Secure Resource Access for Open Systems

JAMES RIELY AND MATTHEW HENNESSY

ABSTRACT. In open distributed systems of mobile agents, where code from remote sites may run locally, protection of sensitive data and system resources is of paramount importance. We present a security-based typing system that provides such protection, using a mix of static and runtime typing; mobile agents are allowed access to local resources in accordance with security clearance. We formalize security violations as runtime errors and prove that, using our semantics, security violations cannot occur at "good" sites, *i.e.* sites under control of a particular administrative domain.

1 Introduction

In distributed systems, security is of the utmost importance, for example ensuring the integrity of a local address space or limiting access to certain data to appropriate principals. This is particularly so in *open* networks, where program code may move between administrative domains and remote locations may, intentionally or otherwise, harbor malicious code.

Type safety is a necessary pre-condition to secure computation. In [9, 16] typing schemes have been proposed for mobile agents which guarantee no misuse of local resources during execution; in the latter paper the guarantee holds true even in the presence of hostile agents. Several other schemes to ensure that foreign agents are type-safe have been suggested. They are all variations on the following principle:

> Before an incoming agent P is run at a site it is checked by the site to ensure that it will not violate implicit or explicit access privileges.

For example in Java [37] applets from non-local URLs are checked by a bytecode verifier before loading; the bytecode verification can be formalized using type rules, [34, 19]. Another instance of the principle may be found in the proof-carrying-code of[22, 23]; an incoming agent must provide a proof that the code it is proposing to run locally satisfies appropriate constraints set down by the host.

Type safety alone, however, is not sufficient. There are often certain resources that an incoming agent should not use at all, even in a type-safe way. For example a machine may hold confidential information, such as financial data, which incoming agents should not be able to access. More generally incoming agents should have access to local resources in accordance with their allocated privileges. For example read access to certain areas of memory may be relatively unconstrained while write access is allowed only to certain designated agents; more sensitive operations, such as killing active threads, may be reserved for system agents which originate locally. To describe such constraints, access privileges are granted in accordance with some *security policy* — with security levels chosen to represent the access rights of the various principals in the system. It is then up to particular administrative domains to ensure that this security policy is never breached.

In this paper, we develop a strategy for ensuring secure resource access in open systems. In Java, access control is enforced at runtime, as each access is requested. Capability-passing approaches, as we studied in [15], do not require such dynamic checks, but neither do they extend smoothly to open networks. Here we show that load-time checking is sufficient to establish secure access control in open networks. As in [28], code initialized on a site is checked statically; code that migrates across the network is checked at load-time. However, here, in addition to type safety as in [28], our results also ensure secure access control. They use a novel type language and typing system based on security levels; in particular locations are assigned security levels which indicate the degree to which they can be trusted.

jriely@csc.ncsu.edu. DePaul CTI, 243 S. Wabash, Chicago, IL 60604, USA. Research funded by NSF grant EIA-9805604.
matthewh@cogs.susx.ac.uk. COGS, Univ. of Sussex, Brighton BN1 9QH, UK. Research funded by EPSRC project GR/K60701 and CONFER II.

In the rest of this introduction we describe (1) the formal framework used for our results, (2) security policies and their application to closed networks, (3) the extensions required to handle open networks. We discuss related work in Section 5. In the interest of clarity, we use a somewhat simplified notation throughout the rest of this section.

1.1 The Formal Framework: Dπ

In Dπ a distributed network is described by terms such as

$$\ell[\![P]\!] \,|\, k[\![Q]\!] \,|\, \ell[\![R]\!]$$

representing three agents, P and R running at a site named ℓ and Q at site k. The agents, or threads, P, Q, and R, are terms from an augmented polyadic π-calculus. In addition to the usual channel based communication of the π-calculus,

$$k[\![a!\langle v\rangle\, P]\!] \;\mid\; k[\![a?(X)\, Q]\!] \longrightarrow k[\![P]\!] \;\mid\; k[\![Q\{\!|{}^{v}/_{X}|\!\}]\!]$$

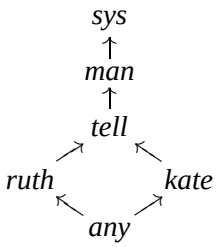
Dπ includes the movement operator "go", with the reduction rule

$$k[\![\mathsf{go}\,\ell.\, P]\!] \longrightarrow \ell[\![P]\!] \qquad (*)$$

representing the migration of the agent P from the site k to the site ℓ. Dπ locations correspond roughly to address spaces; thus the move operator takes an agent from one address space to another. Each location provides certain resources for roving agents and access to these resources is maintained using a type discipline which associates types to every resource at a location. The type of a location defines the discipline it expects agents to follow when accessing the address space. The type will also constrain access to particular local resources via a level of security clearance associated with agents. Specifically we introduce a new language of security types, allowing us to develop a new type discipline which can express and enforce discretional access policies in open networks containing hostile agents. These security types are obtained by adding security levels to the type language of [15]

1.2 Security Topologies and Security Policies

A *security topology* is a complete lattice $\mathcal{S}$ of security levels. As an example consider the following topology, where $\sigma \to \rho$ indicates that σ is a lower security-level than ρ, which might be associated with the management of a bank account:



Here, *sys* is the highest security level, given only to the system software; *man* represents the security level of bank managers; *tell* the level of bank tellers; *ruth* and *kate* the level of two particular clients; and *any* the lowest level, available to anyone. For expository purposes we have deliberately chosen a very simple topology, with a modest goal: the protection of resources at a single location, a bank account.

Given such a security topology $\mathcal{S}$, a *security policy* assigns a security level to each location ℓ and capability γ. For example suppose that *accnt* is the location which represents the bank account in question, with methods for depositing and withdrawing money and for closing the account; the first two take an integer parameter, while the third, , takes none. Then *accnt* might be declared with the type

$$\mathit{accnt} : \mathsf{loc}_{\mathit{sys}}\big\{\mathit{deposit}{:}\mathsf{w}_{\mathit{any}}\langle\mathsf{int}\rangle, \mathsf{r}_{\mathit{sys}}\langle\mathsf{int}\rangle \;\; \mathit{withdraw}{:}\mathsf{w}_{\mathit{ruth}}\langle\mathsf{int}\rangle, \mathsf{r}_{\mathit{ssys}}\langle\mathsf{int}\rangle \;\; \mathit{close}{:}\mathsf{w}_{\mathit{man}}\langle\rangle, \mathsf{r}_{\mathit{sys}}\langle\rangle\big\}$$

The location itself, *accnt*, has associated with it the security level *sys*, indicating that any agent with security level up to and including *sys*, *i.e.* any agent, can have access to the site. However there are further constraints on access to particular resources at *accnt*. For example only agents at security level *sys* can read from the three resources *deposit*, *withdraw* and *close*; intuitively only system software can service requests to deposit to, withdraw from, or close the account. On the other hand any agent can deposit into the account, *i.e.* has access to the write capability on the resource *deposit*, while only agents with security clearance at least *ruth* can withdraw. Finally only agents at security level *man*, that of the bank manager, can close the account.

The security level σ associated with the locations plays an important role. Only agents at level at most σ are given access to the location, but more importantly we know that agents originating from that location are also at level at most σ. Thus, for example, if we suppose that the two clients *ruth* and *kate* have associated with them the locations *ruth-pc*, *kate-pc* respectively, then these should have associated with them types of the form

$$\mathit{ruth\text{-}pc}{:}\mathsf{loc}_{\mathit{ruth}}\{\ldots\} \quad \mathit{kate\text{-}pc}{:}\mathsf{loc}_{\mathit{kate}}\{\ldots\}$$

In a well behaved network this guarantees that all agents migrating from *ruth-pc* have either security level *ruth* or *any*, while those emanating from *kate-pc* have security level *kate* or *any*.

Note that the security policy we have just outlined presents selective views of the location *accnt* to different principals, depending on their security clearance. For example from the point of view of an agent with security clearance *any*, or indeed *kate*, it has the type

$$\mathit{accnt} : \mathsf{loc}\big\{\mathit{deposit}{:}\mathsf{w}\langle\mathsf{int}\rangle\big\}$$

For an agent at level *ruth* it looks like

$$\mathit{accnt} : \mathsf{loc}\big\{\mathit{deposit}{:}\mathsf{w}\langle\mathsf{int}\rangle,\ \ \mathit{withdraw}{:}\mathsf{w}\langle\mathsf{int}\rangle\big\}$$

while from the point of view of the account management software, at level *sys*, it has the type

$$\mathit{accnt} : \mathsf{loc}\big\{\mathit{deposit}{:}\mathsf{r}\langle\mathsf{int}\rangle,\ \ \mathit{withdraw}{:}\mathsf{r}\langle\mathsf{int}\rangle,\ \ \mathit{close}{:}\mathsf{r}\langle\rangle\big\}$$

allowing it to read and process requests.

Of course many networks will not respect a given security policy. In our example, the agent

$$\mathit{kate\text{-}pc}[\![\mathsf{go}\ \mathit{accnt}.\ \mathit{withdraw}!\langle 500\rangle]\!] \qquad (\ddagger)$$

violates the security policy since an agent from *kate-pc* attempts to withdraw \$500 from *ruth*'s account. Note that the violation is not that a low-level agent, emanating from *kate-pc* gains access to the high-level site *accnt*; this is allowed, for example to give access to a low-level resource such as *deposit*. The violation lies in the attempt to access to the higher-level resource *withdraw*, through which, presumably, some money is transferred to *kate-pc*.

Formally we define violation of a security policy Γ as the occurrence of a run-time error in a network N, giving rise to a family of predicates

$$N \xrightarrow{\Gamma} \mathsf{err}\,k$$

This states that in network N, some agent is in violation of the security policy Γ; further, the violation occurs at location k. The question then arises if we can design a typing system which guarantees the absence of such run-time errors, the absence of security violations.

This question is answered positively in Section 3. We design a language of *secure types*, *i.e.* coherent collections of location and resource capabilities annotated with security levels, a subtyping relation between these types, and a typing system for Dπ. From these we obtain the following results:

SUBJECT REDUCTION: Well-typing is preserved by reduction.
TYPE SAFETY: Well-typed networks are incapable of security violations.

Suppose that security policy Γ represents a valid typing environment, and N is well-typed with respect to Γ. Then these results together prove that no agent in N can ever violate the security constraints imposed by Γ. For example, our typing system rejects (‡) with respect to the security policy just outlined above:

$$\Delta = \left\{ \begin{array}{l} \mathit{accnt} : \mathsf{loc}_{\mathit{sys}} \left\{ \begin{array}{rl} \mathit{deposit} : & \mathsf{w}_{\mathit{any}}\langle \mathsf{int} \rangle,\ \mathsf{r}_{\mathit{sys}}\langle \mathsf{int} \rangle \\ \mathit{withdraw} : & \mathsf{w}_{\mathit{ruth}}\langle \mathsf{int} \rangle,\ \mathsf{r}_{\mathit{sys}}\langle \mathsf{int} \rangle \\ \mathit{close} : & \mathsf{w}_{\mathit{man}}\langle\rangle,\ \mathsf{r}_{\mathit{sys}}\langle\rangle \end{array} \right\} \\ \mathit{ruth\text{-}pc} : \mathsf{loc}_{\mathit{ruth}}\{\ldots\} \\ \mathit{kate\text{-}pc} : \mathsf{loc}_{\mathit{kate}}\{\ldots\} \end{array} \right\} \qquad (\dagger)$$

An important and novel aspect of the typing system is that it ensures that incoming agents are executed at an appropriate security level, as determined by the security level of the agent's source location. For example if a location has been assigned the least security level, which in general we denote by $\bot$, then agents emanating from this site will be executed at level $\bot$, meaning intuitively that they are completely untrusted.

1.3 Open Networks

The Subject Reduction and Type Safety theorems cited above ensure that our secure types are sufficient to guarantee secure resource access in *closed* networks. In closed networks, all principals are administered in concert and thus it is possible to type-check all the agents in the network before they are executed. However in open networks, one cannot assume that all agents from remote sites are well-typed. Nevertheless in these networks the security of well-behaved principals, such as *accnt*, must be maintained even in the presence of malicious agents, such as (‡). The major contribution of this paper is the observation that, by introducing security levels into the type language this can be achieved; capability-based typing systems can also be applied to open networks.

In order to prevent the agent (‡) from violating the security guarantees at *accnt*, we modify the rule for agent movement (∗) so that incoming agents are type-checked before they are allowed to run locally:

$$\ell\langle\!\langle \Delta \rangle\!\rangle \mid k[\![\mathsf{go}\,\ell.P]\!] \longmapsto \ell\langle\!\langle \Delta \rangle\!\rangle \mid \ell[\![P]\!] \qquad \text{if } \Delta \Vdash^{k}_{\ell} P$$

Here $\ell\langle\!\langle \Delta \rangle\!\rangle$ is a *filter* for site ℓ, and $\Delta \Vdash^{k}_{\ell} P$ is an instance of the *runtime typing* relation. Filters provide each "good" site with a safe approximation of the global security policy Γ. This runtime typing relation guarantees that:

> If P comes from location k and $\Delta \Vdash^{k}_{\ell} P$, then it is safe to run P at ℓ, assigning it the security level that ℓ assigns to k.

In addition to the move rule, other reduction rules are also modified to reflect the fact that filters may change over time as information about new resources in the network is obtained.

We establish Subject Reduction and Type Safety results for open networks, showing that this form of runtime typechecking offers adequate protection from malicious agents. These results use an extended set of types, called *partial security types* as their use only guarantees type-safe behavior at a subset of sites, the "good" sites. Partial security types are obtained by introducing a new type constructor lbad, along with a typing rule that says:

$$\text{If } \Gamma(\ell) = \mathsf{lbad} \text{ then, for any } P,\ \ell[\![P]\!] \text{ is well-typed with respect to } \Gamma.$$

For example let Δ be as defined in (†). If we take

$$\Delta'(\ell) = \begin{cases} \mathsf{lbad} & \text{if } \ell = \mathit{kate\text{-}pc} \\ \Delta(\ell) & \text{otherwise} \end{cases} \qquad (\S)$$

Table 1 Syntax

X,Y ::=	*Patterns*	u,v,w ::=	*Values*
x	Variable	bv_σ	Base Value
$(X_1,..,X_n)$	Tuple	e	Name
P,Q ::=	*Threads*	x	Variable
$u!\langle v\rangle\,P$	Output	$(u_1,..,u_n)$	Tuple
$u?(X{:}\mathrm{T})\,P$	Input		
$*P$	Replication	M,N ::=	*Networks*
stop	Termination	$\mathbf{0}$	Empty
$P \mid Q$	Composition	$M \mid N$	Composition
$(\nu\, e{:}\mathrm{T})P$	Restriction	$(\nu^\sigma_k\, e{:}\mathrm{T})N$	Restriction
$\mathsf{go}_\sigma\, u.P$	Movement	$k[\![P]\!]_\sigma$	Agent

then (‡) is well-typed with respect to Δ' in the partial typing system.

This means that the partial typing system can be used, even in open networks, to ensure that security violations can occur only at *bad* sites (sites which are assigned the type lbad); the security policy Γ is respected at all *good* sites (sites which are not assigned the type lbad).

2 The Language Dπ with Security Policies

The main syntactic categories of the language, defined in Table 1, are threads P, agents $k[\![P]\!]_\sigma$, and networks N. Agents are threads running at a particular location and with a particular security level. Networks are collections of agents. The syntax is parameterized with respect to the sets *Base*, of base values, ranged over by bv; *Name*, of names, ranged over by *a*–*m*; *Var*, of variables, ranged over by *x*–*z*; and $\mathcal{S}$, of security levels, ranged over by σ, ρ. We require that $\mathcal{S}$ forms a complete lattice, ordered by $\sqsubseteq$, with maximal element $\top$, minimal element $\bot$, greatest-lower-bound operation $\sqcap$, and least-upper-bound operation $\sqcup$. We defer the discussion of types, T, until Section 2.2.

Note that security levels are used to annotate agents, network-level restrictions, the go operator, and base values. Although these annotations can be generated automatically (using standard techniques), they simplify the presentation of the calculus and allow finer control of resource access than is available automatically.

EXAMPLE 2.1. As an example of a network, consider the term:

$$\ell[\![P]\!]_\top \mid (\nu\, k{:}\mathsf{loc}_\top)\big(\ell[\![Q]\!]_\top \mid m[\![R]\!]_\bot\big)$$

This network contains three agents, $\ell[\![P]\!]_\top$, $\ell[\![Q]\!]_\top$ and $k[\![R]\!]_\bot$. The first two agents are running at location ℓ with high security, the third at location m, at low security. Moreover Q and R share knowledge of a private, high security location, k. Since P is outside the scope of k it cannot send subagents to run there. This is possible for both Q and R but the type rules will guarantee that the low-security agent R can only access low-security resources at k. □

NOTATION. We routinely drop type annotations from terms when they are not of interest. We also use standard abbreviations from the π-calculus, *e.g.* dropping final occurrences of stop and writing $(v_1,..,v_n)$ as $\widetilde{v}$. The variables in the pattern X are bound by the input construct $u?(X{:}\mathrm{T})\,P$, the scope is P. The name e is bound by the restrictions $(\nu\, e{:}\mathrm{T})P$ and $(\nu^\sigma_k\, e{:}\mathrm{S})N$, the scopes are P, and N, respectively. The functions $\mathrm{fn}(P)$ and $\mathrm{fv}(P)$ return respectively the sets of free names and free variables occurring in P. In the sequel we identify terms up to renaming of bound names and variables. A term with no free variables is *closed*. We write $P\{\!\{u/x\}\!\}$ to denote the capture-avoiding substitution of u for X in P. □

Table 2 Standard Reduction

(STR-EXTR)	$M \mid (\nu^{\sigma}_{k}\, e{:}\mathrm{T})N \equiv (\nu^{\sigma}_{k}\, e{:}\mathrm{T})(M \mid N)$	if $e \notin \mathrm{fn}(M)$
(STR-GARBAGE$_1$)	$(\nu^{\sigma}_{k}\, e{:}\mathrm{T})\,\mathbf{0} \equiv \mathbf{0}$	
(STR-GARBAGE$_2$)	$k[\![\mathsf{stop}]\!]_{\sigma} \equiv \mathbf{0}$	
(STR-COPY)	$k[\![*P]\!]_{\sigma} \equiv k[\![P]\!]_{\sigma} \mid k[\![*P]\!]_{\sigma}$	
(STR-SPLIT)	$k[\![P \mid Q]\!]_{\sigma} \equiv k[\![P]\!]_{\sigma} \mid k[\![Q]\!]_{\sigma}$	
(STR-NEW)	$k[\![(\nu\, e{:}\mathrm{T})P]\!]_{\sigma} \equiv (\nu^{\sigma}_{k}\, e{:}\mathrm{T})\, k[\![P]\!]_{\sigma}$	
(RED-MOVE)	$k[\![\mathsf{go}_{\rho}\, \ell.P]\!]_{\sigma} \longmapsto \ell[\![P]\!]_{\rho}$	
(RED-COMM)	$k[\![a!\langle v\rangle\, P]\!]_{\sigma} \mid k[\![a?(X)\, Q]\!]_{\rho} \longmapsto k[\![P]\!]_{\sigma} \mid k[\![Q\{^{v}/_{X}\}]\!]_{\rho}$	

2.1 Standard Reduction

The standard semantics is defined using two relations over closed network terms: a structural relation ($M \equiv N$) and a reduction relation ($M \longmapsto M'$). The main reduction relation we are interested in is composed from these two:

$$(\longrightarrow) \stackrel{def}{=} (\equiv \cdot \longmapsto \cdot \equiv)$$

This relation allows for any amount of structural reordering before and after each primary reduction step. The axioms defining these relations, which may be applied in any network context, are given in Table 2.

The structural congruence is defined to be the least equivalence relation that is substitutive over networks,[1] satisfies the standard commutative monoid laws for composition and the axioms given in Table 2. These axioms provide means for the extension of the scope of a name, for garbage collection of unused names and terminated threads, and for the division and replication of agents. The reduction relation $\longmapsto$ is defined to be the least relation that is substitutive over networks and satisfies the reduction axioms of Table 2. The communication axiom (RED-COMM) is adapted from that for the standard π-calculus; note that in the distributed setting, communication can only occur between colocated agents. The rule (RED-MOVE), $k[\![\mathsf{go}_{\rho}\, \ell.P]\!]_{\sigma} \longmapsto \ell[\![P]\!]_{\rho}$, states that an agent running at k with security level σ can move to ℓ, adopting the new security level ρ. This rule is clearly very powerful, as it allows for arbitrary changes of security levels. This power will be tamed by the typing relation, which will guarantee, among other things, that $\rho \sqsubseteq \sigma$.

EXAMPLE 2.2. Consider a network with two agents, one at h, a high security site, and one at ℓ, a low level security site. The agent at h wishes to send a fresh integer channel a, located at h, to the other agent using the channel b, located at ℓ. This network N could be written:

$$\begin{array}{rll}
 & \ell[\![b?(z,x)\, Q]\!]_{\perp} \mid h[\![(\nu\, a{:}\mathrm{A})\,(P \mid \mathsf{go}_{\perp}\, \ell.b!\langle h,a\rangle)]\!]_{\top} & \\
\longrightarrow & \ell[\![b?(z,x)\, Q]\!]_{\perp} \mid (\nu^{\top}_{h}\, a{:}\mathrm{A})\big(h[\![P \mid \mathsf{go}_{\perp}\, \ell.b!\langle h,a\rangle]\!]_{\top}\big) & (\text{RED-NEW}) \\
\equiv & \ell[\![b?(z,x)\, Q]\!]_{\perp} \mid (\nu^{\top}_{h}\, a{:}\mathrm{A})\big(h[\![P]\!]_{\top} \mid h[\![\mathsf{go}_{\perp}\, \ell.b!\langle h,a\rangle]\!]_{\top}\big) & (\text{STR-SPLIT}) \\
\longrightarrow & \ell[\![b?(z,x)\, Q]\!]_{\perp} \mid (\nu^{\top}_{h}\, a{:}\mathrm{A})\big(h[\![P]\!]_{\top} \mid \ell[\![b!\langle h,a\rangle]\!]_{\perp}\big) & (\text{RED-MOVE}) \\
\longrightarrow & (\nu^{\top}_{h}\, a{:}\mathrm{A})\ell[\![Q\{^{h,a}/_{z,x}\}]\!]_{\perp} \mid h[\![P]\!]_{\top} & (\text{STR-EXTR}), (\text{RED-COMM}), (\text{STR-GARBAGE}_2)
\end{array}$$

Beside each reduction, we have written the axioms used to infer it, omitting mention of the monoid laws. An example of a process Q that uses the received value (z,x) is '$\mathsf{go}_{\perp}\, z.x!\langle 1\rangle$', which after the communication becomes '$\mathsf{go}_{\perp}\, h.a!\langle 1\rangle$'.

Apriori it may seem that this computation involves a security leak from the high level site h to the low level site ℓ. However even in well-typed networks we will allow low level sites to acquire certain information about high-level sites; information appropriate to their security status. For example, in order to type N, our typing system will required that a be a low-security resource. □

[1] A relation $\succ$ is *substitutive over networks* if $N \succ N'$ implies $N \mid M \succ N' \mid M$, $M \mid N \succ M \mid N'$, and $(\nu^{\sigma}_{k}\, e{:}\mathrm{T})N \succ (\nu^{\sigma}_{k}\, e{:}\mathrm{T})N'$. The monoid laws are: $M \mid \mathbf{0} \equiv M$, $M \mid N \equiv N \mid M$, and $M \mid (N \mid O) \equiv (M \mid N) \mid O$.

Table 3 Pre-Types and Pre-Capabilities: Syntax and Subtyping

$K, L ::= \mathsf{loc}_\sigma\{\gamma_1, .., \gamma_n\}$				
$A, B ::= \mathsf{res}\{\gamma_1, .., \gamma_n\}$				
$S, T ::=$	*Pre-Types*	*Sub-Types*		
$\mathcal{B}_\sigma$	Base types	$\mathcal{B}_\sigma <: \mathcal{B}_\rho$	if $\sigma \sqsubseteq \rho$	
K	Location types	$\mathsf{loc}_\sigma\{\widetilde{\delta}\} <: \mathsf{loc}_\rho\{\widetilde{\gamma}\}$	if $\rho \sqsubseteq \sigma$ and	$\forall \gamma_j : \exists \delta_i : \delta_i <: \gamma_j$
A	Resource types	$\mathsf{res}\{\widetilde{\delta}\} <: \mathsf{res}\{\widetilde{\gamma}\}$	if	$\forall \gamma_j : \exists \delta_i : \delta_i <: \gamma_j$
$(T_1, .., T_n)$	Tuple types	$\widetilde{S} <: \widetilde{T}$	if $\forall i : S_i <: T_i$	
$K[A_1, .., A_n]$	Dependent types	$K[\widetilde{A}] <: L[\widetilde{B}]$	if $K <: L$ and $\widetilde{A} <: \widetilde{B}$	
$\delta, \gamma ::=$	*Pre-Capabilities*	*Sub-Capabilities*		
a:A	Use resource	a:A $<:$ a:B	if $A <: B$	
x:T	Use value	x:S $<:$ x:T	if $S <: T$	
new_σ	Create resource	$\mathsf{new}_\sigma <: \mathsf{new}_\rho$	if $\sigma \sqsubseteq \rho$	
$\mathsf{w}_\sigma\langle T\rangle$	Write resource	$\mathsf{w}_\sigma\langle S\rangle <: \mathsf{w}_\rho\langle T\rangle$	if $\sigma \sqsubseteq \rho$ and $T <: S$	
$\mathsf{r}_\sigma\langle T\rangle$	Read resource	$\mathsf{r}_\sigma\langle S\rangle <: \mathsf{r}_\rho\langle T\rangle$	if $\sigma \sqsubseteq \rho$ and $S <: T$	

2.2 Security Policies

The formal definition of security policies is given in terms of *pre-types*, T, whose syntax is given in Table 3. For convenience, we introduce separate metavariables for resource and location pre-types, respectively A and K. Their basic structure is taken from [15] to which the reader is referred for a detailed description and rationale. Here the capabilities are annotated with security levels; intuitively an agent running at security level σ can only access capabilities whose security level is at most σ. Briefly:

- Resources may have a *read* capability at a given type T and security level σ, $\mathsf{r}_\sigma\langle T\rangle$.
- Resources may also have a *write* capability at a given type and security level, $\mathsf{w}_\sigma\langle T\rangle$.
- Locations may have the capability to use a typed resource, a:A, or variable, x:T.
- Locations may also have the capability to create a new resource (or location) at security level σ, new_σ.

A location pre-type is then a finite collection of location capabilities; a resource pre-type is a finite collection of resource capabilities. Finally, value pre-types may also include (1) base types at security level σ, $\mathcal{B}_\sigma$; (2) tuples, $(T_1, .., T_n)$; and (3) dependent or *located* tuples, $K[A_1, .., A_n]$. Dependent tuples are used to transmit information about non-local resources; their role is explained in detail in [15].

NOTATION. In examples, we sometimes drop the security-level subscript $\bot$; for example, rendering $\mathsf{int}_\bot$ as int. We also sometimes, as in the Introduction, use the following shorthand for resource types:

$$\mathsf{r}_\sigma\langle T\rangle \stackrel{def}{=} \mathsf{res}\{\mathsf{r}_\sigma\langle T\rangle\} \qquad \mathsf{w}_\sigma\langle T\rangle \stackrel{def}{=} \mathsf{res}\{\mathsf{w}_\sigma\langle T\rangle\} \qquad \mathsf{rw}_\sigma\langle T\rangle \stackrel{def}{=} \mathsf{res}\{\mathsf{w}_\sigma\langle T\rangle, \mathsf{r}_\sigma\langle T\rangle\} \qquad \square$$

Pre-types are used in the typing system, which relies crucially on a *subtyping relation* ($<:$), also given in Table 3. For capabilities, $\delta <: \gamma$ means, approximately, that γ is a more restrictive than δ, and for types $S <: T$ means T contains fewer capabilities than S. This captures the intuition that it is always safe to restrict the set of capabilities, but not to expand it. In this definition we call attention to the contravariance of both the write capability on resources and the security levels on locations.

DEFINITION 2.3. A *security policy* Γ is a partial map from names to closed location pre-types. $\square$

There are numerous ways in which a given security policy can be violated in a given network. In Table 4 we collect these security violations as runtime errors. Formally we define a family of predicates $M \xrightarrow{\Gamma} \mathsf{err}\, k$, which may be read: "$M$ violates security policy Γ at location k". Two of the

Table 4 Runtime Error with respect to a security policy

(ERR-W)	$k[\![a!\langle v\rangle P]\!]_\sigma$	$\xrightarrow{\Gamma}$ err k	if $\Gamma(k) \not<: \mathsf{loc}_\sigma\{a{:}\mathsf{res}\{\mathsf{w}_\sigma\langle \mathrm{T}\rangle\}\}$ (all T) or $\Gamma(k) <: \mathsf{loc}_\sigma\{a{:}\mathsf{res}\{\mathsf{w}_\sigma\langle \mathrm{T}\rangle\}\}$ and $\Gamma \sqcap \{_k v{:}\mathrm{T}\}$ undef
(ERR-R)	$k[\![a?(X{:}\mathrm{T})P]\!]_\sigma$	$\xrightarrow{\Gamma}$ err k	if $\Gamma(k) \not<: \mathsf{loc}_\sigma\{a{:}\mathsf{res}\{\mathsf{r}_\sigma\langle \mathrm{T}\rangle\}\}$ or $\mathrm{T} \notin \mathit{Type}_\sigma$
(ERR-GO)	$k[\![\mathsf{go}_\rho\, \ell.P]\!]_\sigma$	$\xrightarrow{\Gamma}$ err k	if $\rho \not\sqsubseteq \sigma$
(ERR-NEW)	$(\nu_k^\sigma e{:}\mathrm{T})N$	$\xrightarrow{\Gamma}$ err k	if $\Gamma(k) \not<: \mathsf{loc}_\sigma\{\mathsf{new}_\sigma\}$
(ERR-STR$_1$)	$(\nu_k^\sigma e{:}\mathrm{T})N$	$\xrightarrow{\Gamma}$ err $\ell\{^k/_e\}$	if $N \xrightarrow{\Gamma\sqcap\{_k e:\mathrm{T}\}}$ err ℓ $(e \notin \mathrm{fn}(\Gamma))$
(ERR-STR$_2$)	$M \mid N$	$\xrightarrow{\Gamma}$ err k	if $M \xrightarrow{\Gamma}$ err k or $N \xrightarrow{\Gamma}$ err k

most important clauses in the definition are the following. (1) If ρ is not dominated by the current security level σ (*i.e.* $\rho \not\sqsubseteq \sigma$), then the agent $k[\![\mathsf{go}_\rho\, \ell.P]\!]_\sigma$ violates every security policy. (2) The agent $k[\![a!\langle v\rangle P]\!]_\sigma$ violates any security policy Γ that

- does not allow σ-threads (*i.e.* threads with security level σ) to run at k, *i.e.* $\Gamma(k) \not<: \mathsf{loc}_\sigma$,
- does not allow σ-threads write access to a, *i.e.* for all T, $\Gamma(k) \not<: \mathsf{loc}\{a{:}\mathsf{w}_\sigma\langle \mathrm{T}\rangle\}$, or
- does not allow transmission of v on a, *i.e.* $\Gamma(k) <: \mathsf{loc}\{a{:}\mathsf{w}\langle \mathrm{T}\rangle\}$, but v cannot be assigned T.

Note that the last case is the crucial one; it prohibits high-security data from being transmitted on low-security resources.

3 Typing

In this section we design a typing system which guarantees no violations of "coherent" security policies, essentially those which define *type environments*.

3.1 Types

We first define the collection of well-formed types and to do so it is convenient to divide them into *kinds*, $\mathcal{K}$, depending upon their security level:

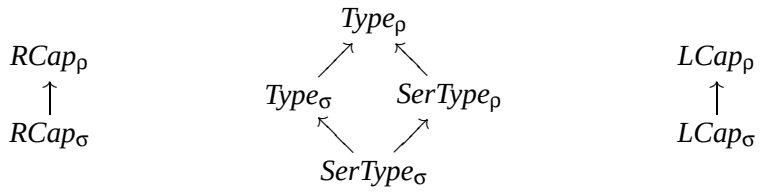
$\mathcal{K}$ ::=	Type_σ	Values accessible at security level σ (and above)
|	$\mathit{SerType}_\sigma$	Values accessible at security level σ, transferable between sites
|	RCap_σ	Resource capabilities accessible at security level σ
|	LCap_σ	Location capabilities accessible at security level σ

Intuitively if $\mathrm{T} \in \mathit{Type}_\sigma$, then values of type T are accessible at security level σ or higher; they may not be available at lower security levels.

Serializable types (*SerType*) are introduced for values whose interpretation is location independent and thus may be transmitted from site to site. For example the dependent value $(k,a){:}\mathrm{K}[\mathrm{A}]$ is serializable, as it can be interpreted anywhere in a system, whereas $(k,a){:}(\mathrm{K},\mathrm{A})$ is not; here k and a are independent and therefore the actual location of the resource a is unknown to the receiver.

The formation rules for types, taking the form $\mathrm{T} \in \mathcal{K}$ where T is a pre-type, are given in Table 5. The formation rules rely on three other relations:

- A well-formedness relation for capabilities, also given in Table 5.
- A *subkinding* relation between kinds, $\mathcal{K} <: \mathcal{K}'$. Intuitively subkinding is generated by the following diagram, where $\sigma \sqsubseteq \rho$ and $\mathcal{K} <: \mathcal{K}'$ is represented as $\mathcal{K} \to \mathcal{K}'$:



- A *compatibility* relation between capabilities, $\gamma \asymp \delta$, given in Table 6.

Table 5 Types and Capabilities

(TYPE-BASE)	$\mathcal{B}_\sigma \in \mathit{SerType}_\sigma$	
(TYPE-LOC)	$\mathsf{loc}_\sigma\{\widetilde{\gamma}\} \in \mathit{SerType}_\sigma$	if $\forall i\colon \gamma_i \in \mathit{LCap}_\sigma$ and $\forall i,j\colon \gamma_i \asymp \gamma_j$
(TYPE-RES)	$\mathsf{res}\{\widetilde{\gamma}\} \in \mathit{Type}_\sigma$	if $\forall i\colon \gamma_i \in \mathit{RCap}_\sigma$ and $\forall i,j\colon \gamma_i \asymp \gamma_j$
(TYPE-TUP)	$(\mathrm{T}_1, .., \mathrm{T}_n) \in \mathcal{K}$	if $\forall i\colon \mathrm{T}_i \in \mathcal{K}$
(TYPE-DEP)	$\mathrm{K}[\mathrm{A}_1, .., \mathrm{A}_n] \in \mathit{SerType}_\sigma$	if $\mathrm{K} \in \mathit{SerType}_\sigma$ and $\widetilde{\mathrm{A}} \in \mathit{Type}_\sigma$
(TYPE-SUB)	$\mathrm{T} \in \mathcal{K}'$	if $\mathrm{T} \in \mathcal{K}$ and $\mathcal{K} <: \mathcal{K}'$
(CAP-SUB)	$\gamma \in \mathcal{K}'$	if $\gamma \in \mathcal{K}$ and $\mathcal{K} <: \mathcal{K}'$
(CAP-NEW)	$\mathsf{new}_\sigma \in \mathit{LCap}_\sigma$	
(CAP-RES)	$u{:}\mathrm{T} \in \mathit{LCap}_\sigma$	if $\mathrm{T} \in \mathit{Type}_\sigma$
(CAP-W)	$\mathsf{w}_\sigma\langle \mathrm{T} \rangle \in \mathit{RCap}_\sigma$	if $\mathrm{T} \in \mathit{Type}_\sigma$
(CAP-R)	$\mathsf{r}_\sigma\langle \mathrm{T} \rangle \in \mathit{RCap}_\sigma$	if $\mathrm{T} \in \mathit{Type}_\sigma$

Table 6 Compatibility and Environment Generation

$$\frac{u \neq v}{u{:}\mathrm{S} \asymp v{:}\mathrm{T}}\ (\text{COMPAT-RES}) \qquad \frac{\gamma \neq \mathsf{new}_\vartheta}{\mathsf{new}_\sigma \asymp \gamma}\ (\text{COMPAT-NEW}) \qquad \frac{\mathrm{S} <: \mathrm{T}}{\mathsf{w}_\sigma\langle \mathrm{S} \rangle \asymp \mathsf{r}_\rho\langle \mathrm{T} \rangle}\ (\text{COMPAT-RW})$$

$$\frac{}{\mathsf{r}_\sigma\langle \mathrm{S} \rangle \asymp \mathsf{r}_\rho\langle \mathrm{T} \rangle}\ (\text{COMPAT-RR}) \qquad \frac{}{\mathsf{w}_\sigma\langle \mathrm{S} \rangle \asymp \mathsf{w}_\rho\langle \mathrm{T} \rangle}\ (\text{COMPAT-WW}) \qquad \frac{\delta \asymp \gamma}{\gamma \asymp \delta}\ (\text{COMPAT-SYM})$$

$$\frac{\mathrm{K} \in \mathit{SerType}}{\{{}_w k{:}\mathrm{K}\} = \{k{:}\mathrm{K}\}}\ (\text{GENENV-LOC}) \qquad \frac{\mathrm{T} \in \mathit{SerType}}{\{{}_w x{:}\mathrm{T}\} = \{x{:}\mathrm{T}\}}\ (\text{GENENV-UNIV}) \qquad \frac{\mathrm{A} \in \mathit{Type}}{\{{}_w a{:}\mathrm{A}\} = \{w{:}\mathsf{loc}_\perp\{a{:}\mathrm{A}\}\}}\ (\text{GENENV-RES})$$

$$\frac{\mathrm{T} \in \mathit{Type} \qquad \mathrm{T} \notin \mathit{SerType}}{\{{}_w x{:}\mathrm{T}\} = \{w{:}\mathsf{loc}_\perp\{x{:}\mathrm{T}\}\}}\ (\text{GENENV-LOCAL})$$

$$\frac{\sigma \sqsubseteq \rho}{\{{}_w \mathsf{bv}_\sigma{:}\mathcal{B}_\rho\} = \varnothing}\ (\text{GENENV-BASE}) \qquad \frac{\forall i\colon \{{}_w v_i{:}\mathrm{T}_i\} = \Delta_i}{\{{}_w(v_1, .., v_n){:}(\mathrm{T}_1, .., \mathrm{T}_n)\} = \Delta_1 \sqcap \cdots \sqcap \Delta_n}\ (\text{GENENV-TUP})$$

$$\frac{\{{}_w u{:}\mathrm{K}\} = \Delta_u \qquad \{{}_u \widetilde{v}{:}\widetilde{\mathrm{A}}\} = \Delta_v}{\{{}_w(u, \widetilde{v}){:}\mathrm{K}[\widetilde{\mathrm{A}}]\} = \Delta_u \sqcap \Delta_v}\ (\text{GENENV-DEP})$$

The formation rules for types are quite subtle. Consider, for example, the two point lattice of security levels $\perp \sqsubset \top$. This defines two kinds of types: "insecure" types $\mathit{Type}_\perp$ and "secure" types $\mathit{Type}_\top \supseteq \mathit{Type}_\perp$. Insecure agents are required to use types in $\mathit{Type}_\perp$, whereas secure agents are also allowed to use the additional types in $\mathit{Type}_\top$. We have:

$$\begin{array}{lll} \text{(a)} & \mathsf{int}_\perp \in \mathit{Type}_\top & \mathsf{int}_\top \notin \mathit{Type}_\perp \\ \text{(b)} & \mathsf{w}_\perp\langle\rangle \in \mathit{RCap}_\top & \mathsf{w}_\top\langle\rangle \notin \mathit{RCap}_\perp \\ \text{(c)} & \mathsf{loc}_\perp \in \mathit{Type}_\top & \mathsf{loc}_\top \notin \mathit{Type}_\perp \\ \text{(d)} & \mathsf{new}_\perp \in \mathit{LCap}_\top & \mathsf{new}_\top \notin \mathit{LCap}_\perp \end{array}$$

Insecure integers are accessible to secure agents, but secure integers are not accessible to insecure agents (a). Similar constraints apply to resources (b) and locations (c). Finally, secure agents can be trusted to create new insecure locations and to relate information about them, but *insecure* agents should *not* be trusted to create *secure* sites or relate information about them (d).

Finally note that there is no relation between subtyping and accessibility at a given security level. For example we have:

$$\begin{array}{lll} \mathit{Type}_\perp \not\ni & \mathsf{res}\{\mathsf{w}_\top\langle\rangle, \mathsf{r}_\perp\langle\rangle\} <: \mathsf{r}_\perp\langle\rangle & \in \mathit{Type}_\perp \\ \mathit{Type}_\perp \ni & \mathsf{r}_\perp\langle\rangle <: \mathsf{r}_\top\langle\rangle & \notin \mathit{Type}_\perp \\ \mathit{Type}_\perp \not\ni & \mathsf{loc}_\top <: \mathsf{loc}_\perp & \in \mathit{Type}_\perp \\ \mathit{Type}_\perp \ni & \mathsf{int}_\perp <: \mathsf{int}_\top & \notin \mathit{Type}_\perp \end{array}$$

PROPOSITION 3.1. *Type is a preorder with respect to* $<:$*, with a partial meet operation* $\sqcap$.

Proof. The proof requires that security levels have both a meet and a join. For example:

$$\mathsf{loc}_\sigma\{\mathsf{new}_{\sigma'}\} \sqcap \mathsf{loc}_\rho\{\mathsf{new}_{\rho'}\} = \mathsf{loc}_{\sigma \sqcup \rho}\{\mathsf{new}_{\sigma' \sqcap \rho'}\}$$ □

Table 7 Standard Typing

$$\frac{\mathrm{T} \in \mathit{Type}_\sigma \quad \Gamma(u) <: \mathrm{T}}{\Gamma \Vdash^\sigma_w u{:}\mathrm{T}} \text{(VAL-UNIV)} \qquad \frac{\mathrm{T} \in \mathit{Type}_\sigma \quad \Gamma(w) <: \mathsf{loc}\{u{:}\mathrm{T}\}}{\Gamma \Vdash^\sigma_w u{:}\mathrm{T}} \text{(VAL-LOCAL)} \qquad \frac{\rho \sqsubseteq \rho' \sqsubseteq \sigma \quad \mathsf{bv} \in \mathsf{vals}(\mathcal{B})}{\Gamma \Vdash^\sigma_w \mathsf{bv}_\rho{:}\mathcal{B}_{\rho'}} \text{(VAL-BASE)}$$

$$\frac{\Gamma \Vdash^\sigma_w u_i{:}\mathrm{T}_i \ \ (\forall i)}{\Gamma \Vdash^\sigma_w \widetilde{u}{:}\widetilde{\mathrm{T}}} \text{(VAL-TUP)} \qquad \frac{\Gamma \Vdash^\sigma_w u{:}\mathrm{K} \quad \Gamma \Vdash^\sigma_u \widetilde{v}{:}\widetilde{\mathrm{A}}}{\Gamma \Vdash^\sigma_w (u,\widetilde{v}){:}\mathrm{K}[\widetilde{\mathrm{A}}]} \text{(VAL-DEP)}$$

$$\frac{\Gamma \Vdash^\sigma_w u{:}\mathsf{loc}_\rho \quad \Gamma \Vdash^\rho_u P}{\Gamma \Vdash^\sigma_w \mathsf{go}_\rho\, u.P} \text{(TRD-GO)} \qquad \frac{\Gamma \Vdash^\sigma_w P \quad \Gamma \Vdash^\sigma_w Q}{\Gamma \Vdash^\sigma_w \mathsf{stop}, P \mid Q, *P} \text{(TRD-STR)}$$

$$\frac{\Gamma \Vdash^\sigma_w w{:}\mathsf{loc}\{\mathsf{new}_\sigma\} \quad e \notin \mathrm{fn}(\Gamma),\ \mathrm{T} \neq \mathsf{lbad} \quad \mathrm{T} \in \mathit{Type}_\sigma \quad \Gamma \sqcap \{_w e{:}\mathrm{T}\} \Vdash^\sigma_w P}{\Gamma \Vdash^\sigma_w (\nu e{:}\mathrm{T})P} \text{(TRD-NEW)}$$

$$\frac{\Gamma \Vdash^\sigma_w u{:}\mathsf{res}\{\mathsf{w}_\sigma\langle \mathrm{T}\rangle\} \quad \Gamma \Vdash^\sigma_w v{:}\mathrm{T} \quad \Gamma \Vdash^\sigma_w P}{\Gamma \Vdash^\sigma_w u!\langle v\rangle P} \text{(TRD-W)} \qquad \frac{\Gamma \Vdash^\sigma_w u{:}\mathsf{res}\{\mathsf{r}_\sigma\langle \mathrm{T}\rangle\} \quad \mathrm{fv}(X) \text{ disjoint } \mathrm{fv}(\Gamma) \quad \Gamma \sqcap \{_w X{:}\mathrm{T}\} \Vdash^\sigma_w Q}{\Gamma \Vdash^\sigma_w u?(X{:}\mathrm{T})\,Q} \text{(TRD-R)}$$

$$\frac{\Gamma(k) <: \mathsf{loc}_\sigma \quad \Gamma \Vdash^\sigma_k P}{\Gamma \vdash k[\![P]\!]_\sigma} \text{(NET-RUN)} \qquad \frac{\Gamma \vdash M \quad \Gamma \vdash N}{\Gamma \vdash \mathbf{0}, M \mid N} \text{(NET-STR)}$$

$$\frac{\Gamma(k) <: \mathsf{loc}\{\mathsf{new}_\sigma\} \quad e \notin \mathrm{fn}(\Gamma),\ \mathrm{T} \neq \mathsf{lbad} \quad \mathrm{T} \in \mathit{Type}_\sigma \quad \Gamma \sqcap \{_k e{:}\mathrm{T}\} \vdash N}{\Gamma \vdash (\nu^\sigma_k e{:}\mathrm{T})N} \text{(NET-NEW)}$$

3.2 Typing

We say that a security policy Γ is a *type environment*, if every pre-type in the range is a serializable type, *i.e.* $\forall u \in \mathrm{dom}(\Gamma)\colon \Gamma(u) \in \mathit{SerType}$. These are used in the standard typing system, given in Table 7. The main judgments take the form

$$\Gamma \vdash N$$

where Γ is a type environment and N is a network.[2] In this extended abstract we only briefly explain some of the typing rules. At the network level we can conclude $\Gamma \vdash k[\![P]\!]_\sigma$, if the agent P can be typed to run at location k at security level σ. Thus for agents, typing is with respect to both a location and a security level, giving rise to type judgments of the form

$$\Gamma \Vdash^\sigma_k P$$

Inferring judgments of this form is slightly more complicated. For example $\Gamma \Vdash^\sigma_w \mathsf{go}_\rho\, u.P$ depends on being able to establish that P can run at location u at the appropriate security level, $\Gamma \Vdash^\rho_u P$, and that u does indeed represent a location where activity at this security level is allowed. The latter requires a third form of judgments, for values:

$$\Gamma \Vdash^\sigma_w v{:}\mathrm{T}$$

Intuitively, if $\Gamma \Vdash^\sigma_w v{:}\mathrm{T}$ then value v is accessible to σ-threads at location w (at type T). The particular value judgment required when typing $\mathsf{go}_\rho\, u.P$ is $\Gamma \Vdash^\sigma_w u{:}\mathsf{loc}_\rho$.

The rules for input and name restriction require notation for extensions to type environments. For example to conclude $\Gamma \Vdash^\sigma_w u?(X{:}\mathrm{T})\,Q$ we need to establish that Q is well-typed in an environment constructed by extending Γ with appropriate associations between the variables in X and the type T. To accomplish this, in Table 6 we define a notation for constructing simple environments, which uses the meet operator on types. The reader is referred to page 10 of [28] where the notation is discussed in detail.[3] The main results of this section can now be stated:

THEOREM 3.2 (SUBJECT REDUCTION). *If* $\Gamma \vdash N$ *and* $N \longrightarrow N'$ *then* $\Gamma \vdash N'$. □

[2] Indeed throughout Table 7 only type environments (and not the more general security policies) may be used.

[3] In general this constructs security policies rather than type environments but one can show that all applications of this construction in Tables 7 and 8 actually generate type environments.

THEOREM 3.3 (TYPE SAFETY). *If* $\Gamma \vdash N$ *then for every location* k, $N \xrightarrow{\Gamma}\!\!\!\!\!\!\rightarrow \mathsf{err}\, k$. □

Crucial to the proof of Theorem 3.2, are the following standard properties, adapted to our setting:

$$\text{If } \mathrm{S} \in \mathit{Type}_\sigma,\ \mathrm{T} \in \mathit{Type}_\rho,\ \Gamma \Vdash^\sigma_w v{:}\mathrm{S} \text{ and } \mathrm{S} <: \mathrm{T}, \text{ then } \Gamma \Vdash^\rho_w v{:}\mathrm{T} \qquad \text{(SPECIALIZATION)}$$

$$\text{If } \Gamma \Vdash^\sigma_w P \text{ and } \Delta <: \Gamma \text{ then } \Delta \Vdash^\sigma_w P \qquad \text{(WEAKENING)}$$

$$\text{If } \Gamma \Vdash^\sigma_u v{:}\mathrm{T} \text{ and } \Gamma \sqcap \{{}_u X{:}\mathrm{T}\} \Vdash^\sigma_w P \text{ then } \Gamma \Vdash^\sigma_{w\{v/x\}} P\{\!\{v/x\}\!\} \qquad \text{(SUBSTITUTION)}$$

EXAMPLE 3.4. Let us now reexamine network N, given in Example 2.2. Suppose Γ is a type environment which associates with h and ℓ the types $\mathsf{loc}_\top\{\mathsf{new}_\top\}$ and $\mathsf{loc}_\bot\{b{:}\mathsf{rw}_\bot\langle\mathsf{loc}_\bot\{\mathrm{A}\}\rangle\}$ respectively, where A is the proposed type of the resource to be generated at the high-security site h. Then the ability to type the network N under Γ depends essentially on the security level used in the type A. The reader may check that, for example, with A equal to $\mathsf{rw}_\bot\langle\mathsf{int}\rangle$ then N can be typed; an essential feature in the typing system is the contravariance of location types with respect to security levels. On the other hand if A is $\mathsf{rw}_\top\langle\mathsf{int}\rangle$ then $\Gamma \vdash N$ cannot be derived. □

4 Security in Open Systems

In this section we briefly outline how security types can be applied in the runtime typechecking of mobile agents and thereby offer protection from malicious behavior.

In order to formalize the notion that some sites are untyped, we introduce a new location type, lbad, into the type language. We call the resulting types *partial* security types. Intuitively, a site with type lbad is untyped, and may therefore harbor agents which do not respect the security policies in place. Location pre-types are now defined:

$$\mathrm{K},\mathrm{L} ::= \mathsf{loc}_\sigma\{\widetilde{a}{:}\widetilde{\mathrm{A}},\widetilde{x}{:}\widetilde{\mathrm{B}}\} \mid \mathsf{lbad}$$

Given a security topology, we must chose which security classifications are outside our control. We do this using the predicate ours. This is subject to the constraint:

$$\sigma \sqsubseteq \rho \text{ and } \mathrm{ours}(\sigma) \text{ implies } \mathrm{ours}(\rho)$$

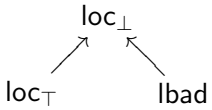
We also extend the definition of types (Table 5) to include:

$$\mathsf{lbad} \in \mathit{SerType}_\bot$$

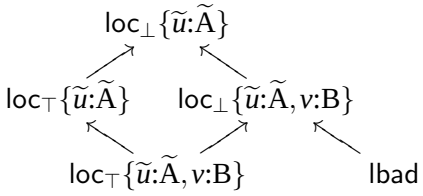
The subtype relation is extended to partial pre-types by adding the following subtyping rule:

$$\mathsf{lbad} <: \mathsf{loc}_\sigma\{\widetilde{u}{:}\widetilde{\mathrm{A}}\} \text{ if } \text{not } \mathrm{ours}(\sigma)$$

This allows untyped locations to access the least secure resources on at a site, but no others. If we consider the two-point lattice of security levels, the subtyping relation is generated by:



or more generally:



The reader familiar with [28] will note that the subtyping relation presented here is very similar to that of "networks with trust" in our prior work. Here we have elaborated on the notion of "trusted type", allowing varying degrees of trust, represented by the non-minimal security levels. If the filter for k records ℓ as a location with security level properly greater than $\bot$, then ℓ must be well-typed, and therefore k can "trust" ℓ.

4.1 Runtime Typechecking

To accomplish runtime typechecking, it is necessary to add type information to running networks. Following [28] we do this by adding a *filter* $k\langle\!\langle\Delta\rangle\!\rangle$ for each location k in a network. The filter includes a type environment Δ which gives k's current view of the resources in the network. Suppose that in a network N, location k knows that there is resource named a of type A at location ℓ. This intuition is captured by requiring that N have a subterm $k\langle\!\langle\Delta\rangle\!\rangle$ such that $\Delta(\ell) <: \mathsf{loc}\{a{:}\mathrm{A}\}$.

Formally, we extend the syntax of networks (Table 1) to include filters, as follows:

$$N ::= \ldots \mid k\langle\!\langle\Delta\rangle\!\rangle$$

We say that a term $k\langle\!\langle\Delta\rangle\!\rangle$ is a *filter for* k and we assume that in well-formed networks every location has associated with it exactly one filter.

In Table 8 we modify the reduction semantics to take advantage of the presence of filters. The purpose of filters is to check that incoming agents are well-typed and thus the main change to the semantics is replace the migration rule (RED-MOVE) with:

$$\ell\langle\!\langle\Delta\rangle\!\rangle \mid k[\![\mathsf{go}\,\ell.P]\!] \longmapsto \ell\langle\!\langle\Delta\rangle\!\rangle \mid \ell[\![P]\!] \qquad \text{if } \Delta(k) <: \mathsf{loc}_\rho \text{ and } (\mathrm{ours}(\rho) \text{ or } \Delta \Vdash^{\rho k}_{\ell} P)$$

Here $\Delta \Vdash^{\rho k}_{\ell} P$ is a *runtime typing relation*, which intuitively says that P is well-formed to move to location ℓ at security level ρ, if originating from k.

The rules for runtime typing with respect to the resulting partial types are also given in Table 8. These extend the static typing rules of Section 3, but there are innovations. The first rule, $(\mathrm{VAL_R\text{-}SLF_1})$ is quite subtle; its soundness depends on the definition of the subtyping relation over partial types. Essentially it allows an incoming agent to refer to its source location k and to associate the minimum security level to it, regardless of whether the filter environment Δ contains any information about k. The rule $(\mathrm{VAL_R\text{-}SLF_2})$ allows an incoming agent to refer to resources at its source, as long as these are presented with minimal security. The rules $(\mathrm{TRD_R\text{-}BAD})$ and $(\mathrm{TRD_R\text{-}RET})$ allow agents to have arbitrary behavior, in some cases, when moving to other sites.

4.2 The protection of good sites

Here we give a very brief account of the argument which states that this scheme of runtime typechecking offers security protection to good sites. First the notion of "good site" is formalized in terms of a static typing relation which uses partial security types: $\Gamma \vdash N$. This relation is defined by extending the standard typing system Table 7 with the rules in Table 8 to handle filters and lbad. Note that while the static typing system interprets these rules with respect to an omniscient authority (Γ), the runtime typing system interprets these rules with respect to the knowledge contained in a filter (Δ, where $\Gamma <: \Delta$). Whereas untypability with respect to Γ indicates that a network is malformed, untypability with respect to a filter Δ may simply indicate that the filter has insufficient information to determine whether an agent is malicious or not.

Intuitively if $\Gamma \vdash N$ and $\Gamma(\ell) \neq \mathsf{lbad}$ then ℓ is a good site. It is this intuitive interpretation of static typing which makes the following theorems interesting. They may be read as saying good sites are protected against malicious behavior.

THEOREM 4.1. *In the partial typing system, if* $\Gamma \vdash N$ *and* $N \longrightarrow N'$ *then* $\Gamma \vdash N'$. □

THEOREM 4.2. *In the partial typing system, if* $\Gamma \vdash N$ *and* $\Gamma(k) \neq \mathsf{lbad}$ *then* $N \xrightarrow{\Gamma}\!\!\!\!\!\!\rightarrow \mathsf{err}\,k$. □

5 Related Work

The language Dπ, a distributed version of the π-calculus [20], was introduced in [29] and simplified and improved in [15]. The secure type system presented here may be viewed as an extension to two previous type systems we have explored. The first, in [15], allows the selective distribution of access rights to resources in a closed network, *i.e.* networks in which all agents are assumed to be well-behaved. (This type system uses an extension of the types from [27]; related type systems, for

Table 8 Semantics of Open Networks

Reduction axioms:

$$(\text{RED}_\text{F}\text{-MOVE}) \quad \begin{array}{ll} & k[\![\text{go}\,\ell.\,P]\!]_\sigma \mid \ell\langle\!\langle\Delta\rangle\!\rangle \\ \longmapsto & \ell[\![P]\!]_\sigma \mid \ell\langle\!\langle\Delta\rangle\!\rangle \quad \text{if } \Delta(k) <: \mathsf{loc}_\rho \text{ and } (\text{ours}(\rho) \text{ or } \Delta \Vdash^{\rho k}_{\ell} P) \end{array}$$

$$(\text{RED}_\text{F}\text{-COMM}) \quad \begin{array}{ll} & k[\![a!\langle v\rangle\,P]\!]_\sigma \mid k[\![a?(X{:}\mathrm{T})\,Q]\!]_\sigma \mid k\langle\!\langle\Delta\rangle\!\rangle \\ \longmapsto & k[\![P]\!]_\sigma \mid k[\![Q\{^{v}/_{X}\}]\!]_\sigma \mid k\langle\!\langle\Delta \sqcap \{_k v{:}\mathrm{T}\}\rangle\!\rangle \end{array}$$

Structural axioms: all rules but (STR-NEW) from Table 2

$$(\text{STR}_\text{F}\text{-NEWR}) \quad \begin{array}{l} k[\![(\nu\, a{:}\mathrm{A})P]\!]_\sigma \mid k\langle\!\langle\Delta\rangle\!\rangle \\ \equiv (\nu^{\sigma}_{k}\, a{:}\mathrm{A})\,\big(k[\![P]\!]_\sigma \mid k\langle\!\langle\Delta \sqcap \{_k a{:}\mathrm{A}\}\rangle\!\rangle\big) \quad \text{if } a \notin \text{fn}(\Delta) \end{array}$$

$$(\text{STR}_\text{F}\text{-NEWL}) \quad \begin{array}{l} k[\![(\nu\, \ell{:}\mathrm{L})P]\!]_\sigma \mid k\langle\!\langle\Delta\rangle\!\rangle \\ \equiv\ (\nu^{\sigma}_{k}\, \ell{:}\mathrm{L})\big(k[\![P]\!]_\sigma \mid k\langle\!\langle\Delta \sqcap \{\ell{:}\mathrm{L}\}\rangle\!\rangle \mid \ell\langle\!\langle\{\ell{:}\mathrm{L}\}\rangle\!\rangle\big) \quad \text{if } \ell \notin \text{fn}(\Delta) \cup \{k\} \end{array}$$

Static typing: all rules from Table 7

$$\frac{\Gamma(k) = \mathsf{lbad}}{\Gamma \vdash k\langle\!\langle\Delta\rangle\!\rangle}\ (\text{NET}_\text{F}\text{-FILT}_\text{B}) \qquad \frac{\Gamma <: \Delta \quad \Gamma(k) = \Delta(k)}{\Gamma \vdash k\langle\!\langle\Delta\rangle\!\rangle}\ (\text{NET}_\text{F}\text{-FILT}_\text{G}) \qquad \frac{\Gamma(w) = \mathsf{lbad}}{\Gamma \vdash^{\sigma}_{w} P}\ (\text{TRD}_\text{F}\text{-BAD})$$

$$\frac{\Gamma(k) = \mathsf{lbad} \quad \ell \notin \text{fn}(\Gamma) \quad \Gamma \sqcap \{\ell{:}\mathsf{lbad}\} \vdash N}{\Gamma \vdash (\nu^{\sigma}_{k}\, \ell{:}\mathrm{L})N}\ (\text{NET}_\text{F}\text{-BAD})$$

Runtime typing: all rules from Table 7, '$\Vdash^{\sigma k}_{w}$' replacing '$\vdash^{\sigma}_{w}$'

$$\frac{\mathrm{K} \in \mathit{SerType}_\sigma \quad \mathsf{lbad} <: \mathrm{K}}{\Delta \Vdash^{\sigma k}_{w} k{:}\mathrm{K}}\ (\text{VAL}_\text{R}\text{-SLF}_1) \qquad \frac{\mathrm{A} \in \mathit{Type}_\sigma}{\Delta \Vdash^{\sigma k}_{k} a{:}\mathrm{A}}\ (\text{VAL}_\text{R}\text{-SLF}_2) \qquad \frac{\Delta(u) = \mathsf{lbad}}{\Delta \Vdash^{\sigma k}_{w} \text{go}_\rho\, u.\,P}\ (\text{TRD}_\text{R}\text{-BAD})$$

$$\frac{}{\Delta \Vdash^{\sigma k}_{w} \text{go}_\rho\, k.\,P}\ (\text{TRD}_\text{R}\text{-RET})$$

somewhat different languages, may be found in [8, 9, 31].) The second is the partial typing system from [28], for use in open systems containing possibly malicious agents; it ensures well-typed behavior at *good* sites. Here we have addressed the issue of selective distribution of access rights in similar systems; distributed networks which harbor malicious agents. Our solution involves the use of security levels, associated with both locations and resources. We design a capability based typing system in which specific capabilities have associated with them a minimum security level. Our formal results imply that

- well-behaved sites can indeed selectively distribute local access rights to principals throughout the network, and
- this distribution will be respected without compromising local resources, even in the presence of malicious agents.

Several studies have addressed the issue of static typing for languages with remote resources [25, 5, 30, 17], but none of these address secure resources or open networks. Proof carrying code and related techniques [37, 22, 18, 21] address the problem of runtime typechecking in open networks, but do not consider either secure resources or references to remote resources.

Type systems have been used to study secure transmission of data, using cryptographic techniques [2, 1, 7]; while the goals of this work are related to ours, there appears to be little technical overlap. Recently, it has also been shown that secure dataflow analysis [10, 11] can be expressed using typing rules [26, 36, 33, 19, 14].

Finally, we note that several calculi, in addition to Dπ, have been proposed for the study of distributed computing [12, 3, 4, 30, 32, 35, 24]. While there is an active interest in typing systems for such languages [13, 6, 9], we are aware of no other work on capability-based or secure typing systems applicable to open systems.

References

[1] M. Abadi. Secrecy by typing in security protocols. In *Proceedings of TACS97*, volume 1218 of *Lecture Notes in Computer Science*, pages 611–637. Springer-Verlag, 1997.

[2] M. Abadi and A. D. Gordon. A calculus for cryptographic protocols: The spi calculus. *Information and Computation*, To appear. Available as Compaq SRC Research Report 149 (1998).

[3] Roberto Amadio. An asynchronous model of locality, failure, and process mobility. In *COORDINATION '97*, volume 1282 of *Lecture Notes in Computer Science*. Springer-Verlag, 1997.

[4] L. Cardelli and A. D. Gordon. Mobile ambients. In Maurice Nivat, editor, *Proc. FOSSACS'98, International Conference on Foundations of Software Science and Computation Structures*, volume 1378 of *Lecture Notes in Computer Science*, pages 140–155. Springer-Verlag, 1998.

[5] Luca Cardelli. A language with distributed scope. *Computing Systems*, 8(1):27–59, January 1995. A preliminary version appeared in Proceedings of the 22nd ACM Symposium on Principles of Programming.

[6] Luca Cardelli and Andrew Gordon. Types for mobile ambients. In *Conference Record of the ACM Symposium on Principles of Programming Languages*, San Antonio, January 1999. ACM Press.

[7] Mads Dam. Proving trust in systems of second-order processes. In *Hawaii International Conference on Systems Science*. IEEE Computer Society Press, 1998.

[8] R. De Nicola, G.-L. Ferrari, and R. Pugliese. Coordinating mobile agents via blackboards and access rights. In *COORDINATION '97*, volume 1282 of *Lecture Notes in Computer Science*, pages 220–237. Springer-Verlag, 1997.

[9] R. DeNicola, G. Ferrari, and R. Pugliese. Types as specifications of access policies. In J. Vitek and C. Jensen, editors, *Secure Internet Programming: Security Issues for Distributed and Mobile Objects*, Lecture Notes in Computer Science. Springer-Verlag, 1999.

[10] D. Denning. A lattice model of secure information flow. *Communications of the ACM*, 19(5):236–242, 1976.

[11] D. Denning. Certification of programs for secure information flow. *Communications of the ACM*, 20:504–513, 1977.

[12] C. Fournet, G. Gonthier, J.J. Levy, L. Marganget, and D. Remy. A calculus of mobile agents. In U. Montanari and V. Sassone, editors, *CONCUR: Proceedings of the International Conference on Concurrency Theory*, volume 1119 of *Lecture Notes in Computer Science*, pages 406–421, Pisa, August 1996. Springer-Verlag.

[13] Cédric Fournet, Cosimo Laneve, Luc Maranget, and Didier Rémy. Implicit typing à la ML for the join-calculus. In *CONCUR: Proceedings of the International Conference on Concurrency Theory*, Lecture Notes in Computer Science, Warsaw, August 1997. Springer-Verlag.

[14] Nevin Heintz and Jon G. Riecke. The SLam calculus: Programming with secrecy and integrity. In *Conference Record of the ACM Symposium on Principles of Programming Languages*, San Diego, January 1998. ACM Press.

[15] Matthew Hennessy and James Riely. Resource access control in systems of mobile agents. Computer Science Technical Report 2/98, University of Sussex, 1998. Available from http://www.cogs.susx.ac.uk/. Extended abstract in *3rd International Workshop on High-Level Concurrent Languages (HLCL'98)*, volume 16(3) of *Electronic Notes in Theoretical Computer Science* (http://www.elsevier.nl/locate/entcs), Nice, September 1998. Elsevier.

[16] Matthew Hennessy and James Riely. Type-safe execution of mobile agents in anonymous networks. In J. Vitek and C. Jensen, editors, *Secure Internet Programming: Security Issues for Distributed and Mobile Objects*, Lecture Notes in Computer Science. Springer-Verlag, 1999. Also available as Computer Science Technical Report 3/98, University of Sussex, 1998. Available from http://www.cogs.susx.ac.uk/.

[17] Frederick Coleville Knabe. *Language Support for Mobile Agents*. PhD thesis, Carnegie-Mellon University, 1995.

[18] Dexter Kozen. Efficient code certification. Technical Report 98-1661, Cornell University, Department of Computer Science, 1998. Available from http://www.cs.cornell.edu/kozen/secure.

[19] Xavier leroy and Francois Rouaix. Security properties of typed applets. In *Conference Record of the ACM Symposium on Principles of Programming Languages*, San Diego, January 1998. ACM Press.

[20] Robin Milner. The polyadic π-calculus: a tutorial. Technical Report ECS-LFCS-91-180, Laboratory for Foundations of Computer Science, Department of Computer Science, University of Edinburgh, UK, October 1991. Also in *Logic and Algebra of Specification*, ed. F. L. Bauer, W. Brauer and H. Schwichtenberg, Springer-Verlag, 1993.

[21] Greg Morrisett, David Walker, Karl Crary, and Neal Glew. From System F to typed assembly language. In *Conference Record of the ACM Symposium on Principles of Programming Languages*, pages 85–97, San Diego, January 1998. ACM Press.

[22] George Necula. Proof-carrying code. In *Conference Record of the ACM Symposium on Principles of Programming Languages*. ACM Press, January 1996.

[23] George C. Necula and Peter Lee. Safe, untrusted agents using proof-carrying code. In *Mobile Agent Security,*, number 1419 in Lecture Notes in Computer Science. Springer-Verlag, 1998.

[24] R. De Nicola, G. Ferrari, and R. Pugliese. Klaim: a kernel language for agents interaction and mobility. *IEEE Transactions on Software Engineering*, 24(5):315–330, 1998.

[25] Atsuhi Ohori and Kazuhiko Kato. Semantics for communication primitives in a polymorphic language. In *Conference Record of the ACM Symposium on Principles of Programming Languages*, Charleston, January 1993. ACM Press.

[26] J. Palsberg and P. Ørbaek. Trust in the λ-calculus. In *Static Analysis Symposium*, volume 983 of *Lecture Notes in Computer Science*, pages 314–329. Springer-Verlag, 1995.

[27] Benjamin Pierce and Davide Sangiorgi. Typing and subtyping for mobile processes. *Mathematical Structures in Computer Science*, 6(5):409–454, 1996. Extended abstract in LICS '93.

[28] James Riely and Matthew Hennessy. Trust and partial typing in open systems of mobile agents. Computer Science Technical Report 4/98, University of Sussex, 1998. Available from http://www.cogs.susx.ac.uk/. Extended Abstract in *Conference Record of the ACM Symposium on Principles of Programming Languages*, San Antonio, January 1999. ACM Press.

[29] James Riely and Matthew Hennessy. A typed language for distributed mobile processes. In *Conference Record of the ACM Symposium on Principles of Programming Languages*, San Diego, January 1998. ACM Press.

[30] Tatsurou Sekiguchi and Akinori Yonezawa. A calculus with code mobility. In *FMOODS '97*, Canterbury, July 1997. Chapman and Hall.

[31] Peter Sewell. Global/local subtyping and capability inference for a distributed π-calculus. In *Proceedings of the International Colloquium on Automata, Languages and Programming*, volume 1433 of *Lecture Notes in Computer Science*. Springer-Verlag, July 1998.

[32] Peter Sewell, Pawel Wojciechowski, and Benjamin Pierce. Location-independent communication for mobile agents: a two-level architecture. In *Workshop on Internet Programming Languages (WIPL)*, Chicago, 1998. Available from http://www.cl.cam.ac.uk/users/pes20/.

[33] Geoffrey Smith and Dennis Volpano. Secure information flow in a multi-threaded imperative language. In *Conference Record of the ACM Symposium on Principles of Programming Languages*, San Diego, January 1998. ACM Press.

[34] R. Stata and M. Abadi. A type system for java bytecode subroutines. In *Conference Record of the ACM Symposium on Principles of Programming Languages*, San Diego, January 1998. ACM Press.

[35] J. Vitek and G. Castagna. A calculus of secure mobile computations. In *Workshop on Internet Programming Languages (WIPL)*, Chicago, 1998. Available from http://cuiwww.unige.ch/~jvitek/.

[36] D. Volpano, G. Smith, and C. Irvine. A sound type system for secure flow analysis. *Journal of Computer Security*, 4(3):1–21, 1996.

[37] Frank Yellin. Low-level security in java. In *WWW4 Conference*, 1995. Available from http://www.javasoft.com/sfaq/verifier.html.