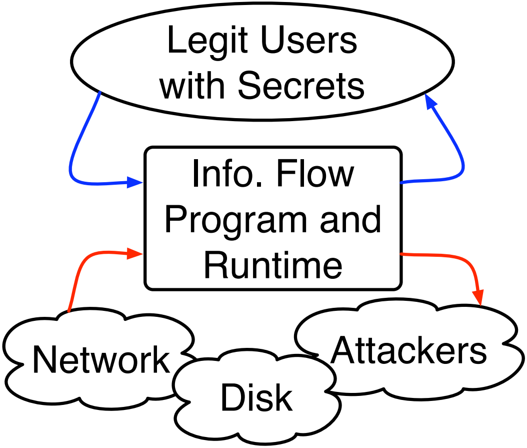
A Cryptographic Decentralized Label Model

Jeffrey A. Vaughan and Steve Zdancewic

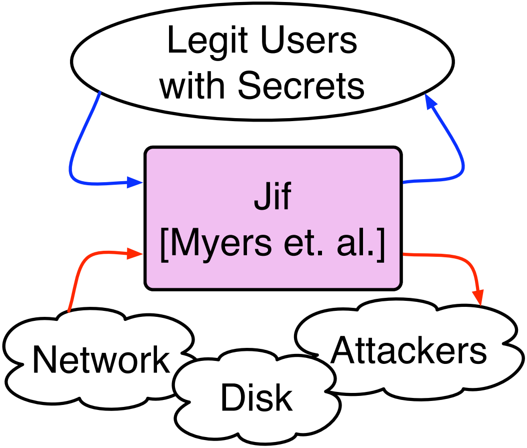
Department of Computer and Information Science
University of Pennsylvania

IEEE Security and Privacy
May 22, 2007

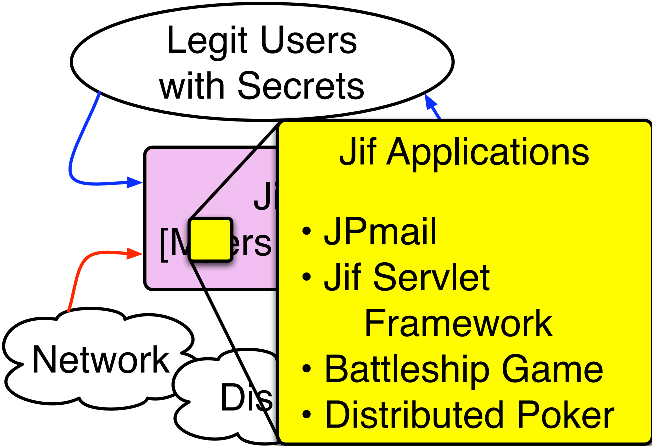
Information flow protects secrets from disclosure.



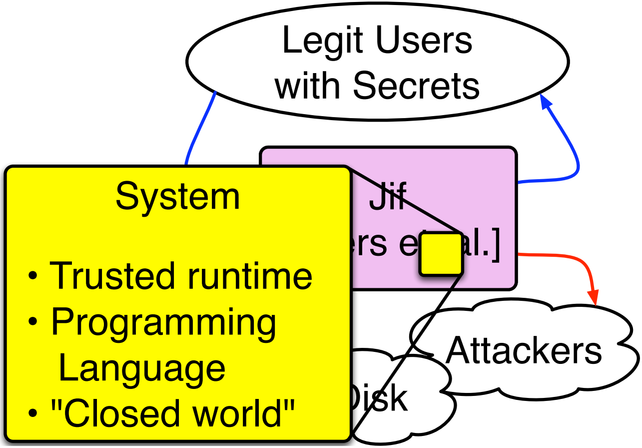
Information flow protects secrets from disclosure.



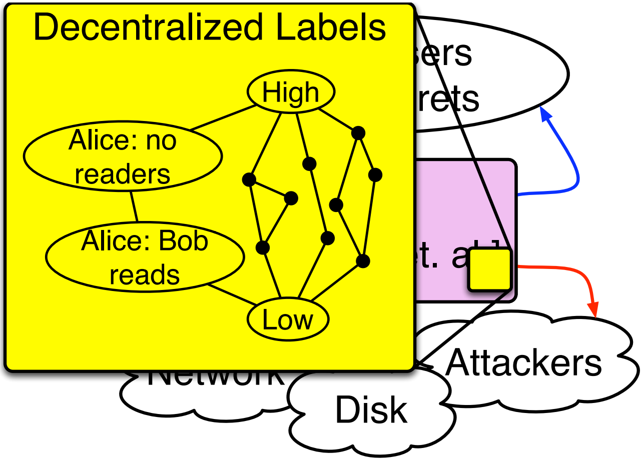
Information flow protects secrets from disclosure.



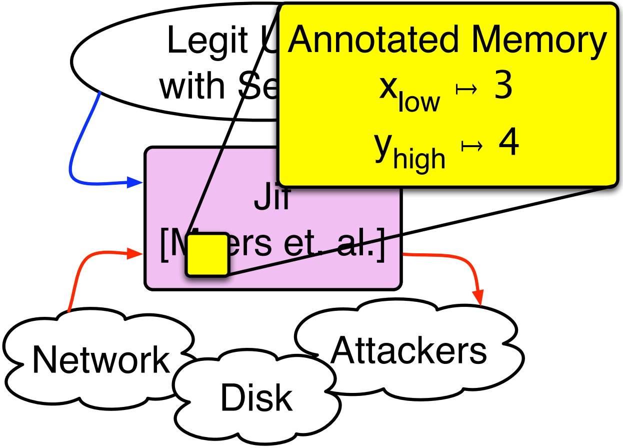
Information flow protects secrets from disclosure.



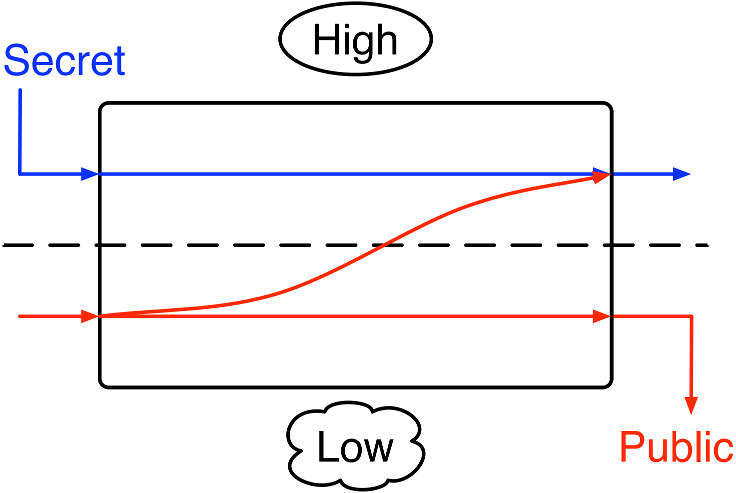
Information flow protects secrets from disclosure.



Information flow protects secrets from disclosure.

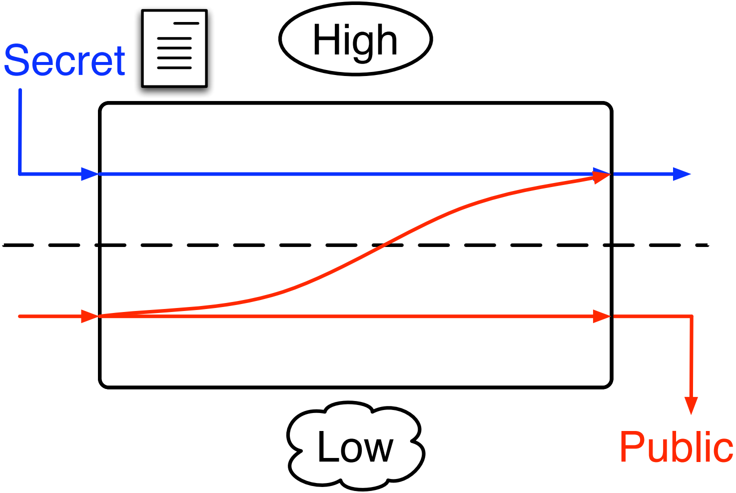


Noninterference: high inputs don't affect low outputs



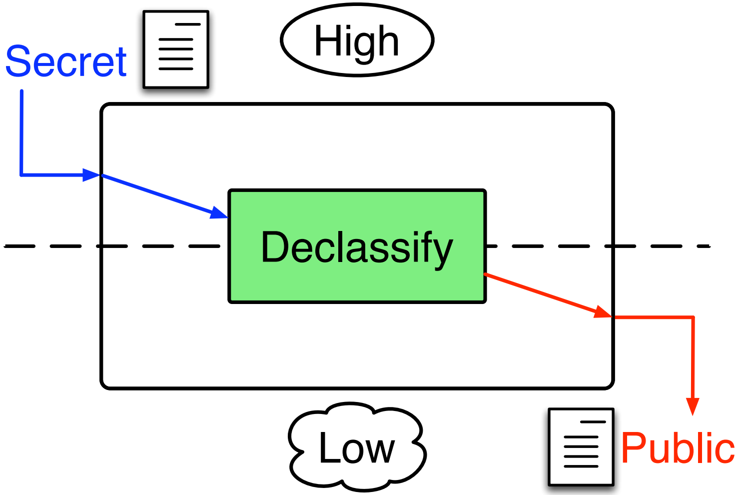
[Denning & Denning CACM '77] [Pottier & Simonet TOPLAS '03]
[Volpano, Smith, & Irvine JCS '96] [. . .]

Noninterference: high inputs don't affect low outputs



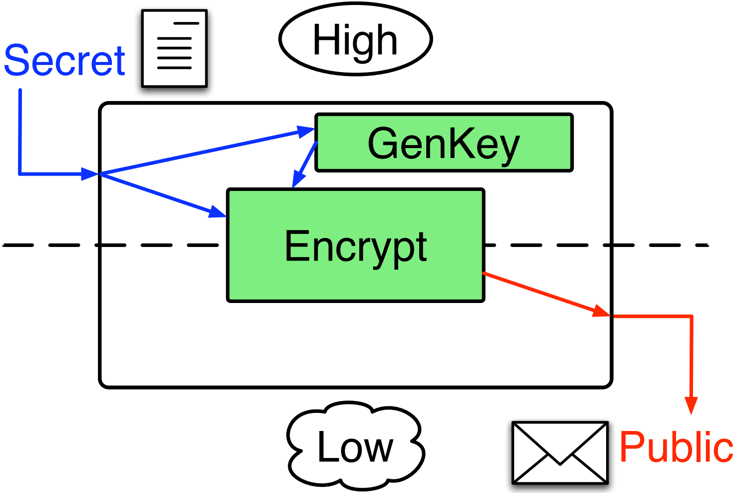
[Denning & Denning CACM '77] [Pottier & Simonet TOPLAS '03] [Volpano, Smith, & Irvine JCS '96] [. . .]

Some programs need to violate noninterference.



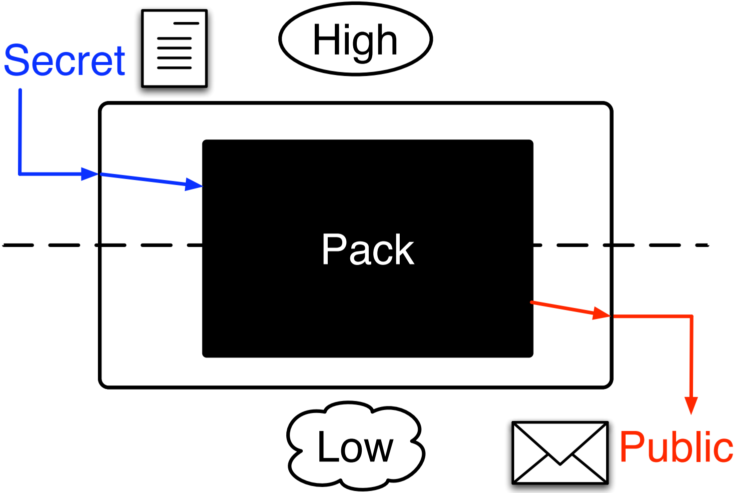
Satisfies (decentralized) robust declassification instead of noninterference [Myers & Chong CSFW '06].

Encryption can restore noninterference.

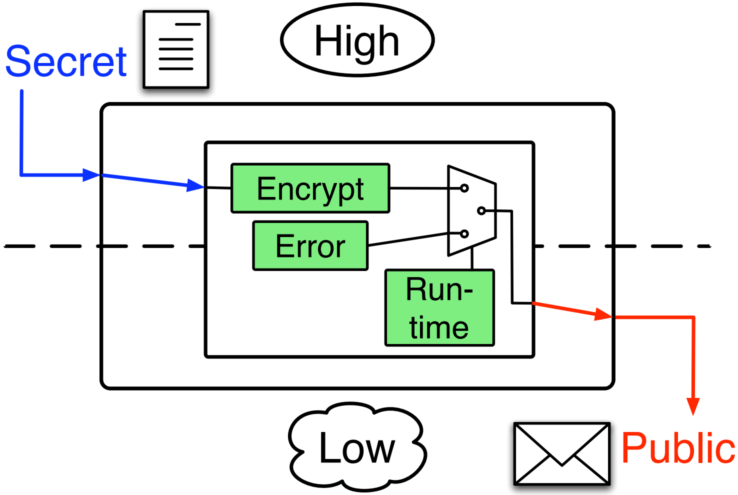


[Chothia, Duggan, & Vitek CSFW ’02] [Sumii & Pierce POPL ’04]
[Laud & Vene ACSAC ’05] [Askarov, Hedin, & Sabelfeld ISAS ’06]

Our idea: make the cryptography transparent.



Our idea: make the cryptography transparent.



Our solution: label directed implicit packing.

The Cryptographic Decentralized Label Model unifies

- a high-level, information-flow *language*,
- declarative *labels* that describe security policies,
- and cryptographic *packages* that implement policies.

Key notation

$$\begin{array}{ccccc} \text{data} & + & \text{policies} & \Rightarrow & \text{package} \\ v & + & \ell & \Rightarrow & \langle v \rangle_\ell \end{array}$$

SImp language can pack and unpack labeled data.

Definition (SImp Syntax)

types	τ	$::=$	$\texttt{int} \mid \ldots \mid \texttt{pkg}$	
values	v	$::=$	$0 \mid 1 \mid \ldots$	
		$\mid$	$\langle v \rangle_\ell$	package
expressions	e	$::=$	$\ldots$	
		$\mid$	$\texttt{pack}\ e\ \texttt{at}\ \ell$	package intro
		$\mid$	$\texttt{unpack}\ e\ \texttt{as}\ \tau\{\ell\}$	package elim

Packages may be constructed and analyzed according to ℓ.

SImp can implement a simple messaging system.

Example

```
text: string{high}        dest: string{low}
out: pkg{low}             in: pkg{low}

text := readLine()
match (pack text at {high}) with
  ok(p)  => out := p; send(out)
  error  => skip

            ⋮

in := receive()
match (unpack in as text{high})
  ok(t) => text := t;  printLine(text)
  error => skip
```

Pack succeeds iff runtime has sufficient authority.

The SImp runtime contains

- a memory, M, and
- an authority, $\overline{p}$.

Evaluation Model

$$\frac{\text{precondition}}{\text{runtime state} \vdash \text{from} \rightarrow \text{to}}$$

Definition (Pack Evaluation)

$$\frac{\overline{p} \textit{ writes } \ell}{\overline{p}; M \vdash \texttt{pack } v \texttt{ at } \ell \rightarrow \texttt{ok}(\langle v \rangle_\ell)} \quad \textsc{E-Pack-Ok}$$

$$\frac{\neg(\overline{p} \textit{ writes } \ell)}{\overline{p}; M \vdash \texttt{pack } v \texttt{ at } \ell \rightarrow \texttt{error}} \quad \textsc{E-Pack-Fail}$$

Without enough keys, it is *infeasible* for the runtime to perform these operations.

Unpacking also performs dynamic checks.

Definition Fragment (Unpack Evaluation)

$$\frac{\overline{p} \textit{ reads } \ell \qquad \ell_0 \leq \ell \qquad \vdash v_0 : \tau}{\overline{p}; M \vdash \texttt{unpack } \langle v_0 \rangle_{\ell_0} \texttt{ as } \tau\{\ell\} \rightarrow \texttt{ok}(v_0)}$$

Checks ensure

- cryptographic feasibility
- information flow
- type safety (values have correct shapes)

Unpacking also performs dynamic checks.

Definition Fragment (Unpack Evaluation)

$$\frac{\overline{p}\ \mathit{reads}\ \ell \qquad \ell_0 \leq \ell \qquad \vdash v_0 : \tau}{\overline{p}; M \vdash \texttt{unpack}\ \langle v_0 \rangle_{\ell_0}\ \texttt{as}\ \tau\{\ell\} \rightarrow \texttt{ok}(v_0)}$$

Checks ensure

- cryptographic feasibility
- information flow
- type safety (values have correct shapes)

Unpacking also performs dynamic checks.

Definition Fragment (Unpack Evaluation)

$$\frac{\overline{p} \textit{ reads } \ell \qquad \ell_0 \leq \ell \qquad \vdash v_0 : \tau}{\overline{p}; M \vdash \texttt{unpack } \langle v_0 \rangle_{\ell_0} \texttt{ as } \tau\{\ell\} \rightarrow \texttt{ok}(v_0)}$$

Checks ensure

- cryptographic feasibility
- information flow
- type safety (values have correct shapes)

Unpacking also performs dynamic checks.

Definition Fragment (Unpack Evaluation)

$$\frac{\overline{p} \textit{ reads } \ell \qquad \ell_0 \leq \ell \qquad \vdash v_0 : \tau}{\overline{p}; M \vdash \texttt{unpack } \langle v_0 \rangle_{\ell_0} \texttt{ as } \tau\{\ell\} \rightarrow \texttt{ok}(v_0)}$$

Checks ensure

- cryptographic feasibility
- information flow
- type safety (values have correct shapes)

Unpacking also performs dynamic checks.

Definition Fragment (Unpack Evaluation)

$$\frac{\overline{p}\ \mathit{reads}\ \ell \qquad \ell_0 \leq \ell \qquad \vdash v_0 : \tau}{\overline{p}; M \vdash \texttt{unpack}\ \langle v_0 \rangle_{\ell_0}\ \texttt{as}\ \tau\{\ell\} \rightarrow \texttt{ok}(v_0)}$$

Checks ensure

- cryptographic feasibility
- information flow
- type safety (values have correct shapes)

Is failure to pack a covert channel?

Example

```
h: bool{high},    l: bool{low},    v: pkg{?}

if h then
  v := pack 0 at low
else
  v := pack 0 at high
          ⋮
match (unpack v as bool{low}) with
  ok(_) => l := true
  error => l := false
```

Is failure to pack a covert channel?

Example

```
h: bool{high},    l: bool{low},    v: pkg{?}

if h then
  v := pack 0 at low
else
  v := pack 0 at high
          ⋮
match (unpack v as bool{low}) with
  ok(_) => l := true
  error => l := false
```

Rule: High guard requires variables assigned in branches are high.

Constraints: v is high

Is failure to pack a covert channel?

Example

```
h: bool{high},    l: bool{low},    v: pkg{?}

if h then
  v := pack 0 at low
else
  v := pack 0 at high
          ⋮
match (unpack v as bool{low}) with ⇐
  ok(_) => l := true
  error => l := false
```

Rule: Unpack does not declassify secret data.

Constraints: v is high; v is low

Is failure to pack a covert channel?

Example

```
h: bool{high},    l: bool{low},    v: pkg{?}

if h then
  v := pack 0 at low
else
  v := pack 0 at high
          ⋮
match (unpack v as bool{low}) with
  ok(_) => l := true
  error => l := false
```

Constraints: v is high; v is low; high = low

Is failure to pack a covert channel?

Example

```
h: bool{high},    l: bool{low},    v: pkg{?}

if h then
  v := pack 0 at low
else
  v := pack 0 at high
          ⋮
match (unpack v as bool{low}) with
  ok(_) => l := true
  error => l := false
```

$\therefore$ reject statically

Pack provides a limited declassify, unpack an endorse.

Definition Fragment (Pack Security Typing, 1st attempt)

If

- *e has label* ℓ_e

then

`pack` e `at` ℓ_e *has label* ℓ.

Definition Fragment (Unpack Security Typing, 1st attempt)

If

- *e has label* ℓ_e*, and*
- *labels* ℓ *and* ℓ_e *are equal,*

then

`unpack` e `as` $\tau\{\ell\}$ *has label* ℓ.

Pack provides a limited declassify, unpack an endorse.

Definition Fragment (Pack Security Typing)

If

- *e has label ℓ_e, and*
- *labels ℓ and ℓ_e have equal integrity components,*

then

`pack` e `at` ℓ_e *has label* ℓ.

Definition Fragment (Unpack Security Typing)

If

- *e has label ℓ_e, and*
- *labels ℓ and ℓ_e have equal confidentiality components,*

then

`unpack` e `as` $\tau\{\ell\}$ *has label* ℓ.

Can we “cast away” security labels?

Example

```
h: bool{high},    l: bool{low},    v: pkg{low}

v := pack h at {high}
          ⋮
match (unpack v as bool{low}) with
  ok(true)  => l := true
  ok(false) => l := false
  error     => skip
```

Can we "cast away" security labels?

Example

```
h: bool{high},    l: bool{low},    v: pkg{low}

v := pack h at {high}
           ⋮
match (unpack v as bool{low}) with
  ok(true)  => l := true
  ok(false) => l := false
  error     => skip
```

State: $h = true$

Can we "cast away" security labels?

Example

```
h: bool{high},    l: bool{low},    v: pkg{low}

v := pack h at {high}      ⇐
            ⋮
match (unpack v as bool{low}) with
  ok(true)  => l := true
  ok(false) => l := false
  error     => skip
```

State: $h = \textit{true}$; $v = \langle \textit{true} \rangle_{high}$

Can we "cast away" security labels?

Example

```
h: bool{high},    l: bool{low},    v: pkg{low}

v := pack h at {high}
          ⋮
match (unpack v as bool{low}) with ⇐
  ok(true)  => l := true
  ok(false) => l := false
  error     => skip
```

Check: *high* $\leq$ *low*

Can we "cast away" security labels?

Example

```
h: bool{high},    l: bool{low},   v: pkg{low}

v := pack h at {high}
           ⋮
match (unpack v as bool{low}) with ⇐
  ok(true)  => l := true
  ok(false) => l := false
  error     => skip
```

∴ unpacking fails

Can we “cast away” security labels?

Example

```
h: bool{high},    l: bool{low},    v: pkg{low}

v := pack h at {high}
          ⋮
match (unpack v as bool{low}) with
  ok(true)  => l := true
  ok(false) => l := false
  error     => skip         ⇐
```

State: $h = \textit{true}$; $v = \langle \textit{true} \rangle_{\textit{high}}$

Can we “cast away” security labels?

Example

```
h: bool{high},    l: bool{low},   v: pkg{low}

v := pack h at {high}
           ⋮
match (unpack v as bool{low}) with
  ok(true)  => l := true
  ok(false) => l := false
  error     => skip
```

Conclusion: not a leak

Evaluation respects noninterference.

Property ($M_1 \cong_\ell M_2$)

Memories M_1 and M_2 are equivalent to an observer with power ℓ, if the observer cannot distinguish the memories.

Example

M_1
$x_{low} \mapsto 2$
$y_{low} \mapsto \langle 3 \rangle_{high}$
$z_{high} \mapsto 3$

M_2
$x_{low} \mapsto 2$
$y_{low} \mapsto \langle 4 \rangle_{high}$
$z_{high} \mapsto 4$

$$M_1 \cong_{low} M_2$$

$$M_1 \ncong_{high} M_2$$

Property (Noninterference)

$$M_1 \xrightarrow{\text{evaluation}} M_1'$$

$$M_2 \xrightarrow{\text{evaluation}} M_2'$$

$$M_1 \cong_\ell M_2 \implies M_2' \cong_\ell M_1'$$

SImp is parameterized by a *security lattice*.

Definition Fragment (Security lattice)

A lattice whose elements are composed of orthogonal confidentiality and integrity components is a security lattice.

Example

$$high = \{private!tainted\}$$

$$\{private!trusted\} \qquad \{public!tainted\}$$

$$low = \{public!trusted\}$$

Decentralized labels specify security policies.

$$\ell = \{\mathsf{Alice}: \ \mathsf{Bob} \,!\, \mathsf{Charlie}; \quad \mathsf{Bob}: \ \mathsf{Alice} \,!\, \mathsf{Charlie}, \mathsf{Dave}\}$$

- A label is a list of policies.
- A policy consists of
  - an owner (who may distrust other owners),
  - a reader set, and
  - a writer set.

Property ($\overline{p}$ *reads* ℓ)

Principal set $\overline{p}$ reads ℓ when each owner in ℓ permits some member of $\overline{p}$ to read.

Example

	reads	writes
Alice	✓	x
Dave	x	x
{Alice, Dave}	x	✓

Labeling is a variant of Myers and Liskov's DLM [TOSEM '00].

Decentralized labels specify security policies.

$$\ell = \{\text{Alice} : \text{Bob} \, ! \, \text{Charlie}; \; \text{Bob} : \text{Alice} \, ! \, \text{Charlie}, \text{Dave}\}$$

- A label is a list of policies.
- A policy consists of
  - an owner (who may distrust other owners),
  - a reader set, and
  - a writer set.

Property ($\overline{p}$ *reads* ℓ)

Principal set $\overline{p}$ reads ℓ when each owner in ℓ permits some member of $\overline{p}$ to read.

Example

	reads	writes
Alice	✓	x
Dave	x	x
{Alice, Dave}	x	✓

Labeling is a variant of Myers and Liskov's DLM [TOSEM '00].

Decentralized labels specify security policies.

$$\ell = \{\text{Alice}: \text{Bob} ! \text{Charlie}; \quad \text{Bob}: \text{Alice} ! \text{Charlie}, \text{Dave}\}$$

- A label is a list of policies.
- A policy consists of
  - an owner (who may distrust other owners),
  - a reader set, and
  - a writer set.

Property ($\overline{p}$ *reads* ℓ)

Principal set $\overline{p}$ reads ℓ when each owner in ℓ permits some member of $\overline{p}$ to read.

Example

	reads	writes
Alice	✓	x
Dave	x	x
{Alice, Dave}	x	✓

Labeling is a variant of Myers and Liskov's DLM [TOSEM '00].

Decentralized labels specify security policies.

$$\ell = \{\text{Alice}: \ \text{Bob} \,!\, \text{Charlie};$$
$$\text{Bob}: \ \text{Alice} \,!\, \text{Charlie}, \text{Dave}\}$$

- A label is a list of policies.
- A policy consists of
  - an owner (who may distrust other owners),
  - a reader set, and
  - a writer set.

Property ($\overline{p}$ *reads* ℓ)

Principal set $\overline{p}$ reads ℓ when each owner in ℓ permits some member of $\overline{p}$ to read.

Example

	reads	writes
Alice	✓	x
Dave	x	x
{Alice, Dave}	x	✓

Labeling is a variant of Myers and Liskov's DLM [TOSEM '00].

Decentralized labels specify security policies.

$$\ell = \{\text{Alice}: \text{Bob} ! \text{Charlie}; \quad \text{Bob}: \text{Alice} ! \text{Charlie}, \text{Dave}\}$$

- A label is a list of policies.
- A policy consists of
  - an owner (who may distrust other owners),
  - a reader set, and
  - a writer set.

Property ($\overline{p}$ *reads* ℓ)

Principal set $\overline{p}$ reads ℓ when each owner in ℓ permits some member of $\overline{p}$ to read.

Example

	reads	writes
Alice	✓	x
Dave	x	x
{Alice, Dave}	x	✓

Labeling is a variant of Myers and Liskov's DLM [TOSEM '00].

Decentralized labels specify security policies.

$$\ell = \{\text{Alice}: \text{Bob}\,!\,\text{Charlie};$$
$$\text{Bob}: \text{Alice}\,!\,\text{Charlie, Dave}\}$$

- A label is a list of policies.
- A policy consists of
  - an owner (who may distrust other owners),
  - a reader set, and
  - a writer set.

Property ($\overline{p}$ *reads* ℓ)

Principal set $\overline{p}$ reads ℓ when each owner in ℓ permits some member of $\overline{p}$ to read.

Example

	reads	writes
Alice	✓	x
Dave	x	x
{Alice, Dave}	x	✓

Labeling is a variant of Myers and Liskov's DLM [TOSEM '00].

Decentralized labels specify security policies.

$$\ell = \{\text{Alice}:\ \text{Bob}\,!\,\text{Charlie};$$
$$\text{Bob}:\ \text{Alice}\,!\,\text{Charlie}, \text{Dave}\}$$

- A label is a list of policies.
- A policy consists of
  - an owner (who may distrust other owners),
  - a reader set, and
  - a writer set.

Property ($\overline{p}$ *reads* ℓ)

Principal set $\overline{p}$ reads ℓ when each owner in ℓ permits some member of $\overline{p}$ to read.

Example

	reads	writes
Alice	✓	x
Dave	x	x
{Alice, Dave}	x	✓

Labeling is a variant of Myers and Liskov's DLM [TOSEM '00].

Labels and packages need meaning outside of SImp.

- Cryptographic assumptions
  - Each principal is mapped to a well-known public key.
  - Cryptographic functions follow the Dolev-Yao model.
- Goals of interpretation
  - Package confidentiality protects data from eavesdroppers.
  - Package integrity protects the program from data.
  - Packages can created and consumed offline.

Packages compile to cryptographic messages.

Example (Compile $\langle 42 \rangle_{\{\text{Alice: Bob!}\}}$)

1. Generate fresh key pairs (R^+, R^-) and (W^+, W^-).
2. Let $payload = sign(W^-, \{\!|42|\!\}_{R^+})$
3. Let $seal = sign(\text{Alice}, [\texttt{"\{Alice: Bob!\}"}, R^+, W^+, \{\!|R^-|\!\}_{K^+_{\text{Alice}}}, \{\!|R^-|\!\}_{K^+_{\text{Bob}}}, \{\!|W^-|\!\}_{K^+_{\text{Alice}}}])$
4. Return $package = (seal, payload)$

- R^- is a read capability.
- W^- is a write capability.

Package compilation is adequate.

Property ($m_1 \cong_\ell m_2$)

Say $m_1 \cong_\ell m_2$ when messages m_1 and m_2 reveal only equivalent information to Dolev-Yao observers weaker than ℓ.

Lemma (Adequacy for Values)

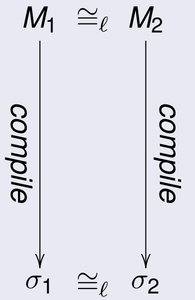
If

$$v_1 \cong_\ell v_2$$

then

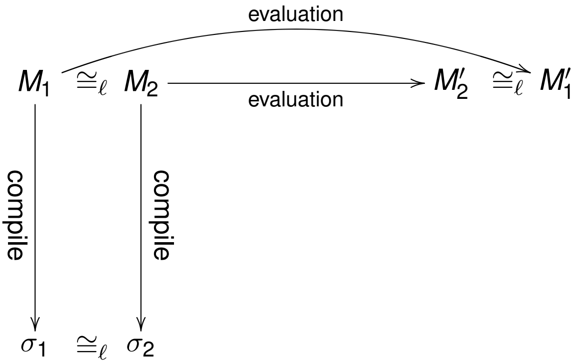
$$compile(v_1) \cong_\ell compile(v_2).$$

Corollary



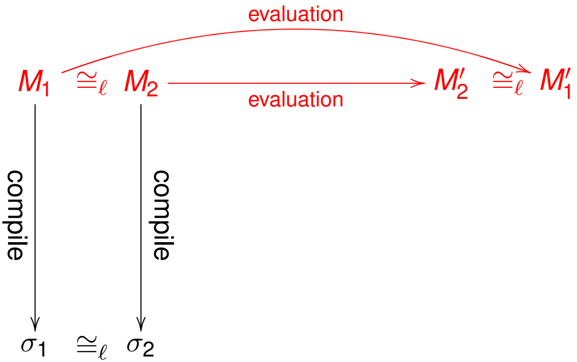
Read the paper for more results.

- Today we discussed noninterference and adequacy.
- In the paper we consider feasibility.



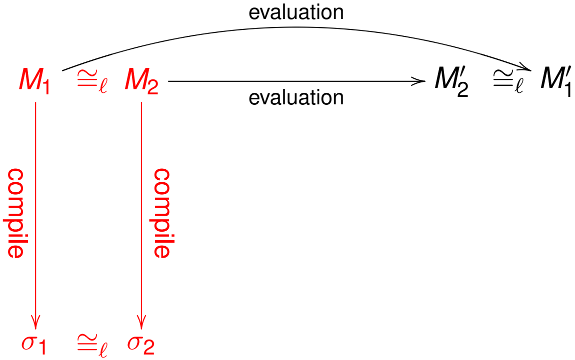
Read the paper for more results.

- Today we discussed noninterference and adequacy.
- In the paper we consider feasibility.



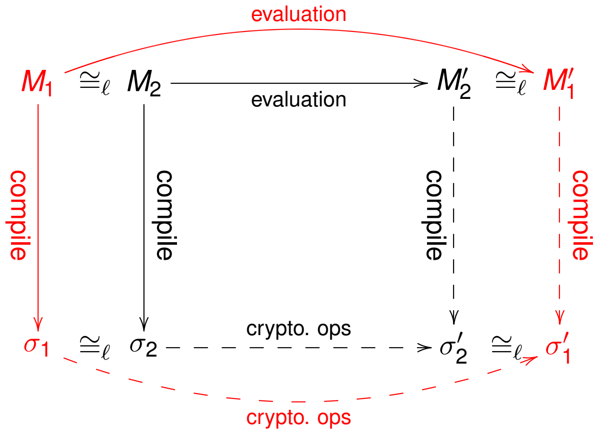
Read the paper for more results.

- Today we discussed noninterference and adequacy.
- In the paper we consider feasibility.



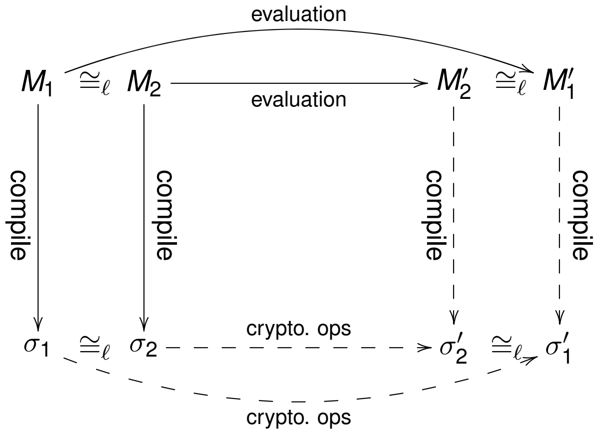
Read the paper for more results.

- Today we discussed noninterference and adequacy.
- In the paper we consider feasibility.



Read the paper for more results.

- Today we discussed noninterference and adequacy.
- In the paper we consider feasibility.



Take home messages

SImp explores a new space in information flow languages with

- declarative policies implemented by a cryptographic mechanism,
- a strong noninterference property,
- and a rich, structural label language.

Appendix

- Typing Rules
- Future Work
- Package Uniqueness
- DLM Comparison
- Expression Noninterference Statement
- Command Noninterference Statement
- Adequacy Statements
- Feasibility Statement
- Cryptographic Operations

Pack provides a limited declassify, unpack an endorse.

Definition Fragment

$$\frac{\Theta; \Gamma \vdash e : \tau\{\ell_e\} \qquad I(\ell_e) = I(\ell)}{\Theta; \Gamma \vdash \texttt{pack}\ e\ \texttt{at}\ \ell_e : (\texttt{pkg} + \texttt{error})\{\ell\}}$$

$$\frac{\Theta; \Gamma \vdash e : \texttt{pkg}\{\ell_e\} \qquad C(\ell_e) = C(\ell)}{\Theta; \Gamma \vdash \texttt{unpack}\ e\ \texttt{as}\ \tau\{\ell\} : (\tau + \texttt{error})\{\ell\}}$$

These rules are safe because of the dynamic checks.

This is just the beginning.

Further research questions:

- Can homomorphic encryption be used for computation within packages?
- How can we compile alternative label models?
  - “share semantics”
  - uniqueness labels
- How does package upgrading and downgrading interact with cryptography?

A explicit non-goal: package uniqueness.

- Replay attacks (vs. legitimate uses of persistence) are best detected at higher levels of abstraction.
- Uniqueness checks appear to require interactive protocols.
- Resolving these challenges would be interesting future work.

Our DLM is a variant of Myers and Liskov's original.

Example

$$\ell = \{\mathsf{Alice}: \ \mathsf{Charlie}\,!\,\emptyset;\ \mathsf{Bob}: \ \mathsf{Dave}\,!\,\emptyset\}$$

- Here:

$$\vdash \{\mathsf{Charlie}, \mathsf{Dave}\}\ \textit{reads}\ \ell$$

- Myers and Liskov:

$$\not\vdash \{\mathsf{Dave}, \mathsf{Charlie}\}\ \textit{reads}\ \ell$$

We don't consider an explicit *acts-for-hierarchy*.

- It should work technically but is orthogonal.
- Intuitively, principal sets "act for" component principals.
- Key difference:
  - Myers and Liskov: Calculate readers, then close under acts-for.
  - Here: Close under acts-for, then calculate readers.

Formal statement of expression noninterference.

Theorem (Expression noninterference)

If

- $\Theta \vdash M_1 \ OK$, $\Theta \vdash M_2 \ OK$ *and* $\Theta \vdash M_1 \cong_\ell M_2$
- $\Theta; \cdot \vdash e_1 : \tau\{\ell_e\}$ *and* $e_1 \cong_\ell e_2$ *where* $\ell_e \leq \ell$
- $\overline{p}; M_1 \vdash e_1 \longrightarrow^* v_1$ *and* $\overline{p}; M_2 \vdash e_2 \longrightarrow^* v_2$

then $v_1 \cong_\ell v_2$.

Command evaluation respects noninterference.

Theorem (Command Noninterference)

If

- $\Theta \vdash M_1$ *OK*, $\Theta \vdash M_2$ *OK and* $\Theta \vdash M_1 \cong_\ell M_2$
- $pc; \Theta; \cdot \vdash c_1$ *and* $c_1 \cong_\ell c_2$
- $\overline{p} \vdash \langle M_1, c_1 \rangle \longrightarrow^* \langle M_1', \texttt{skip} \rangle$ *and*
  $\overline{p} \vdash \langle M_2, c_2 \rangle \longrightarrow^* \langle M_2', \texttt{skip} \rangle$

then $\Theta \vdash M_1' \cong_\ell M_2'$.

Adequacy theorems: compilation is secret preserving.

Lemma (Adequacy of Value Translation)

If $v_1 \cong_\ell v_2$ *and* $\overline{\kappa}$ *is fresh then* $\mathrm{V}[\![v_1]\!]_{\overline{\kappa}} \cong_\ell \mathrm{V}[\![v_2]\!]_{\overline{\kappa}}$.

Corollary (Adequacy of Memory Translation)

If $\Theta \vdash M_1 \cong_\ell M_2$ *and* $\overline{\kappa}$ *is fresh then* $\mathrm{M}[\![M_1]\!]^{\Theta}_{\overline{\kappa}} \cong_\ell \mathrm{M}[\![M_2]\!]^{\Theta}_{\overline{\kappa}}$.

Realizable operations simulate SImp evaluation.

Theorem (Feasibility)

If

- $\Theta \vdash M\ OK$
- $pc; \Theta; \Gamma \vdash c$
- $\overline{p}$ *reads pc and* $\overline{p}$ *writes pc,*
- $\overline{p} \vdash \langle M, c \rangle \longrightarrow \langle M', c' \rangle$

then

$\exists \overline{\kappa}_3, \overline{\kappa}_4.\ \vdash \mathrm{M}[\![M]\!]^{\Theta}_{\overline{\kappa}_1} \cup \mathit{state}(\overline{\kappa}_2, \overline{p}, c) \longrightarrow^* \mathrm{M}[\![M']\!]^{\Theta}_{\overline{\kappa}_3} \cup \mathit{state}(\overline{\kappa}_4, \overline{p}, c)$.

Only some operations are cryptographically realizable.

Definition

$$\frac{}{\vdash \sigma \rightarrow \sigma, \mathit{knows}\ (K_\kappa^+, K_\kappa^-)}\ \textsc{cs-Fresh} \quad \kappa \text{ fresh}$$

$$\textsc{cs-Derive}\ \frac{\sigma \vdash_d m}{\vdash \sigma \rightarrow \sigma, \mathit{knows}\ m} \qquad \textsc{cs-Forget}\ \frac{\sigma' \subseteq \sigma}{\vdash \sigma \rightarrow \sigma'}$$

$$\textsc{cs-Compute}\ \frac{\sigma \vdash_d \texttt{"i}_1\texttt{"} \qquad \sigma \vdash_d \texttt{"i}_2\texttt{"}}{\vdash \sigma \rightarrow \sigma, \texttt{"i}_3\texttt{"}} \text{ where } i_3 = i_1 + i_2$$