up) rule defines the conditions under which the policy can be updated. The type of the update statement pol_b is given by the greatest lower bound of the types of all the policy statements in Δ. Also the update statement requires that its syntactic context be of the form *try*, to ensure that it occurs inside a try statement.

5. RECOMMENDED IDIOM FOR METAPOLICY

The RTI approach to protecting policy imposes some requirements on the labels of policy statements and variables. This section explains how these labels can be defined appropriately in RTI.

The first requirement, mentioned above in section 3, is that D should be able to see the statement $A.r \leftarrow D$ and all members of $B.r_1$ should be able to see the statement $A.r \leftarrow B.r_1$. Otherwise D and the members of $B.r_1$ would gain no benefit from these

respective policy statements. Furthermore, any variable x having confidentiality label $A.r$ should have as part of its integrity label that the writers trusted to participate in defining the membership of $A.r$ are trusted by x.

Consider an assignment $x := y$ in which the confidentiality labels of x and y are $A.r$ and $C.r_2$ respectively. To type-check, such an assignment must be nested within conditionals that test queries entailing $C.r_2 \sqsubseteq A.r$. The statements that will prove that relation at run-time will need to be trusted according to the integrity label of x. Suppose those statements are $A.r \leftarrow B.r_1$ and $B.r_1 \leftarrow C.r_2$. Now whatever other writers may be allowed to influence the addition of these statements, the owners of the roles being defined certainly must be. This shows that in this case the integrity label of x needs to be at most as restrictive as the integrity labels of $A.r \leftarrow B.r_1$ and $B.r_1 \leftarrow C.r_2$. In the rest of this section we present a recommended idiom, and RTI features that support it, which facilitates management of integrity labels.

$$\frac{Q \vdash P_1.r_1 \sqsubseteq P_2.r_2}{Q \vdash P_1'.r_1' \sqsubseteq P_2'.r_2'} \qquad \frac{}{Q \vdash P'.r' \sqsubseteq P}$$

Fig. 5. Additional Static Type Checking Rules for Idiom

In RT, authority to specify policy statements that define a role $A.r$ rests solely with the owner A. In RTI, authority to specify or modify a statement is controlled by the statement's integrity label. This authority interacts with the integrity labels of values that depend on these statements and on which the definition of these statements depend. To align the RTI integrity labels of policy statement precisely with the nature of authority over policy statements in RT, the integrity label of each policy statement $A.r \leftarrow e$ would be just the principal A. Indeed, in the idiom presented here, the statement $A.r \leftarrow e$ is given integrity label A. It is straightforward to generalize this to allow A to define a role, the members of which are delegated authority over the definition of $A.r$. In practice, this is likely to be important so as to enable policy updates to depend on data from other sources. However, for reasons of pedagogy and space, we refrain from introducing this generalization here.

EXAMPLE 9 (POLICY STATEMENT INTEGRITY). *Principals A, B, and D add the following statements to the policy* Π*:*

$$\langle A.r \leftarrow B.r_1, (B.r_1, A)\rangle \tag{1}$$
$$\langle B.r_1 \leftarrow C.r_2, (C.r_2, B)\rangle \tag{2}$$
$$\langle D.r_3 \leftarrow E.r_4, (E.r_4, D)\rangle \tag{3}$$

Suppose that variables x and y have labels $(E.r_4, A.r \sqcap IL_x)$ and $(D.r_3, C.r_2 \sqcap IL_y)$, respectively. Below we will consider the appropriate values for IL_x and IL_y. Consider the following code fragment: `if`$(\langle D.r_3 \sqsubseteq E.r_4, (E.r_4, A.r \sqcap IL_x)\rangle)$ `if`$(\langle A.r \sqsubseteq C.r_2, (E.r_4, A.r \sqcap IL_x)\rangle)$ $x := y$

In this example, there are two integrity requirements for the assignment to x to be legal. The first of these is

$$\Pi\downarrow_{(E.r_4, A.r \sqcap IL_x)} \vdash A.r \sqcap IL_x \sqsubseteq C.r_2 \sqcap IL_y \tag{4}$$

which means that anyone authorized to define the value of y is also authorized to define the value of x. The second integrity requirement is that the policy statements used to satisfy the two queries in the containing conditionals also must have integrity that is at least as restrictive as that of x. In other words, we require

$$\begin{aligned} \Pi\downharpoonright_{(E.r_4, A.r \sqcap IL_x)} &\vdash A.r \sqcap IL_x \sqsubseteq A \wedge \\ \Pi\downharpoonright_{(E.r_4, A.r \sqcap IL_x)} &\vdash A.r \sqcap IL_x \sqsubseteq B \wedge \\ \Pi\downharpoonright_{(E.r_4, A.r \sqcap IL_x)} &\vdash A.r \sqcap IL_x \sqsubseteq D \end{aligned} \tag{5}$$

According to trust-management principles, the use of a role such as $A.r$ in the label of a variable x is appropriate only if A is trusted to determine an appropriate membership for $A.r$, which includes trusting that when A delegates to some other principal B some authority over the definition of $A.r$, B too is trusted for this purpose. Requirements (4) and (5) illustrate the constraints that must be satisfied by the labels IL_x and IL_y if the assignment is to be legal. To facilitate defining the components of integrity labels like IL_x and IL_y, RTI implicitly defines and maintains a shadow role, $A'.r'$, for each role $A.r$ as a side-effect of adding a new policy statement. The shadow role $A'.r'$ is defined to include all principals that have authority to define statements that add principals to $A.r$.

Suppose that for each of statements (1), (2), and (3), we were to add two statements to the dynamic policy:

$$\begin{array}{ll} (1) & A'.r' \leftarrow A, \quad A'.r' \leftarrow B'.r'_1 \\ (2) & B'.r'_1 \leftarrow B, \quad B'.r'_1 \leftarrow C'.r'_2 \\ (3) & D'.r'_3 \leftarrow D, \quad D'.r'_3 \leftarrow E'.r'_4 \end{array}$$

We will not actually add these statements; instead we will extend the definition of what it means for a static or dynamic policy to satisfy queries that involve shadow roles. Now in our example, the minimum values that satisfy these constraints are $IL_x = A'.r' \sqcap D'.r'_3$ and $IL_y = C'.r'_2 \sqcap D'.r'_3$. In general, the integrity label of any variable x should be the meet of the following: (1) any trusted writers of the value of x; (2) $P'.r'$ for each (non-shadow) role $P.r$ appearing in the integrity label of x; plus, optionally, (3) $P'.r'$ for each (non-shadow) role $P.r$ appearing in the confidentiality label of any variable y or policy statement upon which x's value should be allowed to depend and the confidentiality label of which is not identical to that of x. This is why the shadow role $D'.r'_3$ is included in the label of x in the example. Component (3) can be omitted, if the owner of x prefers not to trust the owner of the confidentiality label of y. In this case the assignment $x := y$ is permitted only if $\emptyset \vdash |lab(\Gamma(y))|_C \sqsubseteq |lab(\Gamma(x))|_C$. By a similar analysis, it turns out that the integrity labels of policy statements should be composed in the manner.

To support this idiom, we add the type rules shown in figure 5 and we extend the definition of $\pi \vdash \kappa_1 \sqsubseteq \kappa_2$ as follows:

$$\begin{aligned} &\pi \vdash P'.r' \sqsubseteq P \text{ holds for all } \pi, P, \text{ and } r \\ &\pi \vdash P'_1.r'_1 \sqsubseteq P'_2.r'_2 \text{ if } \pi \vdash P_1.r_1 \sqsubseteq P_2.r_2 \end{aligned}$$

6. AN EXAMPLE

Let us look at the example program in figure 6 and illustrate how the program is type-checked and evaluated in RTI.

Specifying Security Labels. All integrity labels in the example adhere to the idiom introduced in the previous section. The confidentiality labels on the policy statements

1. $\texttt{if}(PatRequestsHIVtherapy)\{$
2. $\quad \texttt{try}_{\{\langle Pat.HIVR \sqsubseteq Hos.HIVDocR, b_1\rangle\}}\{$
3. $\quad\ \texttt{if}(\langle Pat.HIVR \sqsubseteq Hos.HIVDocR, b_1\rangle)$
4. $\quad\quad \texttt{update}(\texttt{add}(\langle Pat.RecsR \leftarrow Hos.HIVDocR, \ell_1\rangle));\}$
5. $\quad \texttt{try}_{\{\langle Pat.HIVR \sqsubseteq Hos.HIVDocW, b_2\rangle\}}\{$
6. $\quad\ \texttt{if}(\langle Pat.HIVR \sqsubseteq Hos.HIVDocW, b_2\rangle)$
7. $\quad\quad \texttt{update}(\texttt{add}(\langle Pat.RecsW \leftarrow Hos.HIVDocW, \ell_2\rangle));\}\}$
8. $\texttt{if}(\langle Pat.RecsR \sqsubseteq DrBob, b_3\rangle)$
9. $\quad DrDisplay := patientRec;$
10. $\texttt{if}(\langle Pat.RecsW \sqsubseteq DrBob, b_4\rangle)$
11. $\quad patientRecHIVRx := DrInput;$

in which

$$\ell_0 = (Pat.HIVR, Pat)$$
$$\ell_1 = b_1 = (Hos.HIVDocR, Pat)$$
$$\ell_2 = b_2 = (Hos.HIVDocW, Pat)$$
$$\ell_3 = b_3 = (DrBob, Pat.RecsW \sqcap Pat'.RecsW' \sqcap Pat'.RecsR')$$
$$\ell_4 = b_4 = (Pat.RecsR, Pat.RecsW \sqcap Pat'.RecsW' \sqcap Pat'.RecsR')$$
$$\ell_5 = (Pat.RecsR, Pat.RecsW \sqcap Pat'.RecsW')$$
$$\ell_6 = (Pat.RecsR, DrBob \sqcap Pat'.RecsR')$$
$$PatRequestsHIVtherapy : bool_{\ell_0}$$
$$DrDisplay : Record_{\ell_3}$$
$$patientRec : Record_{\ell_5}$$
$$patientRecHIVRx : Record_{\ell_4}$$
$$DrInput : Record_{\ell_6}$$

Fig. 6. An example RTI program.

added in lines 4 and 6 have been selected to accommodate the high confidentiality of $PatRequestsHIVtherapy$. Specifically, when the queries in the conditionals on lines 3 and 5 are satisfied, ℓ_1 and ℓ_2 dominate ℓ_0, the label of $PatRequestsHIVtherapy$. In practice, the code fragment comprising lines 1 through 7 needs to be executed only once for a given patient, while lines 8 through 11 might be executed many times.

Static Typing. The type of the update statement in line 4 is obtained by using the (T-AddDel1) static typing rule. The premise of this rule requires that $\Omega.Q \vdash \mathrm{pc} \sqsubseteq_2 \ell_1$. The pc is set to ℓ_0 by the conditional in line 1. At line 4 the pc remains the same, since (T-IfQ) has no effect on the pc. The query conditional at line 3 ensures that the set $\Omega.Q = \{\langle Pat.HIVR \sqsubseteq Hos.HIVDocR, b_1\rangle\}$. Note that it is always the case that $Q \vdash Pat \sqsubseteq Pat$. These two relationships combine to prove $\Omega.Q \vdash \mathrm{pc} \sqsubseteq_2 \ell_1$. Therefore line 4 type-checks. The query conditional in line 3 can be typed because its label is the same as the label of statement 4. Lines 6 and 7 type-check similarly. The assignment on line 9 type-checks under the context $Q_3 = \{\langle Pat.RecsR \sqsubseteq DrBob, b_3\rangle\}$ because $Q_3 \vdash \ell_5 \sqsubseteq_2 \ell_3$ and similarly the assignment on line 11 type-checks under the context $Q_4 = \{\langle Pat.RecsW \sqsubseteq DrBob, b_4\rangle\}$ (the integrity part being the more interesting:

$Q_4 \vdash Pat.RecsW \sqcap Pat'.RecsW' \sqcap Pat'.RecsR' \sqsubseteq DrBob \sqcap Pat'.RecsR')$; $\mathrm{pc} = \bot$ in both contexts. This ensures that the query conditionals on line 8 and 10 type-check.

Execution. Before the execution of statement 1, let the dynamic policy, Π, be

$$\{\langle Hos.HIVDocR \leftarrow DrBob, \ell_7\rangle, \tag{6}$$

$$\langle Hos.HIVDocW \leftarrow DrBob, \ell_7\rangle, \tag{7}$$

$$\langle Pat.HIVR \leftarrow Hos.HIVDocR, \ell_1\rangle, \tag{8}$$

$$\langle Pat.HIVR \leftarrow Hos.HIVDocW, \ell_2\rangle\} \tag{9}$$

in which $\ell_7 = (DrBob, Hos)$. Policy statements (6) and (7) indicate that $Dr.Bob$ is an HIV specialist. Through policy statements (8) and (9), the patient indicates that HIV doctors that are allowed to read and write HIV-related patient records are authorized to know whether the patient has requested HIV therapy. If $PatRequestsHIVtherapy$ is true, program statements 4 and 7 add two new policy statements to the dynamic policy, which allow the hospital HIV specialists to view and change the patient's health record:

$$\langle Pat.RecsR \leftarrow Hos.HIVDocR, \ell_1\rangle \tag{10}$$

$$\langle Pat.RecsW \leftarrow Hos.HIVDocW, \ell_2\rangle \tag{11}$$

Once these policy statements are added, the dynamic policy satisfies the conditionals in statements 8 and 10. The query on line 8 is satisfied by policy statements (6) and (10); these policy statements can both be used to evaluate the query because $\Pi \lfloor_{b_3} \vdash \ell_7 \sqsubseteq_2 b_3$ and $\Pi \lfloor_{b_3} \vdash \ell_1 \sqsubseteq_2 b_3$. The fact that $\Pi \lfloor_{b_3} \vdash |b_3|_I \sqsubseteq |\ell_7|_I$ follows because $\pi \vdash Hos'.HIVDocR' \sqsubseteq Hos$ holds for all π, and $\Pi \lfloor_{b_3} \vdash Pat'.RecsR' \sqsubseteq Hos'.HIVDocR'$ follows from $\Pi \lfloor_{b_3} \vdash Pat.RecsR \sqsubseteq Hos.HIVDocR$, which holds because policy statement (10) is in $\Pi \lfloor_{b_3}$. (This last point follows from $\Pi \lfloor_{b_3} \vdash \ell_1 \sqsubseteq_2 b_3$.) Note that determining that $\ell_1 \sqsubseteq_2 b_3$ makes use of policy statement (6). The query on line 10 is satisfied similarly by using policy statements (7) and (11). Therefore $Dr.Bob$ can view the patient's records and also give a diagnosis and prescription to the patient.

Considering another case where $Dr.Bob$ is not in fact an HIV specialist. $Hos.HIVDocR \leftarrow DrBob$ and $Hos.HIVDocW \leftarrow DrBob$ are not included in Π. The variable $DrInput$ is still labeled $(Pat.RecsR, DrBob)$ and the program type-checks statically, but the execution semantics will not execute the assignments in lines 9 and 11 because Π will not prove the query conditionals on lines 8 and 10.

In a third case, if the variable $PatRequestsHIVtherapy$ is not true, or one of the query conditionals on line 3 or 6 are not satisfied by the dynamic policy Π, the update statements on lines 4 and 7 are not executed, and the corresponding policy statements are not added. Therefore, the conditionals in lines 8 and 10 will not be true. This again ensures that the sensitive patient's record is not leaked to any unauthorized person.

7. NONINTERFERENCE

In this section, we present the noninterference theorem for RTI. The full proof of the theorem can be found in the appendix. The strategy of the proof follows that of Swamy et al. [Swamy et al. 2006], which in turn uses the proof technique of Pottier and Simonet [Pottier and Simonet 2003]. Much of the material in this section follows Swamy et al. quite closely.

An *execution* of a program S_0 starting in the configuration $\mathcal{E}_0$ is denoted by $\langle \mathcal{E}_0, S_0\rangle$ and is a (possibly infinite) sequence of configurations $\mathcal{E}_0, \mathcal{E}_1, \ldots$ and programs $S_0, S_1, \ldots$ such that each step of execution is performed according to the operational semantics; $\mathcal{E}_i, S_i \longrightarrow$

Policy:
$\Pi|_R = \Pi \downharpoonright_{\bigsqcup R}$
Memory:
$M|_{R,\Pi} = \{(x, M(x)) \mid \exists \rho \in R \,.\, \Pi \downharpoonright_{\bigsqcup R} \vdash lab(\Gamma(x)) \sqsubseteq \rho\}$
Dynamic snapshot:
$\cdot|_{R,\Pi} = \cdot \quad (S'')|_{R,\Pi} = (S'')$
Configuration:
$(\Pi, M, \Psi)|_R = (\Pi_R, M|_{R,\Pi}, \cdot)$
Trace:
$$(\mathcal{E}_1, \mathcal{E}_2, \alpha)|_R = \begin{cases} \mathcal{E}_1|_R \text{ if } \mathit{declass}(R, \mathcal{E}_1.\Pi, \mathcal{E}_2.\Pi) \neq \emptyset \\ \mathcal{E}_1|_R, (\mathcal{E}_2, \alpha)|_R \text{ otherwise} \end{cases}$$
in which
$\mathit{declass}(R, \Pi_1, \Pi_2) = (\rho_L(R, \Pi_2) \cap \rho_H(R, \Pi_1))$
$\rho_L(R, \Pi) = \{\rho \mid \exists \rho_R \in R.\Pi \downharpoonright_{\bigsqcup R} \vdash \rho \sqsubseteq \rho_R\}$
$\rho_H(R, \Pi) = \{\rho \mid \forall \rho_R \in R.\Pi \downharpoonright_{\bigsqcup R} \nvdash \rho \sqsubseteq \rho_R\}$

Fig. 7. Trace observability

$\mathcal{E}_{i+1}, S_{i+1}$. The sequence of configurations $\mathcal{E}_0, \mathcal{E}_1, \ldots$ obtained in this way is called a *trace* and is written $\mathrm{Tr}(\langle \mathcal{E}_0, S_0 \rangle)$. We use α to denote a possibly empty trace and $\mathcal{E}, \alpha$ to denote the concatenation of a single configuration and a trace.

The attacker's observation level is presumed to be given by a set of roles R. We assume a type environment Γ is given. The restriction of a trace α to observation level R is denoted by $\alpha|_R$, and is defined in figure 7. As long as the policy remains unchanged, a restricted trace consists of a restriction to each configuration element of the trace. (This is the "otherwise" case of the **Trace** definition in the figure. In this case, the first configuration is restricted to R and then the definition recurs.) Restricting the individual configurations restricts the view of memory (according to Π and Γ), the policy, and the snapshot. Note that $lab(\Gamma(x))$ refers to the label associated with the content of variable x. The policy is restricted by removing the policy statements that are not observable to R.

If a policy update results in *declassification* with respect to the attacker's roles R, then the trace is truncated, as specified in the first case of the **Trace** definition in the figure. The set of roles given by $declas(R, \mathcal{E}_1.\Pi, \mathcal{E}_2.\Pi)$ is non-empty when the execution step from $\mathcal{E}_1$ to $\mathcal{E}_2$ changes the policy in a way that makes some role change from being unobservable to being observable by the attacker. This is exactly the case in which a declassification has occurred that is observable to R. Note that the definitions of $\rho_L(R, \Pi)$ and $\rho_H(R, \Pi)$ use only those statements that are observable to the attacker given by R to determine which roles are observable to the attacker. Below we often write simply ρ_L and ρ_H, with the parameters R and Π being implicit.

The way that trace observability is defined, computation steps that change unobservable portions of the configuration appear to make no change whatsoever. The presence of repeated, unchanged configurations is called stuttering. We write $\alpha \doteq \beta$ if α and β are two different trace sequences, but are identical when unchanged configurations are eliminated.

The statement of noninterference uses the notion of well-formed configuration, written $\Omega \models \mathcal{E}$, which says that the execution configuration is consistent with the static context used to type-check the program.

DEFINITION 10 WELL-FORMED CONFIGURATION. *A configuration* $\mathcal{E} = (\Pi, M, \Psi)$ *is well-formed with respect to a context* Ω*, denoted by* $\Omega \models \mathcal{E}$*, if and only if all the following are true:*

$$(1)\ \mathit{dom}(M) \subseteq \mathit{dom}(\Omega.\Gamma)$$
$$(2)\ \forall \langle q, b\rangle \in \Omega.Q.\Pi\lfloor_b \vdash q$$

Clause (1) ensures that all the memory locations are assigned a type. Clause (2) checks that the static approximation, Q, is consistent with the dynamic policy Π. Note that clause (2) takes account of the fact that queries in Q have associated labels.

LEMMA 11 (STATIC LABEL ORDERING SOUNDNESS). *For all contexts* Ω *and programs* S*, if the derivation of* $\Omega \vdash S$ *contains a sub-derivation of* $\Omega' \vdash S'$*, then the following holds for all policies* Π*:*

$$(\forall \langle q, b\rangle \in \Omega'.Q\,.\,(\Pi\lfloor_b \vdash q)) \Rightarrow$$
$$(\forall \ell_1, \ell_2.\Omega'.Q \vdash \ell_1 \sqsubseteq_2 \ell_2 \Rightarrow \Pi\lfloor_\top \vdash \ell_1 \sqsubseteq_2 \ell_2)$$

Noninterference is proved by relating execution traces of well-formed configurations. The idea is to show that for any attacker, if two such configurations agree on the portions of them that the attacker can observe, then the execution traces restricted to the attacker's observation level will be equivalent up to stuttering.

THEOREM 12 (NONINTERFERENCE). *Suppose that for an RTI program* S *and a pair of configurations* $\mathcal{E}_0$ *and* $\mathcal{E}_1$ *there exists a context* Ω *such that* $\Omega \vdash S$*,* $\Omega \models \mathcal{E}_0$*, and* $\Omega \models \mathcal{E}_1$*. Then for any set of roles* R*, whenever both* $\langle \mathcal{E}_0, S\rangle$ *and* $\langle \mathcal{E}_1, S\rangle$ *terminate, we have*

$$\mathcal{E}_0|_R^\psi = \mathcal{E}_1|_R^\psi \Rightarrow \mathit{Tr}(\langle \mathcal{E}_0, S\rangle)|_R \doteq \mathit{Tr}(\langle \mathcal{E}_1, S\rangle)|_R$$

Intuitively, the proof shows that when executing a type-correct RTI program S, the effects observable to a low-security observer are independent of the high-security parts of the memory and policy with which the program executes. The proof is a straightforward inductive application of the Subject Reduction Theorem, which is proved in the appendix. Because policy updates change the definitions of high and low security, the noninterference result guarantees only that this independence holds at each execution step with respect to the policy in effect at that step. As in [Swamy et al. 2006], our noninterference result focuses on confidentiality and observability. Because our formulation of data integrity is essentially identical, the same argument can be used to show that data and policy integrity is preserved. We do not consider timing or termination channels.

8. COMPARISON OF RX AND RTI

As we already stated earlier, RTI is closely related to another previous work in type-based information flow analysis, Rx. The syntax of RTI resembles that of Rx. Rx consists of RT[ ] roles as labels of variables, RT[ ] policy statements as run-time policy statements, policy update statements and policy queries that approximate the run-time policy. However, RTI differs from Rx with respect to the ordering relation and the metapolicy. We have already illustrated some of the differences in Sections 3 and 4.2. We discuss the rest of the differences in this section.

Metapolicy. In Rx roles, like variables, are also protected by a pair of security labels. These security labels constitute what Rx calls the metapolicy of a role. Rx also uses the

readers-oriented notion of integrity. We discuss some of the problems caused by these design choices in section3. RTI proposes the writer-oriented view of integrity and protecting policy statements instead of individual roles as solutions to these problems. Thus Rx's readers-oriented integrity model follows the trust model and RTI follows the writer model and DLM as described by Li et al. [Li et al. 2003].

Rx uses the notation $C_\Pi(\rho)$ to denote the confidentiality label of ρ. Because Rx uses role-level protection, Rx requires $[\![\rho]\!]_{SP(\Pi)} \subseteq [\![C_\Pi(\rho)]\!]_{SP(\Pi)}$.

In Rx, there is another strong, problematic constraint on the meta-label of a role. If the membership of role ρ depends on the membership of role ρ' (e.g., because there is a policy statement $\rho \leftarrow \rho'$), then the metapolicy of ρ must be at least as restrictive as the metapolicy for ρ', i.e., $[\![C_\Pi(\rho)]\!]_{SP(\Pi)} \subseteq [\![C_\Pi(\rho')]\!]_{SP(\Pi)}$. This is because members of ρ can observe the effect of changes to the membership of ρ' via their effect on ρ. In general, ρ depends on ρ' if it delegates transitively to ρ'. An unfortunate consequence is that if a new statement such as $A.r \leftarrow E$ is added to the policy in the previous paragraph, the principal E should be added not only to $[\![C_\Pi(A.r)]\!]_{SP(\Pi)}$, but also to $[\![C_\Pi(B.r)]\!]_{SP(\Pi)}$, thus changing the denotation of $C_\Pi(B.r)$ each time a new principal is added to $A.r$. More importantly, any principal A can enable himself to view all the members of $B.r$ merely by adding a statement such as $A.r \leftarrow B.r$. For example, a medical supplier could guess what disease Alice has by defining a new role MedSup.snoop, including himself in that role, adding the rule $MedSup.snoop \leftarrow Alice.record$, and then observing that external specialist Bob can read Alice's medical record.

Swamy et al. suggest that one might mitigate this problem by using the integrity label $C_\Pi(B.r)$ to prevent unauthorized delegation to $B.r$. The idea is to require that all the data tested in conditionals leading to an update that adds a statement of the form $A.r \leftarrow B.r$ should be trusted by everyone that trusts the value of $[\![B.r]\!]_{SP(\Pi)}$. This could be accomplished by using the component of that static context known as the program counter (pc), which summarizes the security levels of data tested in conditionals leading to a given point in the program.

This, however, seems unintuitive. The integrity label of $B.r$ now has a major role in protecting the confidentiality of $B.r$. One expects the integrity label of an object to control how its value can be defined, not how its value can be used. Furthermore, members of roles must all agree before their role can be included in another role. In a highly decentralized environment, this does not seem conducive to scalability. Role owners should have autonomy to define their roles however they wish. Overall, this requires that the action of modifying the definition of $C_\Pi(\rho')$ should be decoupled from the act of introducing a new statement $\rho \leftarrow \rho'$.

By contrast, RTI leverages the implicit assumptions used in trust management policy. When A creates $A.r \leftarrow B.r$, this means that A trusts B to correctly define the membership of $B.r$. Therefore, the principals who trust the membership of $A.r$ implicitly trust A, B, and all the other principals whose roles the membership of $A.r$ depends on. On the other hand, since in RTI, membership in $[\![A.r]\!]_{SP(\Pi)}$ does not imply membership in $[\![C_\Pi(B.r)]\!]_{SP(\Pi)}$, there is no need for the members of $[\![B.r]\!]_{SP(\Pi)}$ to trust the pc (at the point the update is performed). This will make policy management more scalable.

Transactions. Rx uses a transaction construct for two purposes. First, it ensures that in the operational semantics, when a policy update is performed, if it causes a violation of the assumptions under which the currently executing code region was type-checked, control

breaks out of that region. This is accomplished in RTI by the try construct. Second, transactions attempt to address the problem of illegal transitive flows, in which data flows from x to y under one policy and then from y to z after a policy modification, yet neither policy authorized the flow from x to z. While transactions can enable a programmer to avoid such illegal flows by placing all flows within a single transaction, Rx does not strictly enforce this and it is often not practical to enforce it either(different flows might be in different methods of the program). In RTI we have omitted transactions in the interest of simplicity and because they do not provide a comprehensive solution to the problem they are designed to address.

9. RELATED WORK

Our related work, as discussed below, occurs in two different areas: trust management and language based security.

9.1 Trust Management

Trust management was introduced by Blaze et al. [Blaze et al. 1996] as a problem in network security for which the authors proposed an approach based on a small collection of general principles, the most important for our context being decentralization of control. The first trust management system, called PolicyMaker, was also introduced, followed by its descendant, KeyNote [Blaze et al. 1999]. These systems were designed to support delegation of authorization to use specific resources. About the same time as PolicyMaker, the Simple Distributed Security Infrastructure (SDSI) [Rivest and Lampson 1996] was introduced. Rather than delegation of authorization, SDSI focused on naming. Because SDSI names can be used to represent groups or roles, there is a strong relationship between SDSI and RT, though full RT is much richer. (RTI barely scratches the surface of RT's expressive power.) More recently, many other trust management systems have been designed and analyzed (e.g., [Gunter and Jim 2000; Appel and Felten 1999; Jim and Suciu 2001] to identify just a few). We use RT because of its usage of roles, its support for highly decentralized policy definition, and its simple declarative semantics.

9.2 Language Based Security

In language based information flow analysis techniques [Sabelfeld and Myers 2003], security labels form a lattice according to the lattice model of security proposed by Denning [Denning 1976]. These approaches statically check whether the program satisfies the noninterference property [Goguen and Meseguer 1982] with respect to the security lattice. Initial security lattices were simple with only two security levels, high and low [Volpano et al. 1996; Denning and Denning 1977]. A high or low security label was associated with the variables based on ownership and access policies. Myers et al. [Myers and Liskov 2000] incorporated the policies based on ownerships and flows directly into the security lattice in their Decentralized Label Model (DLM). The security label consists of confidentiality label and integrity label pair. The confidentiality label consists of owners of the variable and the readers who could read the information in the variable. The integrity label consists of owners and writers who could write to the variable. The security label in RTI also follows the same confidentiality and integrity model for the security labels.

Most of the work in language-based information flow [Sabelfeld and Myers 2003; Broberg and Sands 2006] assumes that the security lattice is known at compile-time and remains fixed during program execution. Noninterference was proved with respect to the

static lattice. Realistic code, however, needs to execute in different environments and the security lattice might change based on the environment. Dynamic principals and labels [Myers et al. ; Chong and Myers 2004; Tse and Zdancewic 2004] that could be instantiated at run-time were proposed to allow dynamic querying of the security lattice. To type-check the programs statically, the languages provided constructs similar to our policy query. These approaches, however, did not provide a way to change the security lattice at run-time.

In contrast, similar to previous work by Hicks et al. [Hicks et al. 2005] and Swamy et al. [Swamy et al. 2006], we provide an update statement to dynamically update the security lattice. We also provide fine-grained protection to the security lattice by protecting the dynamic policy statements with the help of a security label. The net effect is that different principals observe different lattices based on their observation level. Although Rx also protects the lattice, their protection is more coarse-grained because any principal that is allowed to access a particular role, is permitted by the metapolicy of Rx to observe the entire sub-lattice dominated by the role.

Many previous approaches propose some form of declassification construct based on various constraints (see [Sabelfeld and Sands 2007] for a survey). This construct allows temporary change in the security lattice order, such that if a particular value is sufficiently general, according to some constraints, it can be released on a low security output channel. For instance, this might be done if the value has been encrypted. However, one might not want to release other values that have the same security label. Our approach, supports a related, but rather different operation: update statements permanently change the security lattice. If an update results in a value becoming less strictly protected, all the values that are protected by the roles involved are affected. In order to provide declassification in the classical sense, we can extend our approach with a declassification construct that changes the lattice temporarily.

Another useful construct could be providing polymorphic labels and label inferencing [Myers et al. ; Simonet 2003]. We believe that our approach can be extended to allow such labels. For instance, in our example in Section 6, the piece of code that is executed by Dr. Bob is common to all the doctors, and if we were to allow labels that could be instantiated at run-time, the code could be reused.

Almeida Matos et al. [Matos and Boudol 2005] propose a flow construct that can dynamically introduce flows in a portion of the program. The flow, however, is local to the block of code for which it has been defined. One difference is that the programmer can arbitrarily introduce the flows irrespective of the context, whereas in our approach only authorized principals can add the flows. Moreover, the new flows could be visible only to authorized principals. Almeida Matos et al. prove the *non-disclosure policy*, which states that at each step of execution, the program satisfies the noninterference property with respect to the flow policy that holds for that step. In the absence of declassification, our approach proves the same property. However, when there are declassifying updates, the program terminates and no guarantees are given regarding the noninterference. In this case the noninterference theorem proves *noninterference until conditions*, proposed by Chong et al. [Chong and Myers 2004].

10. CONCLUSION

We have identified shortcomings in the recently proposed security-typed language Rx [Swamy et al. 2006]. To address these problems, we have proposed a new security-typed language, RTI, given static type-checking rules and dynamic execution semantics for RTI, and proved noninterference until declassification for RTI.

In comparison to Rx, RTI more fully embraces the trust-management philosophy towards authority and delegation. RTI adopts a writers-oriented approach for integrity policies. This gives role owners the autonomy they need to manage their role definitions without outside interference. RTI protects individual policy statements that define aspects of a role, rather than protecting an entire role as a single unit. This means that the membership of the roles can be kept more private. For example, in RTI a member of a role is not necessarily entitled to learn all the other members of that role, or to see all the statements used to define that role. Finally, we also proposed an idiom for security labels that will assist programmers in defining consistent labellings for policy statements and variables.

ACKNOWLEDGMENTS

Thanks to the anonymous referees for their numerous helpful comments and suggestions.

REFERENCES

APPEL, A. W. AND FELTEN, E. W. 1999. Proof-carrying authentication. In *CCS '99: Proceedings of the 6th ACM conference on Computer and communications security*. ACM Press, New York, NY, USA, 52–62.

BLAZE, M., FEIGENBAUM, J., IOANNIDIS, J., AND KEROMYTIS, A. D. 1999. The KeyNote trust-management system, version 2. IETF RFC 2704.

BLAZE, M., FEIGENBAUM, J., AND LACY, J. 1996. Decentralized trust management. In *Proceedings of the 1996 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, 164–173.

BROBERG, N. AND SANDS, D. 2006. Flow locks: Towards a core calculus for dynamic flow policies. In *ESOP*. 180–196.

CHONG, S. AND MYERS, A. C. 2004. Security policies for downgrading. In *ACM Conference on Computer and Communications Security*. 198–209.

DENNING, D. E. 1976. A lattice model of secure information flow. *Commun. ACM 19,* 5, 236–243.

DENNING, D. E. AND DENNING, P. J. 1977. Certification of programs for secure information flow. *Commun. ACM 20,* 7, 504–513.

GOGUEN, J. A. AND MESEGUER, J. 1982. Security policies and security models. In *IEEE Symposium on Security and Privacy*. 11–20.

GUNTER, C. A. AND JIM, T. 2000. Policy-directed certificate retrieval. *Software: Practice & Experience 30,* 15 (Sept.), 1609–1640.

HICKS, M., TSE, S., HICKS, B., AND ZDANCEWIC, S. 2005. Dynamic updating of information-flow policies. In *FCS*.

JIM, T. AND SUCIU, D. 2001. Dynamically distributed query evaluation. In *PODS '01: Proceedings of the twentieth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*. ACM Press, New York, NY, USA, 28–39.

LI, N. AND MITCHELL, J. C. 2003. RT: A role-based trust-management framework. In *The Third DARPA Information Survivability Conference and Exposition (DISCEX III)*. IEEE Computer Society Press.

LI, N., MITCHELL, J. C., AND WINSBOROUGH, W. H. 2002. Design of a role-based trust-management framework. In *IEEE Symposium on Security and Privacy*. 114–130.

LI, N., MITCHELL, J. C., AND WINSBOROUGH, W. H. 2005. Beyond proof-of-compliance: security analysis in trust management. *J. ACM 52,* 3, 474–514.

LI, P., MAO, Y., AND ZDANCEWIC, S. 2003. Information integrity policies. In *FAST*.

LLOYD, J. W. 1993. *Foundations of Logic Programming*. Springer-Verlag New York, Inc., Secaucus, NJ, USA.

MATOS, A. A. AND BOUDOL, G. 2005. On declassification and the non-disclosure policy. In *CSFW*. 226–240.

MYERS, A. C., CHONG, S., NYSTROM, N., ZHENG, L., AND ZDANCEWIC, S. Jif: Java information flow. Located at http://www.cs.cornell.edu/jif.

MYERS, A. C. AND LISKOV, B. 2000. Protecting privacy using the decentralized label model. *ACM Trans. Softw. Eng. Methodol. 9,* 4, 410–442.

POTTIER, F. AND SIMONET, V. 2003. Information flow inference for ml. *ACM Transactions on Programming Languages and Systems 25,* 1, 117–158.

RIVEST, R. L. AND LAMPSON, B. 1996. SDSI — a simple distributed security infrastructure. Available at *http://theory.lcs.mit.edu/∼rivest/sdsi11.html*.

SABELFELD, A. AND MYERS, A. 2003. Language-based information-flow security. In *IEEE JSAC*.

SABELFELD, A. AND SANDS, D. 2007. Declassification: Dimensions and principles. *Journal of Computer Security*.

SANDHU, R. S., COYNE, E. J., FEINSTEIN, H. L., AND YOUMAN, C. E. 1996. Role-based access control models. *IEEE Computer 29,* 2 (February), 38–47.

SIMONET, V. 2003. The flow caml system (version 1.00): Documentation and user's manual. Tech. Rep. 0282, INRIA.

SWAMY, N., HICKS, M., TSE, S., AND ZDANCEWIC, S. 2006. Managing policy updates in security-typed languages. In *CSFW*. 202–216.

TSE, S. AND ZDANCEWIC, S. 2004. Run-time principals in information-flow type systems. In *IEEE Symposium on Security and Privacy*. 179–193.

VOLPANO, D. M., IRVINE, C. E., AND SMITH, G. 1996. A sound type system for secure flow analysis. In *Journal of Computer Security*. Vol. 4. 167–188.

ZHENG, L. AND MYERS, A. C. 2004. Dynamic security labels and noninterference. In *Formal Aspects in Security and Trust*. 27–40.

Appendix

Following [Swamy et al. 2006] we use the proof technique due to Pottier and Simonet [Pottier and Simonet 2003], in which a pair of executions of an RTI program is represented within the syntax, given in figure 8, of an slightly extended form of the language itself, called RTI^2. Section A of the appendix introduces the RTI^2 language, syntax and semantics and we prove the noninterference property for confidentiality in section B of the appendix. Therefore, in the rest of the appendix, we only consider the confidentiality part of the labels, ℓ, bounds, b, and pc.

A. RTI^2

A.1 RTI^2 Syntax

The first slight extension is that the try statements are labeled not only with a set of invariant constraints Q, but also with a label pc that represents the pc label used to check the try.

RTI expressions	E	::=	$\texttt{true} \mid \texttt{false} \mid x \mid v \mid E \oplus E$
RTI^2 expressions	E	::=	$E \mid \mathsf{v} \mid \mathsf{E} \oplus \mathsf{E} \mid \langle E \| \| E \rangle$
RTI statements	S	::=	$\texttt{skip} \mid S; S \mid \texttt{try}_{(Q,\mathrm{pc})} S \mid x := E$
			$\mid \texttt{while}\,(E)\,S \mid \texttt{if}\,(E)\,S\,S$
			$\mid \texttt{if}\,(\langle q, b\rangle)\,S\,S \mid \texttt{update}\ \Delta$
RTI^2 statements	S	::=	$\texttt{skip} \mid \ll S \| S \gg \mid \mathsf{S}; S$
			$\mid \texttt{if}\,(\mathsf{E})\,S\,S \mid x := \mathsf{E} \mid \texttt{try}_{(Q,\mathrm{pc})} \mathsf{S}$
			$\mid \texttt{if}\,(\langle q, b\rangle)\,S\,S \mid \texttt{update}\ \Delta$

Fig. 8. Syntax of RTI^2.

The syntactic elements that belong to RTI^2 are denoted in the sans serif font (E, S), while those in RTI are denoted by their italic counterparts (E, S). The expression E in the RTI^2 syntax can be a pair $\langle E_1 || E_2 \rangle$, and the statement S can be the pair $\langle S_1 || S_2 \rangle$. E_1 and S_1 represent expression and statement in the first execution, likewise E_2 and S_2 represent expression and statement in the second execution.

A.2 Static Semantics for RTI^2

$$\Omega \vdash v : t_{CL} \quad \text{(T-LIT)} \qquad \frac{\Omega.\Gamma(x) = t_{CL}}{\Omega \vdash x : t_{CL}} \quad \text{(T-VAR)} \qquad \frac{\Omega \vdash \mathsf{E}_1 : t_{CL_1} \quad \Omega \vdash \mathsf{E}_2 : t_{CL_2}}{\Omega \vdash \mathsf{E}_1 \oplus \mathsf{E}_2 : t_{CL_1 \sqcup CL_2}} \quad \text{(T-PLUS)}$$

$$\frac{\begin{array}{c}\rho \in \rho_H \\ \Omega \vdash E_1 : t_{CL} \quad \Omega \vdash E_2 : t_{CL}\end{array}}{\Omega \vdash \ll E_1 \parallel E_2 \gg : t_{CL \sqcup \rho}} \quad \text{(T-EBR)} \qquad \frac{\begin{array}{c}\Omega \vdash x : t_{CL_1} \quad \Omega \vdash \mathsf{E} : t_{CL_2} \quad \Omega.\Phi \in \{\cdot, *\} \\ \Omega.Q \vdash CL_2 \sqsubseteq CL_1 \quad \Omega.Q \vdash \Omega.\mathrm{pc} \sqsubseteq CL_1\end{array}}{\Omega \vdash x := \mathsf{E} : CL_1 \ \texttt{cmd}} \quad \text{(T-Assign)}$$

$$\frac{\Omega.\Phi \in \{\cdot, *\} \quad \Omega \vdash \mathsf{S}_1 : CB_1 \ \texttt{cmd} \quad \Omega \vdash S_2 : CB_2 \ \texttt{cmd}}{\Omega \vdash \mathsf{S}_1; S_2 : (CB_1 \sqcap CB_2) \ \texttt{cmd}} \quad \text{(T-Seq)}$$

$$\frac{\begin{array}{c}\rho \in \rho_H \quad \mathrm{pc}' = \mathrm{pc} \sqcup \rho \\ \Gamma; \mathrm{pc}'; Q; \Phi \vdash S_1 : CB_1 \ \texttt{cmd} \quad \Gamma; \mathrm{pc}'; Q; \Phi \vdash S_2 : CB_2 \ \texttt{cmd}\end{array}}{\Gamma; \mathrm{pc}; Q; \Phi \vdash \ll S_1 || S_2 \gg : CB_1 \sqcap CB_2 \ \texttt{cmd}} \quad \text{(T-SBR)}$$

$$\frac{\Gamma; \mathrm{pc}; \emptyset; try \vdash \mathsf{S} : CB \ \texttt{cmd}}{\Gamma; \mathrm{pc}; \emptyset; \cdot \vdash \texttt{try}_{Q', \mathrm{pc}} \mathsf{S} : CB \ \texttt{cmd}} \quad \text{(T-Try)}$$

$$\frac{\begin{array}{c}\Phi = try \Rightarrow \Phi' = try \quad \Phi \in \{\cdot, *\} \Rightarrow \Phi' = * \\ \Gamma; \widehat{\mathrm{pc}}; Q \cup \{q\}; \Phi' \vdash S_1 : CB_1 \ \texttt{cmd} \quad q \in Q' \\ \Gamma; \widehat{\mathrm{pc}}; Q; \Phi' \vdash S_2 : CB_2 \ \texttt{cmd} \quad CB = CB_1 \sqcap CB_2 \\ \widehat{\mathrm{pc}} = \mathrm{pc} \sqcup (\sqcup \{\rho_1 \mid \emptyset \vdash \rho_1 \sqsubseteq CB\})\end{array}}{\Gamma; \mathrm{pc}; Q; \Phi \vdash \texttt{if}(\langle q, CB \rangle)\ S_1\ S_2\ :\ CB \ \texttt{cmd}} \quad \text{(T-IfQ)}$$

$$\frac{\Gamma; \mathrm{pc}; Q; try \vdash \Delta : \mathrm{pol}_{CB}}{\Gamma; \mathrm{pc}; Q; try \vdash \texttt{update}(\Delta) : CB \ \texttt{cmd}} \quad \text{(T-Up)}$$

$$\frac{\begin{array}{c}\Gamma; \mathrm{pc}; Q; \Phi \vdash E : t_{CL} \\ \Gamma; \mathrm{pc} \sqcup CL; Q; \Phi \vdash S_1 : CB_1 \ \texttt{cmd} \quad \Gamma; \mathrm{pc} \sqcup CL; Q; \Phi \vdash S_2 : CB_2 \ \texttt{cmd}\end{array}}{\Gamma; \mathrm{pc}; Q; \Phi \vdash \texttt{if}(E)\ S_1\ S_2 : CB_1 \sqcap CB_2 \ \texttt{cmd}} \quad \text{(T-IfE)}$$

$$\frac{\Omega.\Phi \in \{\cdot, *\} \quad \Gamma; \mathrm{pc}; Q; \Phi \vdash E : t_{CL} \quad \Gamma; \mathrm{pc} \sqcup CL; Q; \Phi \vdash S : CB \ \texttt{cmd}}{\Gamma; \mathrm{pc}; Q; \Phi \vdash \texttt{while}\,(E)\,S : CB \ \texttt{cmd}} \quad \text{(T-Whl)}$$

Fig. 9. Static semantics of RTI^2

RTI^2 typing judgments given in figure 9 are identical to the RTI typing judgments, but for the fact that they additionally handle bracketed expressions and statements. The bracketed

RTI^2 Configuration:

policy	Π	::=	$\ll \Pi_1 \| \| \Pi_2 \gg$
values	v	::=	$v \mid \ll v_1 \| \| v_2 \gg$
memory	M	::=	$\{(x_1, \mathsf{v}_1), \ldots, (x_n, \mathsf{v}_n)\}$
one snapshot	Ψ^*	::=	$.\mid(\mathsf{S})$
snapshot	Ψ	::=	$\Psi^* \mid \ll \Psi_1^* \| \| \Psi_2^* \gg$
configuration	$\mathcal{C}$	::=	$\Pi; \mathsf{M}; \Psi$

Values $\lfloor \mathsf{v} \rfloor_i$:
$\lfloor \mathsf{v} \rfloor_0 = \mathsf{v} \quad \lfloor v \rfloor_i = v \quad \lfloor \ll v_1 \| \| v_2 \gg \rfloor_i = v_i$
Policy $\lfloor \Pi \rfloor_i$:
$\lfloor \Pi \rfloor_0 = \lfloor \Pi \rfloor_1 \cap \lfloor \Pi \rfloor_2 \quad \lfloor \emptyset \rfloor_i = \emptyset$
$\lfloor \ll \Pi_1 \| \| \Pi_2 \gg \rfloor_1 = \Pi_1 \quad \lfloor \ll \Pi_1 \| \| \Pi_2 \gg \rfloor_2 = \Pi_2$
Memory $\lfloor \mathsf{M} \rfloor_i$:
$\lfloor \mathsf{M} \rfloor_0 = \mathsf{M} \quad \lfloor \emptyset \rfloor_i = \emptyset \quad \lfloor \{(x, v)\} \cup \mathsf{M} \} \rfloor_i = \{(x, v)\} \cup \lfloor \mathsf{M} \rfloor_i$
$\lfloor \{(x, \ll v_1 \| \| v_2 \gg)\} \cup \mathsf{M} \} \rfloor_i = \{(x, v_i)\} \cup \lfloor \mathsf{M} \rfloor_i$
Dynamic Snapshot $\lfloor \Psi \rfloor_i$:
$\lfloor \Psi \rfloor_0 = \Psi \quad \lfloor \Psi^* \rfloor_i = \Psi^* \quad \lfloor \ll \Psi_1^* \| \| \Psi_2^* \gg \rfloor_i = \Psi_i^*$
Configuration $\lfloor \mathcal{C} \rfloor_i$:
$\lfloor (\Pi, \mathsf{M}, \Psi) \rfloor_i = (\lfloor \Pi \rfloor_i, \lfloor \mathsf{M} \rfloor_i, \lfloor \Psi \rfloor_i)$

Fig. 10. RTI^2 configurations and projections

expressions and statements are type-checked under a high pc. The high pc is calculated using the high roles $\rho_H(R, \Pi)$ as given in the figure 7. This ensures that even if the execution diverges in the expressions and statements, it does not contain any flows to observable variables.

A.3 RTI^2 Configurations and Observability

Since RTI^2's syntax consists of pairs of executions, the memory, the policy, and the execution configuration must each accommodate the execution of the respective statements within the pairs. As shown in figure 10, the structure of the configuration, $\mathcal{C}$, incorporates these generalized forms of memory, policy, and try snapshot. The memory, M, consists of a single value for a variable x when both the executions agree on the value, or of a pair of values, one for each execution. One of the values can also be empty when the memory location has not been initialized in the corresponding execution. On the other hand, the policy, Π, is simply a pair of sets of labeled policy statements, $\ll \Pi_1 \| \| \Pi_2 \gg$. The snapshot, Ψ, is a pair of dynamic snapshots, each of which can optionally be empty($\cdot$).

Figure 10 also shows the result of projecting the pairs of values, policy, memory etc. onto a single execution. This projection operator is used in the operational semantics when the execution affects only one side of the execution.

Figure 11 extends the trace observability definitions in figure 7 to parallel executions.

We also extend the well-formedness definition to accommodate RTI^2 configurations as follows:

DEFINITION 13 WELL-FORMED CONFIGURATION $\mathcal{C}$. *A configuration* $\mathcal{C} = (\Pi, \mathsf{M}, \Psi)$ *is well-formed with respect to a context* Ω*, a set of roles* R*, and an* $i \in \{0, 1, 2\}$*,*

Policy $\Pi\,|_R$:
$\Pi\,|_R = \ll \lfloor \Pi \rfloor_1 \downharpoonright_{\bigsqcup R} || \lfloor \Pi \rfloor_2 \downharpoonright_{\bigsqcup R} \gg$
Memory $\mathsf{M}\,|_{R,\Pi}$:
$\mathsf{M}\,|_{R,\Pi} = \lfloor \mathsf{M} \rfloor_1\,|_{R,\lfloor \Pi \rfloor_1} \oplus \lfloor \mathsf{M} \rfloor_2\,|_{R,\lfloor \Pi \rfloor_2}$ in which

$$M_1 \oplus M_2 = \bigcup \begin{cases} (x, M_1(x)) & M_1(x) = M_2(x) \\ (x, \ll M_1(x) || M_2(x) \gg) & M_1(x) \neq M_2(x) \\ (x, \ll M_1(x) ||. \gg) & x \in dom(M_1) \setminus dom(M_2) \\ (x, \ll . || M_2(x) \gg) & x \in dom(M_2) \setminus dom(M_1) \end{cases}$$

Dynamic Snapshot $\Psi\,|_{R,\Pi}$:
$(\mathsf{S})\,|_{R,\Pi} = (\mathsf{S})$
$\ll \Psi_1^* || \Psi_2^* \gg|_{R,\Pi} = \ll \Psi_1^*\,|_{R,\Pi} || \Psi_2^*\,|_{R,\Pi} \gg$
Configuration $\mathcal{C}\,|_R$:
$(\Pi, \mathsf{M}, \Psi)\,|_R = (\Pi\,|_R, \mathsf{M}\,|_{R,\Pi}, .)$

Fig. 11. Observability of RTI^2 configurations

denoted by $\Omega \models_{R,i} \mathcal{C}$*, if and only if all the following are true:*

$$dom(\mathsf{M}) = dom(\Omega.\Gamma) \;\wedge\; \forall x.\Omega \vdash \mathsf{M}(x) : \Omega.\Gamma(x) \quad (1)$$
$$\forall \langle q, CB \rangle \in \Omega.Q.\lfloor \Pi \rfloor_i \downharpoonright_{CB} \vdash q \quad (2)$$
$$\lfloor \mathcal{C}|_R \rfloor_1 \equiv \lfloor \mathcal{C}|_R \rfloor_2 \quad (3)$$

Clause (1) has been strengthened to say not only that the memory and the type system have the same domain, but also that memory values are typed according (T-EBR) whenever they are pairs. This rule requires that the label of value pairs must be high with respect to R. As this is the only rule for typing value pairs, the effect is to prevent value pairs being introduced if the locations are not high-security. The final difference is the addition of clause (3), which requires that the pair of RTI^2 configurations are observationally equivalent with respect to the set of roles R belonging to the low observer.

A.4 Operational Semantics of RTI^2

The operational semantics of RTI^2 is given in figure 12. The execution rule is of the form $\mathsf{S}\,/_i\,\mathcal{C} \;\longrightarrow\; \mathsf{S}'\,/_i\,\mathcal{C}'$ The subscript i can be 0, 1, or 2 depending on whether the operation is common to both the executions or not. If it is 0, then both the executions take a step together and modify only low regions of the memory and policy. On the other hand, when the execution reaches an expression where the two configurations diverge, the configuration is split into two parallel configurations, and the execution rules are labeled 1 or 2. The lifting rules given in figure 13 give the semantics of the split.

A key element of the proof strategy is that the evaluation of statements and expressions is split only in high contexts, where the evaluation effects are not observable to the attacker. This enables us to show inductively that executions remain indistinguishable to the attacker, even when the two parts of the execution are different. In (L-IFQ) for instance, because the label of every variable or policy statement modified within S_1 or S_2 dominates CB, the premise ensures that none of those labels is low (with respect to R).

The premises of (L-IFQ) and (L-TR) bear discussion. No policy statements are needed to show that $\rho \sqsubseteq CB$ (in (L-IFQ)) and that $\rho \sqsubseteq \mathrm{pc}$ (in (L-TR)). This is because of how ρ_H is defined: $\rho_H(R, \Pi) = \{\rho \mid \forall \rho_R \in R.\Pi \downharpoonright_{\bigsqcup R} \not\vdash \rho \sqsubseteq \rho_R\}$. Considering only the (L-IFQ) case for the moment, when proving that some $\rho \in \rho_H$ satisfies $\rho \sqsubseteq CB$, there is no need to use policy statements to show that there is a ρ' such that $\rho \sqsubseteq \rho'$ and $\rho' \sqsubseteq CB$. (For

$$\frac{\lfloor \mathcal{C}.\mathsf{M}(x) \rfloor_i = \mathsf{v}}{x \,/_i\, \mathcal{C} \;\longrightarrow\; \mathsf{v} \,/_i\, \mathcal{C}'} \text{ (E-VAR)} \qquad \frac{\mathsf{E}_1 \,/_i\, \mathcal{C} \;\longrightarrow\; \mathsf{E}_1' \,/_i\, \mathcal{C}}{\mathsf{E}_1 \oplus \mathsf{E}_2 \,/_i\, \mathcal{C} \;\longrightarrow\; \mathsf{E}_1' \oplus \mathsf{E}_2 \,/_i\, \mathcal{C}} \text{ (E-ADL)}$$

$$\frac{\mathsf{E}_2 \,/_i\, \mathcal{C} \;\longrightarrow\; \mathsf{E}_2' \,/_i\, \mathcal{C}}{\mathsf{v} \oplus \mathsf{E}_2 \,/_i\, \mathcal{C} \;\longrightarrow\; \mathsf{v} \oplus \mathsf{E}_2' \,/_i\, \mathcal{C}} \text{ (E-ADR)} \qquad \frac{\mathsf{v} = \mathsf{v}_1 [\![\oplus]\!] \mathsf{v}_2}{\mathsf{v}_1 \oplus \mathsf{v}_2 \,/_i\, \mathcal{C} \;\longrightarrow\; \mathsf{v} \,/_i\, \mathcal{C}} \text{ (E-ADV)}$$

$$\frac{\mathsf{E} \,/_i\, \mathcal{C} \;\longrightarrow\; \mathsf{E}' \,/_i\, \mathcal{C}}{x \;:=\; \mathsf{E} \,/_i\, \mathcal{C} \;\longrightarrow\; x \;:=\; \mathsf{E}' \,/_i\, \mathcal{C}} \text{ (E-ASE)} \qquad \frac{\mathcal{C}' = \mathcal{C}[\mathsf{M} = \mathit{updloc}_i^2(x, \mathsf{v}, \mathcal{C}.\mathsf{M})]}{x \;:=\; \mathsf{v} \,/_i\, \mathcal{C} \;\longrightarrow\; \mathtt{skip} \,/_i\, \mathcal{C}'} \text{ (E-ASV)}$$

$$\mathtt{skip}; S \,/_i\, \mathcal{C} \;\longrightarrow\; S \,/_i\, \mathcal{C} \text{ (E-SKP)} \qquad \frac{E \,/_i\, \mathcal{C} \;\longrightarrow\; \mathsf{E} \,/_i\, \mathcal{C}}{\mathtt{if}\,(E)\; S_1\, S_2 \,/_i\, \mathcal{C} \;\longrightarrow\; \mathtt{if}\,(\mathsf{E})\; S_1\, S_2 \,/_i\, \mathcal{C}} \text{ (E-IFE)}$$

$$\frac{\begin{array}{c}(v \neq 0 \Rightarrow j = 1)\\ (v = 0 \Rightarrow j = 2)\end{array}}{\mathtt{if}\,(v)\; S_1\, S_2 \,/_i\, \mathcal{C} \;\longrightarrow\; S_j \,/_i\, \mathcal{C}} \text{ (E-IFV)} \qquad \begin{array}{c}\mathtt{while}\,(E)\, S \,/_i\, \mathcal{C} \;\longrightarrow\\ \mathtt{if}\,(E)\,\{S; \mathtt{while}\,(E)\, S\}\,\mathtt{skip} \,/_i\, \mathcal{C}\end{array} \text{ (E-WHL)}$$

$$\frac{\mathsf{S} \,/_i\, \mathcal{C} \;\longrightarrow\; \mathsf{S}' \,/_i\, \mathcal{C}'}{\mathsf{S}; S \,/_i\, \mathcal{C} \;\longrightarrow\; \mathsf{S}'; S \,/_i\, \mathcal{C}'} \text{ (E-SEQ)} \qquad \frac{\begin{array}{c}S_i \,/_i\, \mathcal{C} \;\longrightarrow\; S_i' \,/_i\, \mathcal{C}'\\ \{i,j\} = \{1,2\} \;\; S_j' = S_j\end{array}}{\ll S_1 || S_2 \gg \,/_0\, \mathcal{C} \;\longrightarrow \ll S_1' || S_2' \gg \,/_0\, \mathcal{C}'} \text{ (E-BRK)}$$

$$\frac{\begin{array}{c}\forall \rho \in \rho_H . \lfloor \mathcal{C}.\Pi \rfloor_0 \lfloor_{CB} \vdash \rho \not\sqsubseteq CB\\ \lfloor \mathcal{C}.\Pi \rfloor_0 \lfloor_{CB} \vdash q \Rightarrow j = 1 \quad \lfloor \mathcal{C}.\Pi \rfloor_0 \lfloor_{CB} \not\vdash q \Rightarrow j = 2\end{array}}{\mathtt{if}\,\langle q, CB\rangle\; S_1\, S_2 \,/_0\, \mathcal{C} \;\longrightarrow\; S_j \,/_0\, \mathcal{C}} \text{ (E-IFQ-0)} \qquad \frac{\begin{array}{c}i \in \{1,2\}\\ (\lfloor \mathcal{C}.\Pi \rfloor_i) \lfloor_{CB} \vdash q \Rightarrow j = 1\\ (\lfloor \mathcal{C}.\Pi \rfloor_i) \lfloor_{CB} \not\vdash q \Rightarrow j = 2\end{array}}{\mathtt{if}\,\langle q, CB\rangle\; S_1\, S_2 \,/_i\, \mathcal{C} \;\longrightarrow\; S_j \,/_i\, \mathcal{C}} \text{ (E-IFQ-i)}$$

$$\frac{\begin{array}{c}\exists \rho \in \rho_L . \lfloor \mathcal{C}.\Pi \rfloor_0 \lfloor_{\sqcup R} \vdash \mathrm{pc} \sqsubseteq \rho\\ \lfloor \mathcal{C}.\Psi \rfloor_0 = . \quad \Psi^* = (\mathtt{try}_{(Q,\mathrm{pc})} S)\end{array}}{\mathtt{try}_{(Q,\mathrm{pc})} S \,/_0\, \mathcal{C} \;\longrightarrow\; \mathtt{try}_{(Q,\mathrm{pc})} S \,/_0\, \mathcal{C}[\Psi = \Psi^*]} \text{ (E-TR1-0)}$$

$$\frac{\begin{array}{c}i \in \{1,2\} \quad \lfloor \mathcal{C}.\Psi \rfloor_i = .\\ \Psi' = \mathit{updpsi}_i^2(\mathcal{C}.\Psi, (\mathtt{try}_{(Q,\mathrm{pc})} S))\end{array}}{\mathtt{try}_{(Q,\mathrm{pc})} S \,/_i\, \mathcal{C} \;\longrightarrow\; \mathtt{try}_{(Q,\mathrm{pc})} S \,/_i\, \mathcal{C}[\Psi = \Psi']} \text{ (E-TR1-i)}$$

$$\frac{\lfloor \mathcal{C}.\Psi \rfloor_i \neq . \;\; \mathsf{S} \,/_i\, \mathcal{C} \;\longrightarrow\; \mathsf{S}' \,/_i\, \mathcal{C}'}{\mathtt{try}_{(Q,\mathrm{pc})} \mathsf{S} \,/_i\, \mathcal{C} \;\longrightarrow\; \mathtt{try}_{(Q,\mathrm{pc})} \mathsf{S}' \,/_i\, \mathcal{C}'} \text{ (E-TR2)} \qquad \frac{\lfloor \mathcal{C}.\Psi \rfloor_i \neq . \quad \Psi' = \mathit{updpsi}_i^2(\mathcal{C}.\Psi, .)}{\mathtt{try}_{(Q,\mathrm{pc})} \mathtt{skip} \,/_i\, \mathcal{C} \;\longrightarrow\; \mathtt{skip} \,/_i\, \mathcal{C}[\Psi = \Psi']} \text{ (E-TR3)}$$

$$\frac{\Pi' = \mathit{update}_i^2(\mathcal{C}.\Pi, \Delta) \quad \mathit{declass}^2(R, \mathcal{C}.\Pi, \Pi') = \emptyset \quad \lfloor \mathcal{C}.\Psi \rfloor_i = (\mathtt{try}_{(Q,\mathrm{pc})} S)}{\mathtt{update}\;\Delta \,/_i\, \mathcal{C} \;\longrightarrow\; \mathtt{skip} \,/_i\, \mathcal{C}[\Pi = \Pi']} \text{ (E-UP)}$$

Where

$\mathit{declass}^2(R, \Pi_1, \Pi_2) = \mathit{declass}(R, \lfloor \Pi_1 \rfloor_1, \lfloor \Pi_2 \rfloor_1) \cup \mathit{declass}(R, \lfloor \Pi_1 \rfloor_2, \lfloor \Pi_2 \rfloor_2)$

$\mathit{updloc}_0^2(x, \mathsf{v}, \mathsf{M}) = \mathsf{M}[(x, \mathsf{v})]$ $\qquad$ $\mathit{updpsi}_0^2(\Psi, \Psi^*) = \Psi^*$

$\mathit{updloc}_1^2(x, \mathsf{v}, \mathsf{M}) = \mathsf{M}[(x, \ll \mathsf{v} || \lfloor \mathsf{M}(x) \rfloor_2 \gg)]$ $\qquad$ $\mathit{updpsi}_1^2(\Psi, \Psi^*) = \ll \Psi^* || \lfloor \Psi \rfloor_2 \gg$

$\mathit{updloc}_2^2(x, \mathsf{v}, \mathsf{M}) = \mathsf{M}[(x, \ll \lfloor \mathsf{M}(x) \rfloor_1 || \mathsf{v} \gg)]$ $\qquad$ $\mathit{updpsi}_2^2(\Psi, \Psi^*) = \ll \lfloor \Psi \rfloor_1 || \Psi^* \gg$

$\mathit{update}_0^2(\Pi, \Delta) = \mathit{update}(\lfloor \Pi \rfloor_1, \Delta) \oplus \mathit{update}(\lfloor \Pi \rfloor_2, \Delta)$

$\mathit{update}_1^2(\Pi, \Delta) = \mathit{update}(\lfloor \Pi \rfloor_1, \Delta) \oplus \lfloor \Pi \rfloor_2$

$\mathit{update}_2^2(\Pi, \Delta) = \lfloor \Pi \rfloor_1 \oplus \mathit{update}(\lfloor \Pi \rfloor_2, \Delta)$

Fig. 12. Operational semantics of RTI^2

$$\ll \texttt{skip} \parallel \texttt{skip} \gg; S \,/_0\, \mathcal{C} \longrightarrow S \,/_0\, \mathcal{C} \quad \text{(L-SKIP)}$$

$$\texttt{if}\,(\ll E_1 \parallel E_2 \gg)\, S_1\, S_2 \,/_0\, \mathcal{C} \longrightarrow \ll \texttt{if}\,(E_1)\, S_1\, S_2 \parallel \texttt{if}\,(E_2)\, S_1\, S_2 \gg \,/_0\, \mathcal{C} \quad \text{(L-IFE)}$$

$$\frac{\mathsf{v} = \ll \lfloor \mathsf{v}_1 \rfloor_1 + \lfloor \mathsf{v}_2 \rfloor_1 \parallel \lfloor \mathsf{v}_1 \rfloor_2 + \lfloor \mathsf{v}_2 \rfloor_2 \gg}{\mathsf{v}_1 + \mathsf{v}_2 \,/_0\, \mathcal{C} \longrightarrow \mathsf{v} \,/_0\, \mathcal{C}} \quad \text{(L-ADD)}$$

$$\frac{\exists \rho \in \rho_H . \emptyset \vdash \rho \sqsubseteq CB}{\begin{array}{c}\texttt{if}\,(\langle q, CB\rangle)\, S_1\, S_2 \,/_0\, \mathcal{C} \longrightarrow \\ \ll \texttt{if}\,(\langle q, CB\rangle)\, S_1\, S_2 \parallel \texttt{if}\,(\langle q, CB\rangle)\, S_1\, S_2 \gg \,/_0\, \mathcal{C}\end{array}} \quad \text{(L-IFQ)}$$

$$\frac{\exists \rho \in \rho_H . \emptyset \vdash \rho \sqsubseteq \mathrm{pc}}{\texttt{try}_{(Q,\mathrm{pc})}\, S \,/_0\, \mathcal{C} \longrightarrow \ll \texttt{try}_{(Q,\mathrm{pc})}\, S \parallel \texttt{try}_{(Q,\mathrm{pc})}\, S \gg \,/_0\, \mathcal{C}} \quad \text{(L-TR)}$$

Fig. 13. Lifting rules for RTI^2

instance, CB might be a meet of joins of roles, each of which includes ρ'.) In this case, there is no need to consider ρ at all. This is because, by the way ρ_H is defined, ρ' would also satisfy $\rho' \in \rho_H \wedge \rho' \sqsubseteq CB$ and would therefore itself be a witness for the existential in the premise of (L-IFQ), so the premise would be satisfied without need to show $\rho \sqsubseteq \rho'$. Essentially the same argument applies to the premise of (L-TR).

THEOREM 14 (SOUNDNESS). *Given any RTI^2 statement* S *and any RTI^2 configuration* $\mathcal{C}$*, if* $\lfloor \mathcal{C} \rfloor_1 \mid_R = \lfloor \mathcal{C} \rfloor_2 \mid_R$ *and* $\mathsf{S} \,/_i\, \mathcal{C} \longrightarrow \mathsf{S}' \,/_i\, \mathcal{C}'$*, then* $(\lfloor \mathcal{C} \rfloor_1, \lfloor \mathsf{S} \rfloor_1) \longrightarrow^{=} (\lfloor \mathcal{C}' \rfloor_1, \lfloor \mathsf{S}' \rfloor_1)$ *and* $(\lfloor \mathcal{C} \rfloor_2, \lfloor \mathsf{S} \rfloor_2) \longrightarrow^{=} (\lfloor \mathcal{C}' \rfloor_2, \lfloor \mathsf{S}' \rfloor_2)$*, in which* $\longrightarrow^{=}$ *denotes zero or one step of standard execution.*

PROOF. The proof examines each rule in the operational semantics of RTI^2 and verifies that each of the projected configurations takes zero or one steps in the operational semantics of RTI. When the RTI^2 step is given by one of the statements (L-IFE), (L-IFQ), and (L-TR), neither projection takes a step. For the other rules, when $i = 0$, both projections take one step, as can be seen by inspection of the rules. Similarly, one can verify that when $i \in \{1, 2\}$, one projection is unchanged and the other takes a corresponding step in the standard semantics. □

A *maximal*, declassification-free execution is an execution that cannot be extended or cannot be extended without performing a policy update that moves some role from ρ_H to ρ_L. Such configurations can be alternately referred to as stuck configurations(as mentioned in the Pottier and Simonet's paper). Our version of the stuck configuration lemma is as follows:

LEMMA 15 (STUCK CONFIGURATIONS). *Given any RTI^2 configuration* $\mathcal{C}$ *and statement* S*, if no declassification-free execution step can be performed on* $\mathsf{S} \,/_i\, \mathcal{C}$*, then no declassification-free execution step can be performed on* $(\lfloor \mathcal{C} \rfloor_i, \lfloor \mathsf{S} \rfloor_i)$ *for either* $i \in \{1, 2\}$.

PROOF. The proof consists in verifying that for each RTI^2 statement and execution configuration, if no operational semantics rule can be applied to advance the execution, then neither of the RTI executions can take an execution step. Therefore the proof is by induction on structure of S:

—**Case** $\mathsf{S} = \texttt{skip}$**.** S cannot be stuck.

—**Case** $\mathsf{S} = \ll S || S \gg$**.** Since we assume that S is stuck, i.e. neither (E-BRK) nor (L-SKIP) can be applied to S. From the premises of (E-BRK) we can infer that neither configurations can take one step of execution.

—**Case** $\mathsf{S} = \mathsf{S}; S$**.** Since we assume that S is stuck, neither (E-SEQ) nor (R-SEQ) can be applied. From the induction hypothesis on the premises of (E-SEQ) and (R-SEQ), we can infer that no execution step can be performed on $(\lfloor \mathcal{C} \rfloor_i, \lfloor \mathsf{S} \rfloor_i)$ for either $i \in \{1, 2\}$.

—**Case** $\mathsf{S} = \texttt{if}\ (\mathsf{E})\ S\ S, \mathsf{S} = \texttt{while}\ (E)\ S$**.** If E is of the form $\ll E_1 || E_2 \gg$ or v, it cannot be stuck, since (L-IFE) can be applied to it. If E is of the form E and it is stuck, it means that (E-IFE) cannot be applied to it, which means that one of the variables in E was not initialized in either $\lfloor \mathcal{C} \rfloor_1$ or $\lfloor \mathcal{C} \rfloor_2$.

—**Case** $\mathsf{S} = x := \mathsf{E}$**.** Since we assume that S is stuck, (E-ASE) cannot be applied. From the premises we can infer that E cannot take another execution step. This is possible only if (E-ADL), (E-ADR) and (E-ADV) cannot be applied. These executions cannot be applied only if (E-VAR) cannot be applied, i.e. some variable in E is not initialized in both the executions. This will ensure that both the RTI executions are stuck too.

—**Case** $\mathsf{S} = \texttt{try}_{(Q,\mathrm{pc})}\ \mathsf{S}_1$**.** If the rules (E-TR1-0), (E-TR3) or (L-TR) can be applied, S cannot be stuck. The only case in which S is stuck is when (E-TR2) needs to be applied. From the premise of (E-TR2) we infer that S_1 is stuck. By applying the induction hypothesis, we get that both the RTI execution configurations that constitute S_1 are stuck; consequently those that constitute S are also stuck.

—**Case** $\mathsf{S} = \texttt{if}\ \langle q, CB \rangle\ S_1\ S_2$**.** S cannot be stuck since one of (E-IFQ-0) and (L-IFQ) can be applied to it.

—**Case** $\mathsf{S} = \texttt{update}\ \Delta$**.** This statement cannot be stuck.

□

THEOREM 16 (COMPLETENESS). *Given any RTI program statement S and two configurations $\mathcal{E}_1$ and $\mathcal{E}_2$ such that $\mathcal{E}_1 \mid_R = \mathcal{E}_2 \mid_R$, there exists an RTI^2 configuration $\mathcal{C}$ such that $\lfloor \mathcal{C} \rfloor_i = \mathcal{E}_i$ for $i \in \{1, 2\}$. Furthermore, if there are maximal, finite, declassification-free executions $(\mathcal{E}_1, S) \longrightarrow^* (\mathcal{E}_1', S')$ and $(\mathcal{E}_2, S) \longrightarrow^* (\mathcal{E}_2', S'')$ then there exists a maximal, finite, declassification-free execution $S \,/_i\, \mathcal{C} \longrightarrow^* \mathsf{S} \,/_i\, \mathcal{C}'$ such that $\lfloor \mathcal{C}' \rfloor_i = \mathcal{E}_i$ for $i \in \{1, 2\}$.*

PROOF. The proof of the first statement is by construction. Given two RTI configurations $\mathcal{E}_1$ and $\mathcal{E}_2$ such that $\mathcal{E}_1 \mid_R = \mathcal{E}_2 \mid_R$, we can construct an RTI^2 configuration $\mathcal{C}$, such that $\mathcal{C}.\Pi = \ll \mathcal{E}_1.\Pi || \mathcal{E}_2.\Pi \gg$, $\mathcal{C}.\mathsf{M} = \mathcal{E}_1.M \oplus \mathcal{E}_2.M$ and $\mathcal{C}.\Psi = \ll \mathcal{E}_1.\Psi || \mathcal{E}_2.\Psi \gg$. This $\mathcal{C}$ satisfies our requirement that $\lfloor \mathcal{C} \rfloor_i = \mathcal{E}_i$ for $i \in \{1, 2\}$.

For the second part, we begin by showing that the RTI^2 semantics cannot take infinitely many steps without performing a step that corresponds to an RTI step. When a non-split RTI^2 statement transits to a split statement, it does so via one of the lifting rules, (L-IFE), (L-IFQ), and (L-TR), which clearly cannot be repeated. Non-split expressions become split expression only in (E-VAR), in which a variable can be replaced by a split value from memory. Clearly this corresponds to one memory look-up step in each of the two RTI executions. When a non-split RTI^2 statement or expression is executed one step, without transiting to a split statement or expression, and these rules are essentially the same in RTI^2 as they are in RTI, this step corresponds directly to one step in each of the RTI executions. Split RTI^2 statements are handled by (E-BRK) and by (L-SKIP). The premise of (E-BRK)

requires that one of the two RTI executions proceeds one step, so we are done. (L-SKIP) corresponds to a step being taken in both RTI executions.

The number of times (L-ADD) can be applied is bounded by the depth of the expression, as each application raises the split one level closer to the root of the expression. Each of the other rules whereby a split expression can transit to a split expression involve the application of an operator, and hence corresponds to a step in the RTI executions. This covers all cases, concluding the argument.

The argument for the rest of the second part is as follows. Because the RTI operational semantics is deterministic and the RTI^2 semantics is sound, any RTI^2 execution sequence that can be found will correspond to initial sub-sequences of the given RTI sequences. And, as argued above, the RTI^2 execution sequence cannot take an unbounded number of steps without performing a step that corresponds to a step in at least one of the given RTI executions. So the only way for the RTI^2 execution to fail to correspond to the pair of RTI executions would be for it to get stuck in the sense that no declassification-free step can be performed on it, while this does not occur in the each of the RTI executions. However this possibility is ruled out by Lemma 15. □

B. PROVING NONINTERFERENCE

B.1 Augmented Typing Judgments

The augmented typing judgments that can be shown by using rules given in figure 14 are used by the subject reduction theorem. These judgments are shown below to be preserved by execution steps. They capture an induction hypothesis that enables one to show inductively that the two parts of an RTI^2 configuration remain indistinguishable to the adversary as execution proceeds. Note that augmented types drop the command types. These were needed in the standard judgments because of the need to infer appropriate bounds on policy statements to use in evaluating queries. The information-flow due to control dependencies is reasoned about by in the non-interference result entirely through the use of the pc. To support doing so, the type rule (T-IfQ) for RTI^2 raises the pc to reflect potential control dependence on policy statements used to evaluate the query. This was not necessary in RTI, where this form of information flow was controlled by using command types.

B.2 Maintaining Observational Equivalence

Together, the results in this section show that the two RTI executions modeled by a single RTI^2 execution remain indistinguishable to the adversary when a single step is taken. Lemma 17 shows this in high contexts and Lemma 19 shows it in low contexts.

LEMMA 17 (OBSERVATIONAL EQUIVALANCE IN HIGH CONTEXTS). *Suppose for a program* S *and a configuration* $\mathcal{C}$ *such that* $\Omega \vdash_{\mathcal{C},i,R} \mathsf{S}$ *and* $\mathsf{S} /_i \mathcal{C} \longrightarrow \mathsf{S}' /_i \mathcal{C}'$. *It follows that if* $\forall \rho \in R\,.\,(\lfloor \mathcal{C}.\Pi \rfloor_i \,|_R \not\vdash \Omega.\mathrm{pc} \sqsubseteq \rho)$ *then* $\mathcal{C} \,|_R = \mathcal{C}' \,|_R$.

PROOF. The proof is by induction on the structure of the derivation of the single step. When $\mathcal{C} = \mathcal{C}'$, the lemma holds trivially. The only observable components of a configuration $\mathcal{C}$ are $\mathcal{C}.\mathsf{M}$ and $\mathcal{C}.\Pi$. We consider memory and policy effects separately.

Let us first consider memory effects. All updates to memory occur through the function updloc_i^2. Applications of this function appear in the premise of (E-ASV).

In this case, only the memory changes, not the policy. The assignment must be typechecked by using (T-A1) or (T-A2) preceded in either case by (T-Assign), the premise of

$$\frac{\Omega \models_{R,i} \mathcal{C} \quad \lfloor \mathcal{C}.\Psi \rfloor_i = \cdot \quad \exists CB.\Omega \vdash \mathsf{S} : CB\ \texttt{cmd}}{\Omega \vdash_{\mathcal{C},i,R} \mathsf{S}} \text{ (T-A1)}$$

$$\frac{\begin{array}{c}\Omega[Q = Q'] \models_{R,i} \mathcal{C} \ \ \lfloor \mathcal{C}.\Psi \rfloor_i = (\texttt{try}_{(Q,\mathrm{pc})}\mathsf{S}_1) \\ (\mathsf{S} = \texttt{try}_{(Q,\mathrm{pc})}\mathsf{S}_2 \Rightarrow \mathsf{S}' = \mathsf{S}_2) \\ (\mathsf{S} \neq \texttt{try}_{(Q,\mathrm{pc})}\mathsf{S}_2 \Rightarrow \mathsf{S}' = \mathsf{S}) \\ Q' \subseteq Q \ \ \Omega.Q \subseteq Q \ \ \forall \langle q, CB' \rangle \in Q'.\lfloor \mathcal{C}.\Pi \rfloor_i \lfloor_{b'} \vdash q \\ \Omega.\mathrm{pc} = \mathrm{pc} \ \ \exists CB.\Omega[Q = Q'][\Phi = try] \vdash \mathsf{S}' : CB\ \texttt{cmd}\end{array}}{\Omega \vdash_{\mathcal{C},i,R} \mathsf{S}} \text{ (T-A2)}$$

$$\frac{\begin{array}{c}\mathcal{C}.\Psi = \ll \Psi_1^* || \Psi_2^* \gg \quad \exists \rho \in \rho_H.\mathrm{pc}' = \mathrm{pc} \sqcup \rho \\ \forall i \in \{1,2\}.\Omega[\mathrm{pc} = \mathrm{pc}'] \vdash_{\mathcal{C},i,R} S_i\end{array}}{\Omega \vdash_{\mathcal{C},0,R} \ll S_1 || S_2 \gg} \text{ (T-A3)}$$

$$\frac{\Omega \vdash_{\mathcal{C},i,R} \texttt{try}_{(Q,\mathrm{pc})}\mathsf{S} \quad \exists CB.\Omega \vdash S' : CB\ \texttt{cmd}}{\Omega \vdash_{\mathcal{C},i,R} \texttt{try}_{(Q,\mathrm{pc})}\mathsf{S}; S'} \text{ (T-A4)}$$

Fig. 14. Augmented typing judgment

which gives us $\Omega'.Q \vdash \Omega'.\mathrm{pc} \sqsubseteq \mathit{lab}(\Omega'.\Gamma(x))$ in which $\Omega'.\mathrm{pc} = \Omega.\mathrm{pc}$ and $\Omega'.\Gamma = \Omega.\Gamma$. By Lemma 11 and the premise of (T-A2), $\forall \langle q, CB' \rangle \in Q'.\lfloor \mathcal{C}.\Pi \rfloor_i \lfloor_{CB'} \vdash q$, it follows that $\lfloor \mathcal{C}.\Pi \rfloor_i \,|_R \vdash \Omega.\mathrm{pc} \sqsubseteq \mathit{lab}(\Omega.\Gamma(x))$.

Suppose for contradiction that $\mathcal{C}.\mathsf{M} \mid_{R,\mathcal{C}.\Pi} \neq \mathcal{C}'.\mathsf{M} \mid_{R,\mathcal{C}.\Pi}$. Since $\mathcal{C}.\Pi = \mathcal{C}'.\Pi$, for the change to be observable, we must have that $x \in \mathit{dom}(\mathcal{C}.\mathsf{M} \mid_{R,\mathcal{C}.\Pi})$. Since the memory update does not change the domain, we have $\mathit{dom}(\mathcal{C}.\mathsf{M} \mid_{R,\mathcal{C}.\Pi}) = \mathit{dom}(\mathcal{C}'.\mathsf{M} \mid_{R,\mathcal{C}.\Pi})$. From the definition of $\mathsf{M} \mid_{R,\Pi}$, $\mathcal{C}.\mathsf{M} \mid_{R,\mathcal{C}.\Pi} \neq \mathcal{C}'.\mathsf{M} \mid_{R,\mathcal{C}.\Pi}$ is possible only if there exists $\rho \in R$ such that $\lfloor \mathcal{C}.\Pi \rfloor_i \mid_R \vdash \mathit{lab}(\Omega.\Gamma(x)) \sqsubseteq \rho$. However, if this were the case, it would mean that $\lfloor \mathcal{C}.\Pi \rfloor_i \mid_R \vdash \Omega.\mathrm{pc} \sqsubseteq \rho$ for some $\rho \in R$, which contradicts the assumption.

All policy updates occur in the rule (E-UP), where the function update_i^2 is used. If a statement that is added or removed is observable at level R, its label is dominated by some $\rho \in R$. However, the label of such a statement must dominate $\Omega.\mathrm{pc}$. So it follows that some $\rho \in R$ dominates $\Omega.\mathrm{pc}$, which contradicts the hypothesis. Thus, no observable policy effect can occur. □

COROLLARY 18. *If* $\ll S_1 \parallel S_2 \gg /_0 \mathcal{C} \longrightarrow \ll S_1' \parallel S_2' \gg /_0 \mathcal{C}'$ *and* $\Omega \vdash_{\mathcal{C},0,R} \ll S_1 \parallel S_2 \gg$ *then* $\mathcal{C} \mid_R = \mathcal{C}' \mid_R$

PROOF. The proof follows from lemma 17, the premise of (T-SBR) and (T-A3) which requires S_1 and S_2 to be checked under a high pc. □

LEMMA 19. *If* $\mathsf{S} /_0 \mathcal{C} \longrightarrow \mathsf{S}' /_0 \mathcal{C}'$, *then if* $\Omega \vdash_{\mathcal{C},0,R} \mathsf{S}$, *then* $\lfloor \mathcal{C}' \rfloor_1 \mid_R = \lfloor \mathcal{C}' \rfloor_2 \mid_R$

PROOF. The proof of this lemma proceeds similar to that of lemma 17. From the hypothesis we know that $\Omega \vdash_{\mathcal{C},0,R} \mathsf{S}$, and therefore $\lfloor \mathcal{C} \rfloor_1 \mid_R = \lfloor \mathcal{C} \rfloor_2 \mid_R$. For any step in the execution, $\mathcal{C}$ changes only if either the memory changes or if the policy changes or both. Bracketed expressions and statements type-check only under high pc as given by the premises of (T-EBR) and (T-SBR). Since the memory or policy diverges only when bracketed expressions and statements are executed, the observable part of the configuration is

the same and hence the proof. □

B.3 Subject Reduction

This section proves the central theorem from which noninterference follows by a simple induction. This subject reduction theorem shows that augmented well typing is preserved by execution steps. Augmented well typing requires that the configuration is well formed, and this requires that the two RTI executions modeled by a single RTI^2 execution have configurations that are indistinguishable to the adversary.

THEOREM 20 (SUBJECT REDUCTION). *Assume we are given a program* S *and a configuration* $\mathcal{C}$ *such that* $\mathsf{S}\,/_i\,\mathcal{C} \longrightarrow \mathsf{S}'\,/_i\,\mathcal{C}'$. *Given any context* Ω, *any a set of roles* R, *any* $i \in \{0,1,2\}$, *and any confidentiality label* CL *or confidentiality bound* CB, *if* $\Omega \vdash_{\mathcal{C},i,R} \mathsf{S}$ *and* ($i = 0$ *or* $\exists \rho \in \rho_H.\rho \sqsubseteq \Omega.\mathrm{pc}$), *then* $\Omega \vdash_{\mathcal{C}',i,R} \mathsf{S}'$. *Similarly, if* $\mathsf{E}\,/_i\,\mathcal{C} \longrightarrow \mathsf{E}'\,/_i\,\mathcal{C}$, $\Omega \models_{R,i} \mathcal{C}$, *and* $\Omega \vdash \mathsf{E} : t_{CL}$, *then* $\Omega \vdash \mathsf{E}' : t_{CL}$.

PROOF. The proof is by induction on the structure of the derivations of $\mathsf{S}\,/_i\,\mathcal{C} \longrightarrow \mathsf{S}'\,/_i\,\mathcal{C}'$ and $\mathsf{E}\,/_i\,\mathcal{C} \longrightarrow \mathsf{E}'\,/_i\,\mathcal{C}$.

(E-VAR): By assumption $\Omega \vdash x : t_{CL}$ and by clause (1) of $\Omega \models_{R,i} \mathcal{C}$, $\Omega \vdash \mathsf{M}(x) : \Omega.\Gamma(x)$. By the premise of (E-VAR), $\lfloor \mathsf{M}(x) \rfloor_i = \mathsf{v}$. When $i = 0$, we obtain $\Omega \vdash \mathsf{v} : t_{CL}$ immediately, as required. When $i \in \{1,2\}$, the result follows by the fact that (T-EBR) must be used to show $\Omega \vdash \mathsf{v} : t_{CL}$, the premises of which give us $\Omega \vdash \lfloor \mathsf{M}(x) \rfloor_i : t_{CL}$.

(E-ADL, E-ADR, E-ADV): Trivial. Can be proved using the hypothesis, the rules' premises, and the rule (T-PLUS).

(E-ASE, E-SKIP): Trivial.

(E-ASV): To prove $\Omega \vdash_{\mathcal{C}',i,R} \texttt{skip}$ we just need to show that $\Omega \models_{R,i} \mathcal{C}'$. The rest of the proof is trivial. The first clause of $\Omega \models_{R,i} \mathcal{C}'$ follows from the assumption that $\Omega \vdash_{\mathcal{C},i,R} x := \mathsf{v}$, which can be shown only by using (T-A1) or (T-A2), both of which require that $\Omega \models_{R,i} \mathcal{C}$. This can now be used to prove $\Omega \vdash \mathsf{v} : \Omega.\Gamma(x)$ as required for clause (1). Clause (2) is trivially satisfied since $updloc$ does not change Π. When $i = 0$, clause (3) follows from Lemma 19. When $i \neq 0$, it follows from Corollary 18.

(E-SEQ): The proof of this case can be divided into several cases based on the judgment used for the derivation of $\Omega \vdash_{\mathcal{C},i,R} \mathsf{S}; S$.

—Case (T-A1): Since (T-A1) is used, $\lfloor \mathcal{C}.\Psi \rfloor_i = \cdot$. The proof can again be split into two cases: (i) If $\lfloor \mathcal{C}'.\Psi \rfloor_i = \cdot$, then (T-A1) can be applied again by using $\Omega \vdash \mathsf{S}' : CB_1\ \texttt{cmd}$ and $\Omega \models_{R,i} \mathcal{C}'$ from the induction hypothesis. We also get $\Omega \vdash S : CB_2\ \texttt{cmd}$ from the premise of (T-A1) and using (T-Seq), where $CB = CB_1 \sqcap CB_2$. (ii) If $\lfloor \mathcal{C}'.\Psi \rfloor_i \neq \cdot$, then we must prove $\Omega \vdash_{\mathcal{C}',i,R} \mathsf{S}'; S$ by using (T-A4). The first premise of (T-A4) is obtained from the induction hypothesis and the second premise is obtained from the proof of $\Omega \vdash \mathsf{S}, S$ and hence the result.

—Case (T-A2): This means that the pair of statements is in the body of the try block and this is precluded by our static semantics.

—Case (T-A3): Since a sequence of statements cannot be a statement pair, this case is not possible.

—Case (T-A4): In this case, S has the form $\texttt{try}_{(Q,\mathrm{pc})}\mathsf{S}_1$ and the three premises of (T-A4) are $\Omega \vdash_{\mathcal{C},i,R} \texttt{try}_{(Q,\mathrm{pc})}\mathsf{S}_1$, $\Omega \vdash S : CB_2\ \texttt{cmd}$, and $CB = CB_1 \sqcap CB_2$. There are now

three cases: (i) If $\lfloor \mathcal{C}.\Psi \rfloor_i = \cdot$, then (E-TR1-0) or (E-TR1-i) applies, and so $\lfloor \mathcal{C}'.\Psi \rfloor_i \neq \cdot$. From the first premise of (T-A4) shown above and the induction hypothesis, we obtain $\Omega \vdash_{\mathcal{C}',i,R} \texttt{try}_{(Q,\mathrm{pc})} \mathsf{S}_1'$. Combining this with the second premise of (T-A4) shown above and using (T-A4), we obtain the desired result. (ii) If $\lfloor \mathcal{C}.\Psi \rfloor_i \neq \cdot$ and $\lfloor \mathcal{C}'.\Psi \rfloor_i \neq \cdot$, (E-TR2) applies and we proceed as in case (i). (iii) If $\lfloor \mathcal{C}.\Psi \rfloor_i \neq \cdot$ and $\lfloor \mathcal{C}'.\Psi \rfloor_i = \cdot$, (E-TR3) applies and $\mathsf{S}' = \texttt{skip}$. $\Omega \vdash_{\mathcal{C},i,R} \texttt{skip}$ follows from the induction hypothesis. This can be proved only by using (T-A1), from the premises of which we get $\Omega \models_{R,i} \mathcal{C}'$ and $\Omega \vdash \texttt{skip} : CB_1\ \texttt{cmd}$. By using the latter and the second premise of (T-A4) shown above in (T-Seq), we obtain $\Omega \vdash \texttt{skip}; S : CB\ \texttt{cmd}$. By using this, $\Omega \models_{R,i} \mathcal{C}'$, and the case assumption, we obtain the desired result.

(E-IFE, E-IFV, E-WHL): Trivial.

(E-IFQ-0, E-IFQ-i): For $i \in \{0,1,2\}$ the derivation of $\Omega \vdash_{\mathcal{C},i,R} \texttt{if}\ \langle q, CL \rangle\ S_1\ S_2$ must end with an application of either (T-A1) or (T-A2).

In either case, from the assumption that the if statement type-checks, we have that both legs type-check, so the induction obligation follows immediately.

(E-TR1-0): The only difference between $\Omega \vdash_{\mathcal{C},i,R} \texttt{try}_{(Q,\mathrm{pc})} S$ and $\Omega \vdash_{\mathcal{C}',i,R} \texttt{try}_{(Q,\mathrm{pc})} S$ is the fact that $\mathcal{C}.\Psi = \cdot$ and $\mathcal{C}.\Psi' \neq \cdot$. (T-A1) is used to prove the first judgment, while (T-A2) should be used to prove the second. The premises of (T-A2) follow trivially from those of (T-A1).

(E-TR1-i): The proof of this case is similar to that of (E-TR1-0).

(E-TR2, E-TR3): Trivial

(E-UP): Showing that $\Omega \vdash_{\mathcal{C},i,R} \texttt{skip}$ can be done only by using (T-A2). For this, we will take $Q' = \emptyset$ in the premise of (T-A2). Since the skip statement type checks with any Ω, the non-trivial part is showing that $\Omega[Q = \emptyset] \models_{R,i} \mathcal{C}'$. Clauses (1) and (2) are straightforward. For clause (3), it suffices to show that $\lfloor \mathcal{C}'|_R \rfloor_1 = \lfloor \mathcal{C}'|_R \rfloor_2$. $\Omega \models_{R,i} \mathcal{C}$ gives us $\lfloor \mathcal{C}|_R \rfloor_1 = \lfloor \mathcal{C}|_R \rfloor_2$. When $i = 0$, the result follows from Lemma 19. When $i \neq 0$, the result follows from Corollary 18.

(R-SEQ): Trivial

(E-BRK): If i and j could be either 1 or 2, the derivation of $\Omega \vdash_{\mathcal{C},0,R} \ll S_1 || S_2 \gg$ ends in (T-A3), from the premises of which we get $S_j' = S_j$ and $\lfloor \mathcal{C}' \rfloor_j = \lfloor \mathcal{C} \rfloor_j$. Form the induction hypothesis we can prove $\Omega \vdash_{\lfloor \mathcal{C}' \rfloor_i,0,R} S_i'$. Using (T-A3) again we can prove $\Omega \vdash_{\mathcal{C},0,R} \ll S_1' || S_2' \gg$.

(L-IFE): Fix any Ω satisfying $\Omega \vdash_{\mathcal{C},0,R} \texttt{if}\ (\ll E_1 \parallel E_2 \gg)\ S_1\ S_2$. The obligation is to show that $\Omega \vdash_{\mathcal{C},0,R} \ll \texttt{if}\ (E_1)\ S_1\ S_2 \parallel \texttt{if}\ (E_2)\ S_1\ S_2 \gg$. This can be shown by using (T-A3), a premise of which is that there exists a $\rho \in \rho_H$ such that for each $i \in \{1,2\}$,

$$\Omega[\mathrm{pc} = \hat{\mathrm{pc}}] \vdash_{\mathcal{C},i,R} \texttt{if}\ (E_i)\ S_1\ S_2 \tag{12}$$

in which $\hat{\mathrm{pc}} = \Omega.\mathrm{pc} \sqcup \rho$.

Examining the assumption that $\Omega \vdash_{\mathcal{C},0,R} \texttt{if}\ (\ll E_1 \parallel E_2 \gg)\ S_1\ S_2$, the proof of this must end with an application of either (T-A1) or (T-A2), depending on whether or not the statement is inside a transaction. This last step must be preceded by a step using (T-IfE) to show $\Omega \vdash \texttt{if}\ (\ll E_1 \parallel E_2 \gg)\ S_1\ S_2 : CB\ \texttt{cmd}$, which requires using (T-EBR) to show $\Omega \vdash \ll E_1 \parallel E_2 \gg : CB''$, where there exist CB' and ρ' such that $CB'' = CB' \sqcup \rho'$,

$\rho' \in \rho_H$, and $\Omega \vdash E_1 : CB'$ and $\Omega \vdash E_2 : CB'$. The use of (T-IfE) also requires that the following premises can be shown: $\Omega[\mathrm{pc} = \mathrm{pc}''] \vdash S_i : CB_i$ `cmd` for $i \in \{1, 2\}$, in which $\mathrm{pc}'' = \Omega.\mathrm{pc} \sqcup CB''$ and $CB = CB_1 \sqcap CB_2$.

From these, we now begin to meet our obligation. Observe that $\mathrm{pc}'' = \Omega.\mathrm{pc} \sqcup \rho' \sqcup CB'$. From $\Omega \vdash E_1 : CB'$, and $\Omega[\mathrm{pc} = \mathrm{pc}''] \vdash S_i : CL_i$ `cmd` for $i \in \{1, 2\}$, (T-IfE) now gives us $\Omega[\mathrm{pc} = \Omega.\mathrm{pc} \sqcup \rho'] \vdash$ `if` $(E_1)\ S_1\ S_2 : CB$ `cmd`. We obtain $\Omega[\mathrm{pc} = \Omega.\mathrm{pc} \sqcup \rho'] \vdash$ `if` $(E_2)\ S_1\ S_2 : CB$ `cmd` similarly. We now use (T-A1) or (T-A2), depending on whether or not the statement is inside a transaction, to complete the proof of (12), with the required value of ρ given by ρ'. (Notice that in the case that (T-A2) is used, the same value of Q' satisfies the respective premises as was used in the application of (T-A2) that concludes the proof of the assumption.)

(L-SKIP,L-ADD): Trivial.

(L-IFQ): The hypothesis that $\Omega \vdash_{\mathcal{C},0,R}$ `if` $(\langle q, CB\rangle)\ S_1\ S_2$ can be shown by using (T-A1) when the statements is outside any `try` statement and by using (T-A2) when it inside a `try`. In each case, we show how to meet the proof obligation, $\Omega \vdash_{\mathcal{C},0,R} \ll$ `if` $(\langle q, CB\rangle)\ S_1\ S_2 \parallel$ `if` $(\langle q, CB\rangle)\ S_1\ S_2 \gg$, by using (T-A3).

Consider the (T-A1) case first. The following two premises of (T-A1) are known to be met: $\Omega \models_{R,0} \mathcal{C}$ and $\Omega \vdash$ `if` $(\langle q, CB\rangle)\ S_1\ S_2 : CB$ `cmd`. The latter of these can be shown only by using (T-IfQ), which ensures that the two legs of the conditional can be type checked with the pc value given by $\widehat{\mathrm{pc}} = \mathrm{pc} \sqcup (\sqcup\{\rho_1 \mid \emptyset \vdash \rho_1 \sqsubseteq CB\}$. Note that the premise of (L-IFQ) guarantees that $\exists \rho \in \rho_H.\emptyset \vdash \rho \sqsubseteq CB$, so $\widehat{\mathrm{pc}}$ satisfies $\exists \rho' \in \rho_H.\rho' \sqsubseteq \widehat{\mathrm{pc}}$. Fix such a ρ'.

We will use (T-A1) to satisfy the following premise of (T-A3) so as to meet the obligation: $\forall i \in \{1, 2\}.\Omega[\mathrm{pc} = \mathrm{pc}'] \vdash_{\mathcal{C},i,R}$ `if` $(\langle q, CB\rangle)\ S_1\ S_2$ in which $\mathrm{pc}' = \mathrm{pc} \sqcup \rho$ for some $\rho \in \rho_H$. We can take this ρ to be the ρ' fixed in the previous paragraph. Thus, when using (T-IfQ) to satisfy the premise of (T-A1), the same $\widehat{\mathrm{pc}}$ can be used. Each application of (T-A1) requires $\Omega \models_{R,i} \mathcal{C}$ for the respective $i \in \{1, 2\}$. Clauses (1) and (3) follow immediately from $\Omega \models_{R,0} \mathcal{C}$. Clause (2) also follows from this because at least as many policy statements are available to prove $\lfloor \Pi \rfloor_i \downarrow_{CB} \vdash q$ when $i \in \{1, 2\}$ as when $i = 0$. Because $\mathcal{C}$ is unchanged by (L-IFQ), the other premise of (T-A1) follow trivially from the case assumption that (T-A1) is used to prove the hypothesis. Thus we can meet the obligation in this case.

The (T-A2) case is similar. In it (T-A2) is used to satisfy the premise of (T-A3).

(L-TR): Trivial, using (T-A3) and then (T-A1) to prove the premises. $\square$