An Algebraic Approach to Asymmetric Delegation and Polymorphic Label Inference

Silei Ren[1][0000−0002−4182−0211], Coşku Acay[2][0000−0002−0487−1167], and Andrew C. Myers[1][0000−0001−5819−7588]

[1] Cornell University, Ithaca NY 14850, USA
sr2262@cornell.edu,andru@cs.cornell.edu
[2] Observe, Inc., San Mateo CA 94402, USA
coskuacay@gmail.com

Abstract. Language-based information flow control (IFC) enables reasoning about and enforcing security policies in decentralized applications. While information flow properties are relatively extensional and compositional, designing expressive systems that enforce such properties remains challenging. In particular, it can be difficult to use IFC labels to model certain security assumptions, such as semi-honest agents. Motivated by these modeling limitations, we study the algebraic semantics of lattice-based IFC label models, and propose a semantic framework that allows formalizing asymmetric delegation, which is partial delegation of confidentiality or integrity. Our framework supports downgrading of information and ensures their safety through nonmalleable information flow (NMIF). To demonstrate the practicality of our framework, we design and implement a novel algorithm that statically checks NMIF and a label inference procedure that efficiently supports bounded label polymorphism, allowing users to write code generic with respect to labels.

1 Introduction

Information Flow Control (IFC) [23, 38] is a well-established approach for enforcing information security. Using *labels*, IFC systems specify fine-grained policies on information flow that can be fully or partly enforced through compile-time analysis. These policies articulate the confidentiality and integrity goals of IFC systems, which are *security properties* [23]: hyperproperties [12] that constrain the set of system behaviors.

The most prominent security property for IFC systems is *noninterference* [18], but it is too restrictive in practice. A major challenge for adopting language-based IFC is providing developers with expressive yet intuitive ways to specify their intended security policies. To capture more nuanced security policies, the expressiveness of IFC systems is enhanced by *downgrading mechanisms* such as *declassification* of confidential information and *endorsement* of untrusted information. Misuse of these mechanisms is further mitigated by enforcing *nonmalleable information flow* [9] (NMIF), a security property controlling downgrading.

Delegation is another common mechanism for specifying security. Delegation allows one principal to grant (delegate) power to another, expressing that the first principal trusts the second. Delegation can compactly represent important aspects of the system's security policy: when there is delegation between two principals, ensuring the delegator's security also necessitates enforcing the delegatee's security. Delegation is commonly supported not only in information flow control systems [4, 5, 30, 34], but also in a wide range of enforcement mechanisms, including access control [7, 13, 39] (where delegation is often referred to as a *principal* or *role hierarchy*), authorization logics [1, 2, 4, 20, 21], and capability systems [25, 27].

The expressive power of delegation can be increased through what we call *asymmetric delegation*: fine-grained delegation of either *confidentiality* or *integrity*. Intuitively, when a principal Alice delegates her confidentiality to another principal Bob, she allows Bob to observe all information visible to her. When Alice delegates her integrity to Bob, she trusts that all information accepted by Bob has not been maliciously modified. With asymmetric delegation, we can model security settings like the semi-honest trust assumption in cryptographic applications and the security setting of blockchains. In the semi-honest setting, principals trust each other to follow the protocol (trust each other with integrity), but do not trust each other with their secrets (but not confidentiality). In the blockchain setting, principals do not trust each other to follow protocols, but all information is public: they effectively trust each other with respect to confidentiality.

While asymmetric delegation increases the expressive power of IFC systems, its precise role—particularly in the presence of downgrading—remains poorly understood. We address this gap by presenting a general and expressive semantic framework for IFC labels that formalizes both asymmetric delegation and its interaction with downgrading. Although prior work [5, 44] develops IFC systems that support certain forms of asymmetric delegation, these systems lack sound and complete NMIF enforcement. Building on our framework, we develop algorithms for verifying the associated semantic security properties.

Experience with language-based security highlights the importance of IFC label inference to reduce the burden on programmers [3, 34]. In addition, allowing programmers to write code that is generic with respect to labels enhances modularity and code reuse. We support such generic programming through an efficient label inference procedure that supports bounded label polymorphism.

To evaluate our approach, we update the label model of the Viaduct compiler [3] and extend its static information flow analysis. Our implementation features a more concise and modular syntax for specifying trust assumptions, as well as a more efficient label inference procedure.

The rest of the paper is structured as follows:

- §2 motivates asymmetric delegation using a semi-honest secure multiparty computation (MPC) program.
- §3 and §4 study the effects of asymmetric delegation on security properties using a novel semantic framework.
- §5 presents algorithms that statically enforce the security properties.
- §6 introduces an inference procedure supporting bounded label polymorphism.

```
 1  host Alice : {A ∧ B←}
 2  host Bob   : {B ∧ A←}
 3
 4  val a: {A ∧ B←} = Alice.input
 5  val b: {B ∧ A←} = Bob.input
 6  val w: {A ∧ B} = a > b
 7
 8  Alice.output(
 9    declassify w to {A ∧ B←})
10  Bob.output(
11    declassify w to {B ∧ A←})
```

Fig. 1. Yao's Millionaires' problem in Viaduct [3]. The programmer must manually assign labels to hosts.

```
 1  host Alice, Bob
 2  assume Alice = Bob for integrity
 3
 4  val a: {Alice} = Alice.input
 5  val b: {Bob}   = Bob.input
 6  val w: {Alice ⊔ Bob} = a > b
 7
 8  Alice.output(
 9    declassify w to {Alice})
10  Bob.output(
11    declassify w to {Bob})
```

Fig. 2. Yao's Millionaires' problem implemented with delegation. `Alice ⊔ Bob` is shorthand for $\langle \texttt{Alice} \wedge \texttt{Bob}, \texttt{Alice} \vee \texttt{Bob} \rangle$.

2 A Case for Delegation

2.1 Semi-Honest Attackers in Cryptography

Asymmetric delegation can capture a wide variety of security settings. Already mentioned is the semi-honest threat model, widely studied in the cryptography literature [45]. In this model, principals correctly follow the protocol, but attempt to improperly learn other principals' secrets. Modeling the semi-honest setting in IFC systems remains a challenge. We first give an example of modeling semi-honest security in Viaduct [3], a state-of-the-art compiler that translates information flow policies to cryptographic protocols. We then illustrate how delegation improves usability and modularity.

Consider Yao's well-known Millionaires' Problem [45], where Alice and Bob wish to compare their wealth without revealing actual numbers. Figure 1 shows a Viaduct implementation. Lines 1 and 2 declare the hosts `Alice` and `Bob` and assign them information flow labels that capture the security assumptions. The hosts are assigned different and incomparable confidentiality labels (`A` for `Alice` and `B` for `Bob`) but the same integrity label (`A ∧ B`) to reflect the trust relation in the semi-honest model. Lines 4 and 5 gather input from the hosts; input from a host has the same label as that host. Line 6 stores the result of the comparison in `w`, which has a label following standard IFC rules: the result of a computation is more secret and less trusted than all of its inputs. Specifically, `w` has a confidentiality of `A ∧ B` since it is derived using secret data from both hosts, and has an integrity of `A ∧ B` since that is the integrity of both inputs. Finally, lines 9 and 11 output `w` to `Alice` and `Bob`, respectively. Note that sending `w` to `Alice` leaks information about `Bob`'s secret data (`b`), which violates noninterference. Viaduct requires an explicit **`declassify`** statement to indicate that information leakage is intentional.

2.2 Modeling Security with Delegation

Viaduct models security by encoding trust into labels, but this approach has problems. First, programmers must encode security assumptions by carefully crafting host labels, which becomes tricky in large systems with many assumptions. Second, this encoding pollutes the entire program. In fig. 1, every label annotation must acknowledge the semi-honest assumption by carrying around additional integrity (i.e., $\mathtt{A}^{\leftarrow}$ or $\mathtt{B}^{\leftarrow}$). And third, the encoding breaks modularity. For example, to add a new host `Chuck` to the program, we would need to edit every label annotation to carry an extra integrity component of $\mathtt{C}^{\leftarrow}$, requiring changes throughout the program even though `Chuck` is not involved in this portion of the computation. Delegation addresses all of these issues.

Figure 2 implements Yao's Millionaires' Problem using delegation. Hosts are no longer assigned cryptic information flow labels; instead, line 2 directly states the security assumption: `Alice` and `Bob` trust each other for integrity. Variables have intuitive labels that do not need to repeat the semi-honest security assumption: input from `Alice` has label `Alice`. Finally, adding a new host `Chuck` requires no edits to existing code; we only need to add the following lines:[3]

```
host Chuck
assume Alice = Chuck for integrity
```

2.3 Nonmalleable Information Flow

Downgrading statements (**`declassify`** and **`endorse`**) deliberately violate noninterference, so their unrestricted use poses a threat to security. Prior work [33, 47] identifies cases where the attacker can exploit downgrading to gain undue influence over the execution, and proposes *robust declassification* and *transparent endorsement* to limit such cases.

Robust declassification requires that untrusted data is not declassified, and transparent endorsement requires that secret data is not endorsed. NMIF combines these two restrictions, which are key to enabling the Viaduct compiler to securely instantiate programs with cryptography [3].

Here, "secret" and "trusted" are relative to a given attacker, and NMIF must hold for all attackers. In practice, the program cannot be type-checked separately for every possible attacker, so a conservative condition is enforced: downgraded data must be at least as trusted as it is secret. For our example program, this condition means `w` must have integrity stronger than or equal to its confidentiality. This condition is immediate in Viaduct since `w` has label $\langle \mathtt{Alice} \wedge \mathtt{Bob}, \mathtt{Alice} \wedge \mathtt{Bob} \rangle$ in fig. 1. On the other hand, the same variable `w` in fig. 2 has label $\langle \mathtt{Alice} \wedge \mathtt{Bob}, \mathtt{Alice} \vee \mathtt{Bob} \rangle$, which seemingly has weaker integrity than confidentiality (logically, $\mathtt{Alice} \vee \mathtt{Bob}$ does *not* imply $\mathtt{Alice} \wedge \mathtt{Bob}$). However,

[3] In fact, we could even support separate compilation as the program need not be type-checked again: a program considered secure with fewer assumptions is secure with more assumptions.

`Alice` = `Bob` for integrity, so this label is equivalent to ⟨`Alice ∧ Bob, Alice ∧ Bob`⟩ using the following derivation:

$$\texttt{Alice} \vee \texttt{Bob} = \texttt{Alice} \vee \texttt{Alice} = \texttt{Alice} = \texttt{Alice} \wedge \texttt{Alice} = \texttt{Alice} \wedge \texttt{Bob}$$

Delegation necessitates equational reasoning under assumptions, and NMIF creates an interaction between confidentiality and integrity. The combination of these two features is what makes asymmetric delegation tricky: the cleaner syntax comes at the cost of additional technical complexity. The following sections tame this complexity by developing a semantic framework for labels, and algorithms that follow the semantics.

3 Semantic Framework

3.1 The Lattice of Principals

We build our semantic framework upon the *lattice of principals*, used in prior work in authorization logics and information flow systems [3–5, 31, 34, 42, 44].

A principal $p \in \mathbb{P}$ refers to an entity in decentralized systems that can be either concrete, such as users or server machines, or abstract, such as RBAC roles [39] or quorums [51]. In IFC systems, they are often used as labels to annotate policies on use of information [4]. For example, `a: Alice` in fig. 2 requires the variable `a` to only be written by principals that can influence `Alice`'s data, and to remain secret to principals who cannot observe `Alice`'s information.

Principals are ordered by authority. When q delegates trust to p, we say p *acts for* q, written as $p \Rightarrow q$. Conjunction (the "and" logic connective) between principals $p \wedge q$ represents the least combined authority of p and q, and disjunction ("or") $p \vee q$ represents greatest common authority. The maximum authority $\perp$ acts for all other authorities, and the minimum authority $\top$ trusts all other authorities. More authority is associated with elements lower in the lattice: $\perp \Rightarrow p \wedge q \Rightarrow p \Rightarrow p \vee q \Rightarrow \top$. [4]

Additionally, we use a *delegation context*, with the form $\theta = p_1 \Rightarrow q_1, \cdots, p_n \Rightarrow q_n$, to specify delegations that are not implied by the logical structure of the principal lattice. The declaration **`assume`** `Alice = Bob` **`for integrity`** from fig. 2 is an example of a delegation context, specifying both `Alice` ⇒ `Bob` and `Bob` ⇒ `Alice`. Using the delegation context is compatible with much prior work in IFC. For example, the *trust configuration* from FLAM [4], *meta-policies* from the Rx model [44], *interpretation function* from DLM [29], and *authority lattice* from label algebra [28] are all delegation contexts written differently.

The lattice of principals can be interpreted as an authorization logic [1, 17] where each principal is a proposition about authorization policy. As authorization

[4] It might seem odd to represent maximum authority as $\perp$ and minimum authority with $\top$, since some prior work (e.g., [4]) makes the opposite choice. An intuitive justification: the "false" proposition entails everything, so no real principal can have authority $\perp$. All principals are trusted with $\top$.

logics are often built upon propositional intuitionistic logics, whose algebraic models are *Heyting algebra* [37], we assume $\mathbb{P}$ is distributive.[5]

3.2 Delegation and Attackers

In the MPC example, a delegation context makes some principals equivalent. In this subsection, we give delegation a precise semantics.

In systems with decentralized trust, all principals see other principals as potential attackers. In the extreme case where no principal trusts another, the attacker with respect to each principal controls all other principals. Formally, we characterize an attacker $A \subseteq \mathbb{P}$ by the set of principals it controls.

To trust a principal is to disregard the case where it is the attacker. Conversely, when a principal is attacker-controlled, so are the principals it acts for. Formally:

Definition 1 (Consistent Attackers). *A is consistent with θ ($A \models \theta$) when:*

$$\forall p, q \in \mathbb{P} \,\mathbf{.}\, ((p \Rightarrow q) \vee (p \Rightarrow q) \in \theta) \implies (p \in A \implies q \in A)$$

The set of consistent attackers is an *attacker model*: $\mathbb{A}_{|\theta} = \{A \in \mathbb{A} \mid A \models \theta\}$.

The semantic trust levels of principals can be compared based on the set of consistent attackers that control the principals. Formally:

Definition 2 (Acts-for Semantics). *p acts for q (written $\theta \models p \leq q$) when:*

$$\forall A \in \mathbb{A}_{|\theta} \,\mathbf{.}\, p \in A \implies q \in A$$

We use "$\leq$" to denote the semantics of the acts-for relation "$\Rightarrow$". When viewing principals as authorization propositions, "acts-for" stands for "implies", and a delegation context is a theory (list of propositions). Each consistent attacker is a consistent interpretation (truth assignment) of the principals and the delegation context, where 1 is assigned to the principals the attacker controls.

In fact, a delegation context determines a *congruence relation*[6] over the lattice of principals, where $p \equiv_\theta q$ is defined as $(\theta \models p \leq q) \wedge (\theta \models q \leq p)$. As a result, $\equiv_\theta$ induces a quotient lattice $\mathbb{P}/{\equiv_\theta}$ where mutually delegating principals are in the same equivalence class. This quotient lattice is precisely the *Lindenbaum algebra* [8] of the theory θ: the "smallest" algebraic model $\mathbb{P}$ where θ holds.

Theorem 1 (Algebraic Model). *The algebraic model of the principal lattice $\mathbb{P}$ under delegation context θ is the quotient lattice $\mathbb{P}/{\equiv_\theta}$.*[7]

The semantics of consistent attackers makes them the *prime filters* of the lattice of principals. This is a result of the Stone's Representation Theorem of Distributive Lattices [43], which says elements from distributive lattices can be fully characterized by their prime filters.

[5] A Heyting algebra is a distributive lattice that supports the *relative pseudocomplement* ($\rightarrow$) operation. We do not need the $\rightarrow$ operator until label inference.

[6] A congruence is an equivalence relation that preserves the lattice structure.

[7] Full proofs of all theorems from this paper are available in the technical report [36].

Theorem 2 (Attacker Model). $\mathbb{A}_{|\theta}$ *is the set of* prime filters *of* $\mathbb{P}/{\equiv_\theta}$.

Prime filters have intuitive interpretations, which abstracts and generalizes attacker models from prior work [9, 22]. $A \subseteq \mathbb{P}$ is a prime filter when:

- $\top \in A$: All attackers control the weakest authority;
- $\bot \notin A$: No attacker controls the strongest authority;
- If $p \Rightarrow q$ and $p \in A$, then $q \in A$: Attackers are consistent;
- If $p \in A$ and $q \in A$, then $p \wedge q \in A$: An attacker controls the least combined authority of principals it controls;
- If $p \vee q \in A$, then either $p \in A$ or $q \in A$: If two principals are not controlled by an attacker, neither is their greatest common authority.

3.3 Labels

Information flow systems mainly consider two aspects of security: confidentiality (read authority) and integrity (write authority). To express differing confidentiality and integrity policies, we use *the lattice of labels*, which are pairs of principals $\mathbb{L} = \mathbb{P} \times \mathbb{P}$. For a label $\ell = \langle p, q \rangle$, p represents confidentiality and q represents integrity. Like principals, each label reflects the authority required for a principal to access information. For example, information labeled $\langle \bot, \top \rangle$ can be read by no principal except the strongest $\bot$, but it can be influenced by any principal.

A label ℓ act for ℓ' when both ℓ acts for ℓ' for both confidentiality and integrity. Similarly, conjunction/disjunction of labels is defined by the conjunction/disjunction of their confidentiality and integrity. An asymmetric delegation context is a pair of different delegation contexts $\Theta = \langle \theta_{\mathrm{c}}, \theta_{\mathrm{i}} \rangle$.

In general, *asymmetric attackers* $\mathcal{A} \in \mathbf{A} = \mathbb{A} \times \mathbb{A}$ may control different principals for confidentiality and integrity. We write $\mathcal{A} = \langle C \in \mathbb{A}, I \in \mathbb{A} \rangle$, where C represents the principals $\mathcal{A}$ controls for confidentiality, and I those for integrity. Consequently, the attacker model $\mathbf{A}$ is a pair of prime filters.

As visualized in fig. 3, each attacker defines secret ($\mathcal{S}$), public ($\mathcal{P}$), trusted ($\mathcal{T}$) and untrusted ($\mathcal{U}$) sets over the lattice of labels.

$$\mathcal{P}_{\langle C,I \rangle} = \{\langle p, q \rangle \mid p \in C\}\ ,\quad \mathcal{U}_{\langle C,I \rangle} = \{\langle p, q \rangle \mid q \in I\},$$
$$\mathcal{S}_{\langle C,I \rangle} = \{\langle p, q \rangle \mid p \notin C\}\ ,\quad \mathcal{T}_{\langle C,I \rangle} = \{\langle p, q \rangle \mid q \notin I\}$$

4 Security Hyperproperties and Delegation

Our semantic framework formalizes a key insight relating the attacker model and delegation: more delegation means fewer attackers. In turn, fewer attackers should make it easier for programs to be considered secure. To express delegation, we extend prior definitions of IFC security *hyperproperties* [12] with our semantic framework. Rather than a standard proof of security for an IFC-typed core calculus, we abstract away from computation to concentrate on the core judgment in IFC systems: when it is safe to relabel information, under a delegation context.

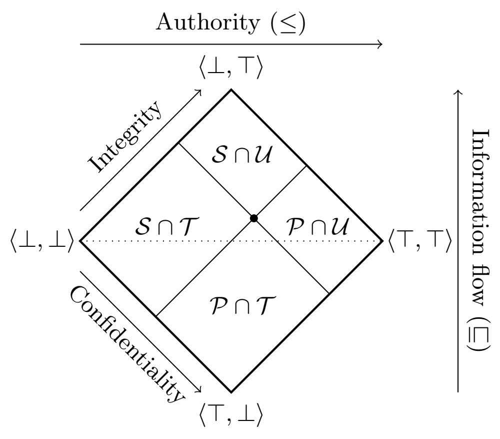
Authority ($\leq$)
$\langle\bot,\top\rangle$
Integrity
$\mathcal{S}\cap\mathcal{U}$
$\mathcal{S}\cap\mathcal{T}$
$\mathcal{P}\cap\mathcal{U}$
$\langle\bot,\bot\rangle$
$\langle\top,\top\rangle$
$\mathcal{P}\cap\mathcal{T}$
Confidentiality
$\langle\top,\bot\rangle$
Information flow ($\sqsubseteq$)

Fig. 3. The lattice of labels $(\mathbb{L}, \leq)$ and the lattice of information flow $(\mathbb{L}, \sqsubseteq)$ share the same underlying set, but use a different ordering. The dotted line depicts labels with equal confidentiality and integrity: strictly above are compromised labels, and on or below are uncompromised labels.

The simple system has states $\sigma \in \Sigma$, intentionally left unspecified. An execution of the system emits a trace t, which is a sequence of states. Define the *behavior* $B \in \mathbb{B}$ of a program to be the *set* of possible execution traces it can emit. A hyperproperty $\mathbb{HP} \subseteq \mathbb{B}$ is a set of behaviors. A program satisfies a hyperproperty when its behavior is a member of the hyperproperty.

In decentralized IFC systems, hyperproperties $\mathbb{HP}_{\mathcal{A}}$ are parameterized by the choice of attacker: a program secure against one attacker may be insecure against another. Let the hyperproperty $\mathbb{HP}_{\mathbf{A}}$ be the hyperproperty that characterizes programs that are secure against all possible attackers from $\mathbf{A}$. A behavior B of a program falls into $\mathbb{HP}_{\mathbf{A}}$ precisely when $B \in \mathbb{HP}_{\mathcal{A}}$ for all $\mathcal{A} \in \mathbf{A}$:

$$\mathbb{HP}_{\mathbf{A}} = \bigcap_{\mathcal{A}\in\mathbf{A}} \mathbb{HP}_{\mathcal{A}} \qquad\qquad \Theta(\mathbb{HP}_{\mathbf{A}}) = \bigcap_{\mathcal{A}\in\mathbf{A}_{|\Theta}} \mathbb{HP}_{\mathcal{A}}$$

When a program assumes a static delegation context Θ, the attacker model is further restricted to the ones consistent with Θ. Therefore, a delegation context can be understood as a hyperproperty transformer. It follows that hyperproperties accept at least as many programs after a delegation context is added.

Theorem 3. $\mathbb{HP}_{\mathbf{A}} \subseteq \Theta(\mathbb{HP}_{\mathbf{A}})$.

This matches our intuition about static delegation: the more trust assumptions, the fewer reasonable attackers, the more programs considered secure.

4.1 Noninterference and the Lattice of Information Flow

For confidentiality, noninterference [13, 38] demands that information should not flow from high to low (secret to public). Noninterference of integrity requires that information should not flow from low to high (untrusted to trusted). To illustrate how delegation affects noninterference, we adapt Clarkson and Schneider's [12] definition of *observational determinism* [18].

Definition 3 (Observational Determinism). *Let $\mathcal{L}$ be any set of* low *labels. Observational determinism $\mathbb{OD}$ is the hyperproperty:*

$$\mathbb{OD}_{\mathcal{L}} = \{B \mid \forall t_1, t_2 \in B \,.\, t_1^0 =_{\mathcal{L}} t_2^0 \implies t_1 \approx_{\mathcal{L}} t_2\}$$

State t^0 denotes the initial state of the trace t. We leave the definition of low equivalence between states unspecified, as in prior work on knowledge-based security [6, 24, 26, 41].

Noninterference for confidentiality $\mathbb{NI}^{\mathrm{c}}_{\mathcal{A}} = \mathbb{OD}_{\mathcal{P}_{\mathcal{A}}}$ and integrity $\mathbb{NI}^{\mathrm{i}}_{\mathcal{A}} = \mathbb{OD}_{\mathcal{T}_{\mathcal{A}}}$ are mere instantiations of observational determinism over public and trusted labels for some attacker $\mathcal{A}$. For an attacker model $\mathbf{A}_{|\Theta}$:

$$\Theta(\mathbb{NI}_{\mathbf{A}}) = \bigcap_{\mathcal{A} \in \mathbf{A}_{|\Theta}} (\mathbb{OD}_{\mathcal{P}_{\mathcal{A}}} \cap \mathbb{OD}_{\mathcal{T}_{\mathcal{A}}})$$

Prior work [19, 23, 30] enforces low equivalence by ensuring that low-labeled information is not influenced by high-labeled information. Concretely, a dynamic *relabel* is safe when it does not relabel high-labeled information to a low label. We formalize safe relabeling as the *flows-to* relation.

Definition 4 (Flows-to). *Label ℓ securely flows to ℓ' ($\Theta \models \ell \sqsubseteq \ell'$) when:*

$$\forall \mathcal{A} \in \mathbf{A}_{|\Theta} \,.\, (\ell' \in \mathcal{P}_{\mathcal{A}} \implies \ell \in \mathcal{P}_{\mathcal{A}}) \wedge (\ell' \in \mathcal{T}_{\mathcal{A}} \implies \ell \in \mathcal{T}_{\mathcal{A}})$$

Equivalently, $\langle \theta_{\mathrm{c}}, \theta_{\mathrm{i}} \rangle \models \langle p, q \rangle \sqsubseteq \langle p', q' \rangle$ when $\theta_{\mathrm{c}} \models p' \leq p$ and $\theta_{\mathrm{i}} \models q \leq q'$.

As visualized in fig. 3, flows-to and the acts-for define two lattices on the same underlying set of labels. The Lattice of Information operators are given by:

$$\langle p_1, q_1 \rangle \sqcup \langle p_2, q_2 \rangle = \langle p_1 \wedge p_2, q_1 \vee q_2 \rangle \qquad \langle p_1, q_1 \rangle \sqcap \langle p_2, q_2 \rangle = \langle p_1 \vee p_2, q_1 \wedge q_2 \rangle$$

4.2 Downgrading and Nonmalleable Information Flow

Some programs, such as the MPC example from fig. 2, intentionally break noninterference. Prior work on dynamic security policies either achieve downgrading by *relabeling* or by *dynamic delegation.* Visually, relabeling moves information downward in fig. 3 and dynamic delegation moves the attacker partition leftward.

Nonmalleable Information Flow (NMIF) To prevent misuse of downgrades by relabeling, Cecchetti et al. [9] propose NMIF, a security hyperproperty that combines robust declassification [47] with transparent endorsement.

Robust declassification requires that secret information flow to public only when the information is trusted. This restriction ensures that attackers do not influence disclosure of information to them. A declassification is only robust when it declassifies secret–trusted information ($\mathcal{S}_{\mathcal{A}} \cup \mathcal{T}_{\mathcal{A}}$).

Transparent endorsement is a dual condition that allows untrusted information to influence trusted information only when the information is public to the attacker. It only allows endorsement of public–untrusted information ($\mathcal{P}_{\mathcal{A}} \cap \mathcal{U}_{\mathcal{A}}$).

Therefore, secret–untrusted information should not be downgraded. Indeed, the key recipe to enforcing NMIF is to enforce noninterference for public or trusted information [9]:

$$\mathbb{NI}^{\blacktriangledown}_{\mathbf{A}} = \mathbb{OD}_{\mathcal{T}_{\mathcal{A}} \cup \mathcal{P}_{\mathcal{A}}}$$

A label is *compromised* when it is secret–untrusted for some attacker [9, 46]. To enforce NMIF, it suffices to reject downgrading information with compromised labels, so that there is no flow out from compromised labels.

Unfortunately, NMIF against $\mathbf{A}_{|\theta}$ rejects all downgrades. Namely, all labels (except for $\langle \top, \bot \rangle$) are compromised against the attacker $\mathcal{A}_{\top} = \langle \{\top\}, \mathbb{P} \rangle$ (it controls every principal for integrity and controls no principal for confidentiality). Therefore, further restrictions on the attacker model are needed.

NMIF Attackers Prior work [46, 47] assumes attackers control more confidentiality than integrity. We call them *valid attackers*:

Definition 5 (Valid Attackers). $\boldsymbol{V} = \{\langle C, I \rangle \in \mathbf{A} \mid I \subseteq C\}$

Restricting attackers to valid ones makes our framework mirror attacker models studied in the cryptography literature [45], where a principal is honest (not controlled by attacker), semi-honest (controlled by attacker for confidentiality), or malicious (controlled by attacker for both confidentiality and integrity). The valid attacker restriction excludes unrealistic "malicious but incurious" attackers.

In fact, much existing work satisfies the valid-attacker assumption by construction. For example, robust declassification [47] originally defines integrity and confidentiality by equivalence relations over system states. The state transitions an active attacker may perform are, by construction, observable by the attacker.

Definition 6 (Uncompromised Labels). *Label ℓ is uncompromised under Θ, written $\Theta \models \blacktriangledown \ell$, when ℓ is either public or trusted for all* valid attackers*:*

$$\forall \mathcal{A} \in \boldsymbol{V}_{|\Theta} \,.\, \ell \in \mathcal{P}_{\mathcal{A}} \cup \mathcal{T}_{\mathcal{A}}$$

It follows that labels of principals are uncompromised: $\Theta \models \blacktriangledown \langle p, p \rangle$.

Uncompromised labels have an alternative characterization: they are the labels with at least as much integrity as confidentiality. Of course, in the presence of asymmetric delegation, we cannot directly compare a label's confidentiality and integrity components since each component has a different set of delegations. We circumvent this problem by introducing a witnessing principal r who has no more integrity than q and no less confidentiality than p.

Theorem 4. $\langle \theta_{\mathrm{c}}, \theta_{\mathrm{i}} \rangle \models \blacktriangledown \langle p, q \rangle \iff \exists r \in \mathbb{P} \,.\, (\theta_{\mathrm{i}} \models q \leq r) \wedge (\theta_{\mathrm{c}} \models r \leq p)$.

Proofs can be found in the technical report [36]. In the absence of asymmetric delegation, theorem 4 reduces to a simple acts-for check ($\theta \models q \leq p$), which prior systems rely on [9, 47].

5 Algorithms

In IFC systems that incorporate delegation, the acts-for relation $\theta \models p_1 \leq p_2$ is frequently checked by label-inference procedures [34] and even at run time, where it can impose significant overhead [11]. Unfortunately, its semantic definition quantifies over the potentially infinite set of all attackers, which makes direct use of the definition infeasible in practice. Similarly, the definition of uncompromised labels $\Theta \models \blacktriangledown\ell$ does not yield an algorithm.

In this section, we assume oracle access to syntactic lattice operations ($\Rightarrow$, $\bot$, $\top$, $\wedge$, $\vee$) of $\mathbb{P}$, and propose sound and complete algorithms that check for the aforementioned relations. Proofs are available in the technical report [36].

5.1 Acts-for Algorithm

Algorithm 1 gives an algorithm for deciding $\theta \models p \leq q$. We write $\theta \vdash p \leq q$ to denote this algorithmic system.

Algorithm 1 (Acts-for $\theta \vdash p \leq q$).

$$\textsc{$\mathbb{P}$-Axiom}\ \frac{p \Rightarrow q}{\cdot \vdash p \leq q} \qquad \textsc{$\mathbb{P}$-Delegation}\ \frac{\theta \vdash p \wedge q' \leq q \qquad \theta \vdash p \leq q \vee p'}{\theta, p' \Rightarrow q' \vdash p \leq q}$$

The algorithm recursively applies the lattice axioms and rule $\mathbb{P}$-Delegation until the delegation context is empty for all sub-cases. The acts-for relation holds when rule $\mathbb{P}$-Axiom applies to all sub-cases.

Theorem 5 (Correctness of Acts-for). *Algorithm 1 terminates, and it is sound and complete with respect to definition 2:* $\theta \vdash p \leq q \iff \theta \models p \leq q$.

5.2 NMIF Algorithm

Neither the semantic definition of uncompromised labels (definition 6) nor their alternative characterization (theorem 4) lends itself to an algorithmic implementation. The semantic definition quantifies over all valid attackers (a potentially infinite set), and the alternative characterization conjures up an intermediate principal. Our solution is to use the alternative characterization but with a "best principal." For that, we use $\min_\theta(p)$, the highest-authority principal in the equivalence class of p.[8]

Definition 7 (Minimal Principal). $\min_\theta(p) \in \mathbb{P}$ *is the (necessarily) unique principal such that* $\theta \models p \leq q$ *if and only if* $\min_\theta(p) \Rightarrow q$ *for any* $q \in \mathbb{P}$.

[8] Recall that higher authority is lower in the authority lattice, thus the use of min as opposed to max.

We must demand more structure on $\mathbb{P}$ to compute $\min_\theta(p)$. Specifically, we rely on an oracle that returns *join-prime* factorizations of arbitrary principals. Intuitively, a join-prime principal cannot be written as the join of other principals. In finite lattices, these are the principals of the form $p = q_1 \wedge \cdots \wedge q_n$. Factorization oracles arise trivially in existing implementations of IFC models, as these are based on lattices with a finite set of principal names [3, 4, 34, 42].

Algorithm 2 (Min $\min_\theta(p) = q$).

$$\textsc{Min-Base}\quad \frac{\text{join-prime}(p) \qquad \forall (p' \Rightarrow q') \in \theta \,\mathbin{\text{.}}\, p \not\Rightarrow p'}{\min_\theta(p) = p}$$

$$\textsc{Min-Pick}\quad \frac{\text{join-prime}(p) \qquad p \Rightarrow p'}{\min_{\theta, p' \Rightarrow q'}(p) = \min_\theta(p \wedge q')}$$

$$\textsc{Min-Factor}\quad \frac{\neg\,\text{join-prime}(p) \qquad p = p_1 \vee \cdots \vee p_n \qquad \forall i \in [n] \,\mathbin{\text{.}}\, \text{join-prime}(p_i)}{\min_\theta(p) = \bigvee_{i \in [n]} \min_\theta(p_i)}$$

The rules from Algorithm 2 can be applied in any order without backtracking because of the uniqueness of $\min_\theta(p)$. Moreover, all derivations are finite since either the size of θ decreases, or we switch from a reducible element to a finite set of irreducible elements.

Theorem 6 (Correctness of Min). *Algorithm 2 terminates, and it is sound and complete with respect to definition 7.*

Our NMIF algorithm simply combines algorithms 1 and 2.

Algorithm 3 (Uncompromised Label Check $\Theta \vdash \blacktriangledown \ell$).

$$\frac{\theta_\mathrm{c} \vdash \min_{\theta_\mathrm{i}}(q) \leq p}{\langle \theta_\mathrm{c}, \theta_\mathrm{i} \rangle \vdash \blacktriangledown \langle p, q \rangle}$$

Theorem 7 (Correctness of Uncompromised Label Check). *Algorithm 3 terminates, and it is sound and complete with respect to theorem 4.*

6 Label Inference

In practical IFC systems, label inference is an important way to avoid redundant user annotations, since it is secure to infer labels of intermediate computations from their inputs and outputs. Typically, label inference is performed by solving a system of constraints over lattice elements [34, 49]. IFC systems further benefit from *bounded label polymorphism*, which allows user to write library code reusable at different security levels. In this section, we show how to do label inference directly over the algebraic model of labels using the algorithms from §5.

```
host Alice, Bob, Chuck
assume Alice = Bob for integrity
assume Bob = Chuck for integrity

fun average(a: int, b: int): int
{
  return (a + b) / 2
}
```

```
fun main() {
  val a = Alice.input
  val b = Bob.input
  val c = Chuck.input

  val r1 = average(a, b)
  val r2 = average(b, c)
}
```

Fig. 4. The `average` function is implicitly polymorphic over the labels of its arguments.

6.1 Bounded Label Polymorphism

As in traditional type systems, allowing code that is generic over labels increases expressiveness significantly. Existing IFC-based languages like Jif [34] and Flow Caml [40] support bounded label polymorphism, which allows functions to be parameterized over labels that are bounded by specified security levels. Annotation burden on users can be further reduced by assuming information flow from function arguments to return values by default.

In fig. 4, the annotation-free polymorphic function `average` is applied in `main` to arguments with different security labels. Shown below is the same function with explicit annotations, all of which can be inferred.

```
fun average[X, Y, Z](a: int{X}, b: int{Y}): int{Z}
    where (X ⊔ Y ⊑ Z)
```

Label inference assigns existential *label variables* to all unlabeled expressions and creates constraints based on IFC typing rules. The constraints are then solved by a constraint solver that uses parameter bounds as delegation contexts.

In each function, polymorphic label variables are treated as label constants, so solutions of label variables are expressed by both label constants and polymorphic label variables. Type checking at call sites ensures that the parameter bounds are satisfied. As functions have their own delegation contexts, a new constraint system is solved for each function.

6.2 Constraint Solver

We describe a novel constraint solver that computes the minimum-semantic-authority solution to constraint systems with delegation contexts. Minimum-authority solutions are desirable because they allow systems to choose cheaper security enforcement mechanisms [3, 10, 15, 16, 48, 50].

Label and Principal Constraints Figure 5 gives the syntax of the label constraint language. Expressions in the constraint language include label constants and label variables, authority projections, as well as standard lattice operations. A constraint either asserts that an expression flows to another, or asserts that an

$$
\begin{array}{lrcl}
\text{L. Constants} & \ell & & \\
\text{L. Variables} & Y & & \\
\text{L. Expressions} & L & ::= & \ell \mid Y \mid L^{\pi} \\
 & & \mid & L_1 \sqcup L_2 \mid L_1 \sqcap L_2 \\
 & & \mid & L_1 \vee L_2 \mid L_1 \wedge L_2 \\
\text{L. Constraints} & C & ::= & L_1 \sqsubseteq L_2 \mid \blacktriangledown L \\
\text{Projections} & \pi & \in & \{\mathrm{c}, \mathrm{i}\}
\end{array}
$$

Fig. 5. Syntax of label constraints.

$$
\begin{array}{lrcl}
\text{P. Constants} & p & & \\
\text{P. Variables} & Y^{\pi} & & \\
\text{P. Expressions} & P^{\pi} & ::= & p \mid Y^{\pi} \\
 & & \mid & P_1^{\pi} \vee P_2^{\pi} \mid P_1^{\pi} \wedge P_2^{\pi} \\
 & & \mid & p_1 \rightarrow P_2^{\pi} \mid \min_{\pi'}(P^{\pi'}) \\
\text{P. Constraints} & D & ::= & P_1^{\pi} \Rightarrow^{\pi} P_2^{\pi}
\end{array}
$$

Fig. 6. Syntax of principal constraints.

expression is uncompromised. We translate constraints over labels to constraints over principals to leverage algorithms from §5. Figure 6 gives the syntax of the principal constraint language. The syntax includes a principal variable Y^{π} for each combination of label variable Y and projection π. That is, Y^{c} represents the confidentiality of Y, and Y^{i} represents Y's integrity.

We index expressions P^{π} by the component π they represent. This prevents expressions like $Y_1^{\mathrm{c}} \wedge Y_2^{\mathrm{i}}$, whose components are mixed. Expressions include principal constants and principal variables, as well as principal-lattice operations $\vee$ and $\wedge$. The operation $\rightarrow$ is called the *relative pseudocomplement* of the meet operation: $p_1 \rightarrow p_2$ is defined as the minimum-authority principal p such that $p_1 \wedge p \Rightarrow p_2$. We use $\rightarrow$ to solve constraints of the form $Y^{\pi} \wedge p_1 \Rightarrow^{\pi} P_2^{\pi}$.The $\min_{\pi}()$ operation allows mixing integrity and confidentiality components; we use it when solving for labels that must be uncompromised. Principal constraints have the form $P_1^{\pi} \Rightarrow^{\pi} P_2^{\pi}$, which stands for $\theta_{\pi} \models P_1^{\pi} \leq P_2^{\pi}$.

Figure 7 gives rules for translating label constraints to principal constraints. The definition of $[\![L]\!]_{\pi}$ is a straightforward encoding of the label-lattice operations. Using $[\![L]\!]$, we translate a flows-to ($\sqsubseteq$) constraint to two acts-for ($\Rightarrow$) constraints, one for each label component. The constraint $\blacktriangledown L$ follows from algorithm 3.

$$\boxed{[\![C]\!] = D_1, \ldots, D_n}$$

$$[\![L_1 \sqsubseteq L_2]\!] = [\![L_2]\!]_{\mathrm{c}} \Rightarrow^{\mathrm{c}} [\![L_1]\!]_{\mathrm{c}},\ [\![L_1]\!]_{\mathrm{i}} \Rightarrow^{\mathrm{i}} [\![L_2]\!]_{\mathrm{i}} \qquad [\![\blacktriangledown L]\!] = [\![L]\!]_{\mathrm{i}} \Rightarrow^{\mathrm{i}} \min_{\mathrm{c}}([\![L]\!]_{\mathrm{c}})$$

$$\boxed{[\![L]\!]_{\pi} = P^{\pi}}$$

$$
\begin{array}{ll}
[\![L_1 \sqcup L_2]\!]_{\pi} = [\![(L_1 \wedge L_2)^{\mathrm{c}} \wedge (L_1 \vee L_2)^{\mathrm{i}}]\!]_{\pi} & [\![\langle p, q\rangle]\!]_{\pi} = \begin{cases} p & \text{if } \pi = \mathrm{c} \\ q & \text{if } \pi = \mathrm{i} \end{cases} \\
[\![L_1 \sqcap L_2]\!]_{\pi} = [\![(L_1 \vee L_2)^{\mathrm{c}} \wedge (L_1 \wedge L_2)^{\mathrm{i}}]\!]_{\pi} & [\![Y]\!]_{\pi} = Y^{\pi} \\
[\![L_1 \vee L_2]\!]_{\pi} = [\![L_1]\!]_{\pi} \vee [\![L_2]\!]_{\pi} & [\![L^{\pi'}]\!]_{\pi} = \begin{cases} [\![L]\!]_{\pi} & \text{if } \pi = \pi' \\ \top & \text{if } \pi \neq \pi' \end{cases} \\
[\![L_1 \wedge L_2]\!]_{\pi} = [\![L_1]\!]_{\pi} \wedge [\![L_2]\!]_{\pi} &
\end{array}
$$

Fig. 7. Translating label constraints to principal constraints.

Solving Principal Constraints Our constraint solver requires the left-hand side of each constraint to be atomic (a constant or a variable), that is, constraints of the form $p_1 \Rightarrow^\pi P_2^\pi$ and $Y_1^\pi \Rightarrow^\pi P_2^\pi$. We exhaustively apply the equational axioms of Heyting algebra (e.g., associativity, absorption, distributivity, etc.) until no left-hand side of any constraint can be further simplified. As equational axioms are syntactic rewrites, it does not change the constraint system over the underlying algebra. Moreover, this process always terminates, and ensures that the left-hand side of each constraint either is atomic or contains a meet ($\wedge$).[9]

Constraint solving fails if the left-hand side of any constraint contains a meet, since such systems do not have unique solutions. For example, the system $Y_1 \wedge Y_2 \Rightarrow \texttt{Alice}$ has no minimal solution: we can assign $\{Y_1 \mapsto \texttt{Alice}, Y_2 \mapsto \top\}$ or $\{Y_1 \mapsto \top, Y_2 \mapsto \texttt{Alice}\}$, but neither solution is better than the other. Compiler implementations need to either restrict the syntax of polymorphic constraints, or report errors during constraint solving.

Once the simplification process succeeds, we extend the algorithm of Rehof and Mogensen [35] for iteratively solving semi-lattice constraints. We initialize all principal variables to $\top$, and use unsatisfied constraints to update variables repeatedly until a fixed point is reached, using the rule:

$$\text{given} \quad Y^\pi \Rightarrow^\pi P^\pi, \quad \text{set} \quad Y^\pi := Y^\pi \wedge \text{current-value}(\Theta, P^\pi),$$

where current-value(Θ, P^π) is the value of P^π according to the current assignment.

Note that constraints that have constants p on the left-hand side are ignored during the fixed point computation. Once a fixed-point solution is reached, we perform the following check for each constraint with a constant left-hand side:

$$\text{given} \quad p \Rightarrow^\pi P^\pi, \quad \text{check} \quad \theta_\pi \models p \leq \text{current-value}(\Theta, P^\pi).$$

We state and prove in the technical report [36] that this process terminates with the minimum-authority solution if all such constraints are satisfied; otherwise, there is no valid solution.

6.3 Implementation

We modified the parser of the Viaduct compiler [3] with the delegation syntax, and extended its static analysis procedure with our label inference algorithm. [10]

Because the original Viaduct constraint solver can only make syntactic comparison ($\Rightarrow$) between labels, it instantiates polymorphic variables with constant principal names. Therefore, Viaduct has to run a *specialization procedure* to create a monomorphic copy of a function at each call site. In nested function calls, the number of monomorphic functions created grows exponentially with the depth of calls. Recursive calls need to be handled explicitly to ensure termination.

Thanks to delegation contexts, label inference no longer requires monomorphic functions and can be done in one pass. For each function call site, the specialization

[9] Translation rules in fig. 7 never generate constraints with $\rightarrow$ or $\min_\pi(\cdot)$ on the left-hand side, and the constraint simplification eliminate all joins ($\vee$) on the left.

[10] Code available at: https://github.com/apl-cornell/viaduct.

procedure memoizes the argument labels, and avoids creating duplicates of monomorphic functions that are instantiated with the same polymorphic argument labels, eliminating the need to treat recursive functions separately. Without duplication, our procedure creates a minimum-sized set monomorphic functions.

Empirically, type inference is fast. On all of the Viaduct benchmarks, it terminates within 300 milliseconds on a Macbook air 2023 with an M2 chip.

7 Related Work

Expressive trust delegation has been widely investigated in access control and capability systems [25, 39], where delegation is called *role hierarchy* [39]. Such systems support role adoption, which is a form of dynamic delegation. However, access control systems generally do not connect to information flow security properties. Many prior IFC systems have delegation abstractions compatible with our framework, but they either lack support for asymmetric delegation [3, 9, 46], or do not explore its effect over hyperproperties [4, 26, 32, 33, 47].

Our inference algorithm differs from prior implementations of syntax-directed IFC label inference [3, 34, 49] by operating directly over the underlying algebra. This algebraic approach enhances extensibility: adding principals or delegations does not invalidate existing analysis. Dolan [14] proposes an inference algorithm for an expressive algebraic type system with type constructors and recursive function types. However, their type system is inherently complex, and the inference doesn't produce easily interpretable representions of type. In contrast, our label system balances expressiveness with a simple and effective inference algorithm.

FLAM [4] proposes a label model and defines *robust authorization* to address security vulnerabilities arising from dynamic delegation. However, robust authorization is a proof-theoretic definition with no semantic model. Arden and Myers [5] propose the FLAC calculus, based on the FLAM label model, and prove robust declassification for a language that downgrades using dynamic delegation. Similarly, FLAC has no attacker semantics. Cecchetti et al. [9] define NMIF, but their system cannot explicitly express delegations between atomic principals.

8 Conclusion and Future Directions

We present an algebraic semantic framework for IFC labels that models asymmetric delegation, along with sound and complete algorithms that enforce security properties and infer security annotations. Our approach provides a solid foundation for building modular, expressive and extensible IFC systems.

Acknowledgments. This work was supported by the National Science Foundation under NSF grant 1704788. Any opinions, findings, conclusions, or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation. We also thank Suraaj Kanniwadi and Ethan Cecchetti for discussions and feedback.

Disclosure of Interests. The authors have no competing interests to declare that are relevant to the content of this article.

Bibliography

[1] Abadi, M.: Access control in a core calculus of dependency. In: 11th ACM SIGPLAN Int'l Conf. on Functional Programming. pp. 263–273. ACM, New York, NY, USA (2006). https://doi.org/10.1145/1159803.1159839

[2] Abadi, M.: Variations in access control logic. In: van der Meyden, R., van der Torre, L. (eds.) Deontic Logic in Computer Science, Lecture Notes in Computer Science, vol. 5076, pp. 96–109. Springer Berlin Heidelberg (2008). https://doi.org/10.1007/978-3-540-70525-3_9

[3] Acay, C., Recto, R., Gancher, J., Myers, A., Shi, E.: Viaduct: An extensible, optimizing compiler for secure distributed programs. In: 42nd ACM SIGPLAN Conf. on Programming Language Design and Implementation (PLDI). pp. 740–755. ACM (Jun 2021). https://doi.org/10.1145/3453483.3454074

[4] Arden, O., Liu, J., Myers, A.C.: Flow-limited authorization. In: 28th IEEE Computer Security Foundations Symp. (CSF). pp. 569–583 (Jul 2015). https://doi.org/10.1109/CSF.2015.42

[5] Arden, O., Myers, A.C.: A calculus for flow-limited authorization. In: 29th IEEE Computer Security Foundations Symp. (CSF). pp. 135–147 (Jun 2016). https://doi.org/10.1109/CSF.2016.17

[6] Askarov, A., Chong, S.: Learning is change in knowledge: Knowledge-based security for dynamic policies. In: 2012 IEEE 25th Computer Security Foundations Symposium. pp. 308–322 (2012). https://doi.org/10.1109/CSF.2012.31

[7] Biba, K.J.: Integrity considerations for secure computer systems. Tech. Rep. ESD-TR-76-372, USAF Electronic Systems Division, Bedford, MA (Apr 1977), https://ban.ai/multics/doc/a039324.pdf, (Also available through National Technical Information Service, Springfield Va., NTIS AD-A039324.)

[8] Blok, W.J., Pigozzi, D.: Algebraizable Logics (1989)

[9] Cecchetti, E., Myers, A.C., Arden, O.: Nonmalleable information flow control. In: 24th ACM Conf. on Computer and Communications Security (CCS). pp. 1875–1891. ACM (Oct 2017). https://doi.org/10.1145/3133956.3134054

[10] Chong, S., Liu, J., Myers, A.C., Qi, X., Vikram, K., Zheng, L., Zheng, X.: Secure web applications via automatic partitioning. In: 21st ACM Symp. on Operating System Principles (SOSP). pp. 31–44 (Oct 2007). https://doi.org/10.1145/1323293.1294265

[11] Chong, S., Vikram, K., Myers, A.C.: SIF: Enforcing confidentiality and integrity in web applications. In: 16th USENIX Security Symp. (Aug 2007), http://www.cs.cornell.edu/andru/papers/sif.pdf

[12] Clarkson, M.R., Schneider, F.B.: Hyperproperties. In: 21st IEEE Computer Security Foundations Symp. (CSF). pp. 51–65 (Jun 2008). https://doi.org/10.1109/CSF.2008.7

[13] Denning, D.E.: A lattice model of secure information flow. Comm. of the ACM **19**(5), 236–243 (1976). https://doi.org/10.1145/360051.360056

[14] Dolan, S.: Algebraic subtyping: Distinguished Dissertation 2017. BCS Learning & Development Ltd, Swindon, GBR (2017)

[15] Fournet, C., le Guernic, G., Rezk, T.: A security-preserving compiler for distributed programs: From information-flow policies to cryptographic mechanisms. In: 16th ACM Conf. on Computer and Communications Security (CCS). pp. 432–441 (Nov 2009), https://doi.org/10.1145/1653662.1653715

[16] Fournet, C., Rezk, T.: Cryptographically sound implementations for typed information-flow security. In: 35th ACM Symp. on Principles of Programming Languages (POPL). pp. 323–335 (Jan 2008), https://doi.org/10.1145/1328438.1328478

[17] Garg, D., Pfenning, F.: Non-interference in constructive authorization logic. In: 19th IEEE Computer Security Foundations Workshop (CSFW) (2006). https://doi.org/10.1109/CSFW.2006.18

[18] Goguen, J.A., Meseguer, J.: Security policies and security models. In: IEEE Symp. on Security and Privacy. pp. 11–20 (Apr 1982). https://doi.org/10.1109/SP.1982.10014

[19] Goguen, J.A., Meseguer, J.: Unwinding and inference control. In: IEEE Symp. on Security and Privacy. pp. 75–86 (Apr 1984). https://doi.org/10.1109/SP.1984.10019

[20] Hirsch, A.K., Amorim, P.H.A.d., Cecchetti, E., Tate, R., Arden, O.: First-order logic for flow-limited authorization. In: 2020 IEEE 33rd Computer Security Foundations Symposium (CSF). pp. 123–138 (2020). https://doi.org/10.1109/CSF49147.2020.00017

[21] Hirsch, A.K., Clarkson, M.R.: Belief semantics of authorization logic. CCS '13, Association for Computing Machinery, New York, NY, USA (2013). https://doi.org/10.1145/2508859.2516667

[22] Hirt, M., Maurer, U.M.: Player simulation and general adversary structures in perfect multiparty computation. J. Cryptol. **13**(1), 31–60 (2000). https://doi.org/10.1007/S001459910003

[23] Kozyri, E., Chong, S., Myers, A.C.: Expressing information flow properties. Foundations and Trends in Privacy and Security **3**(1), 1–102 (2022). https://doi.org/10.1561/3300000008

[24] Landauer, J., Redmond, T.: A lattice of information. In: 6th IEEE Computer Security Foundations Workshop (CSFW). pp. 65–70. IEEE Computer Society Press (Jun 1993). https://doi.org/10.1109/CSFW.1993.246638

[25] Li, N., Grosof, B.N., Feigenbaum, J.: Delegation logic: A logic-based approach to distributed authorization. ACM Transactions on Information and System Security (TISSEC) **6**(1), 128–171 (2003). https://doi.org/10.1145/605434.605438

[26] Li, P., Zhang, D.: Towards a general-purpose dynamic information flow policy (2021), https://arxiv.org/abs/2109.08096

[27] Matetic, S., Schneider, M., Miller, A., Juels, A., Capkun, S.: Delegatee: Brokered delegation using trusted execution environments. In: USENIX Security Symposium. pp. 1387–1403 (2018)

[28] Montagu, B., Pierce, B.C., Pollack, R.: A theory of information-flow labels. In: 26th IEEE Computer Security Foundations Symp. (CSF). pp. 3–17 (Jun 2013). https://doi.org/10.1109/CSF.2013.8

[29] Myers, A.C.: Mostly-Static Decentralized Information Flow Control. Ph.D. thesis, Massachusetts Institute of Technology, Cambridge, MA (Jan 1999)

[30] Myers, A.C., Liskov, B.: A decentralized model for information flow control. In: 16th ACM Symp. on Operating System Principles (SOSP). pp. 129–142 (Oct 1997). https://doi.org/10.1145/268998.266669

[31] Myers, A.C., Liskov, B.: Protecting privacy using the decentralized label model. ACM Transactions on Software Engineering and Methodology **9**(4), 410–442 (Oct 2000). https://doi.org/10.1145/363516.363526

[32] Myers, A.C., Sabelfeld, A., Zdancewic, S.: Enforcing robust declassification. In: 17th IEEE Computer Security Foundations Workshop (CSFW). pp. 172–186 (Jun 2004). https://doi.org/10.1109/CSFW.2004.9

[33] Myers, A.C., Sabelfeld, A., Zdancewic, S.: Enforcing robust declassification and qualified robustness. Journal of Computer Security **14**(2), 157–196 (2006). https://doi.org/10.3233/JCS-2006-14203

[34] Myers, A.C., Zheng, L., Zdancewic, S., Chong, S., Nystrom, N.: Jif 3.0: Java information flow (Jul 2006), http://www.cs.cornell.edu/jif, software release, http://www.cs.cornell.edu/jif

[35] Rehof, J., Mogensen, T.A.: Tractable constraints in finite semilattices. In: 3rd International Symposium on Static Analysis. pp. 285–300. No. 1145 in Lecture Notes in Computer Science, Springer-Verlag (Sep 1996). https://doi.org/10.1007/3-540-61739-6_48

[36] Ren, S., Acay, C., Myers, A.C.: An algebraic approach to asymmetric delegation and polymorphic label inference (technical report) (Apr 2025). https://doi.org/10.48550/arXiv.2504.20432

[37] Rutherford, D.E.: Introduction to Lattice Theory. Oliver and Boyd (1965)

[38] Sabelfeld, A., Myers, A.C.: Language-based information-flow security. IEEE Journal on Selected Areas in Communications **21**(1), 5–19 (Jan 2003). https://doi.org/10.1109/JSAC.2002.806121

[39] Sandhu, R.S.: Role hierarchies and constraints for lattice-based access controls. In: 4th European Symp. on Research in Computer Security (ESORICS) (Sep 1996)

[40] Simonet, V.: The Flow Caml System: documentation and user's manual. Technical Report 0282, Institut National de Recherche en Informatique et en Automatique (INRIA) (Jul 2003)

[41] Soloviev, M., Balliu, M., Guanciale, R.: Security properties through the lens of modal logic (2023). https://doi.org/10.1109/csf61375.2024.00009

[42] Stefan, D., Russo, A., Mazières, D., Mitchell, J.C.: Disjunction category labels. In: Laud, P. (ed.) Proceedings of the 16th Nordic conference on Information Security Technology for Applications. Lecture Notes in Computer Science, vol. 7161, pp. 223–239. Springer (2011). https://doi.org/10.1007/978-3-642-29615-4_16

[43] Stone, M.H.: Topological representations of distributive lattices and brouwerian logics. Časopis pro pěstování matematiky a fysiky **067**(1), 1–25 (1938). https://doi.org/10.21136/CPMF.1938.124080

[44] Swamy, N., Hicks, M., Tse, S., Zdancewic, S.: Managing policy updates in security-typed languages. In: 19th IEEE Computer Security Foundations

Workshop (CSFW). pp. 202–216 (Jul 2006). https://doi.org/10.1109/CSFW.2006.17

[45] Yao, A.C.: Protocols for secure computations. In: 23rd annual IEEE Symposium on Foundations of Computer Science. pp. 160–164 (1982). https://doi.org/10.1109/SFCS.1982.38

[46] Zagieboylo, D., Suh, G.E., Myers, A.C.: Using information flow to design an ISA that controls timing channels. In: 32nd IEEE Computer Security Foundations Symp. (CSF) (Jun 2019). https://doi.org/10.1109/CSF.2019.00026

[47] Zdancewic, S., Myers, A.C.: Robust declassification. In: 14th IEEE Computer Security Foundations Workshop (CSFW). pp. 15–23 (Jun 2001). https://doi.org/10.1109/CSFW.2001.930133

[48] Zdancewic, S., Zheng, L., Nystrom, N., Myers, A.C.: Secure program partitioning. ACM Trans. on Computer Systems **20**(3), 283–328 (Aug 2002). https://doi.org/10.1145/566340.566343

[49] Zhang, D., Myers, A.C., Vytiniotis, D., Peyton Jones, S.: SHErrLoc: A static holistic error locator. ACM Trans. on Programming Languages and Systems **39**(4), 18 (Aug 2017), http://dl.acm.org/citation.cfm?id=3121137

[50] Zheng, L., Chong, S., Myers, A.C., Zdancewic, S.: Using replication and partitioning to build secure distributed systems. In: IEEE Symp. on Security and Privacy. pp. 236–250 (May 2003). https://doi.org/10.1109/SECPRI.2003.1199340

[51] Zheng, L., Myers, A.C.: A language-based approach to secure quorum replication. In: 9th ACM SIGPLAN Workshop on Programming Languages and Analysis for Security (PLAS) (Aug 2014). https://doi.org/10.1145/2637113.2637117