

# A Cryptographic Decentralized Label Model

Daniel K. Lee  
15-819

Fall, 2007

# A Cryptographic Decentralized Label Model

“A Cryptographic Decentralized Label Model”

Jeffrey Vaughan and Steve Zdancewic

*IEEE Symposium on Security and Privacy 2007*

# Simp Language

- ▶ Primitives for enforcing information flow policies, including a restricted form of cryptographic packaging.
- ▶ Similar to pack/unpack of existentials.
- ▶ Requires run-time checks to ensure security of program.

# SImp Cryptographic Implementation

- ▶ Paper gives a translation from SImp language values to cryptographic messages.
- ▶ Depends on the structure of labels used to define security policies, based on the *decentralized label model*.
- ▶ A noninterference result is shown for SImp.
- ▶ Soundness for cryptographic implementation: a Dolev-Yao attacker cannot determine contents of a package without possessing the appropriate keys.

# Policies

$o : \bar{r}!\bar{w}$

- ▶ Labels are lists of policies
- ▶ Policy owner  $o$  certifies that any principal in  $\bar{r}$  can read, any principal in  $\bar{w}$  can write.

# Security Lattice Properties

- ▶ Labels,  $\ell$ , denote members of a lattice.
- ▶  $C(\ell)$  returns a label with  $\ell$ 's confidentiality policy and least restrictive integrity policy.
- ▶  $I(\ell)$  returns a label with  $\ell$ 's least restrictive confidentiality policy and integrity policy.

# Security Lattice Properties

$$\ell = C(\ell) \sqcup I(\ell)$$

$$C(I(\ell)) = \perp$$

$$I(C(\ell)) = \perp$$

$$C(C(\ell)) = C(\ell)$$

$$I(I(\ell)) = I(\ell)$$

$C$  and  $I$  are idempotent.

# Security Lattice Properties

$C$  and  $I$  are assumed to be monotone.

$$\ell \leq \ell' \Leftrightarrow C(\ell) \leq C(\ell') \wedge I(\ell) \leq I(\ell')$$

# Labels

$$l = \{o_1 : \bar{r}_1! \bar{w}_1; o_2 : \bar{r}_2! \bar{w}_2\}$$

Definition of  $l(o)$

$$l(o_1) = (\bar{r}_1, \bar{w}_1)$$

$$l(o_2) = (\bar{r}_2, \bar{w}_2)$$

Otherwise,  $l(o) = (\mathcal{P}, \emptyset)$

$l(o_1).C$  gives  $r_1$  and  $l(o_2).I$  gives  $w_2$

# Labels

- ▶  $l_1 \leq l_2$  holds when
$$\forall o. l_2(o).C \subseteq l_1(o).C \wedge l_1(o).I \subseteq l_2(o).I$$
- ▶  $\bar{p}$  reads  $l$  iff  $\forall o. \exists p \in \bar{p}. p \in \{o\} \cup l(o).I$
- ▶  $\bar{p}$  writes  $l$  iff  $\forall o. \exists p \in \bar{p}. p \in \{o\} \cup l(o).C$ 
  - ▶ Intuitively,  $\bar{p}$  can write when every owner permits at least one member of  $\bar{p}$  to write.

**Lemma 1.** *The DLM is an instance of a security lattice.*

# SImp Syntax

Types	$\tau$	::=	<b>int</b>   $\tau_1 + \tau_2$   <b>pkg</b>
Integers	$i$	::=	...   -1   0   1   ...
Values	$v$	::=	$i$   <b>inl</b> $v$   <b>inr</b> $v$   $\langle v \rangle_e$
Expressions	$e$	::=	$i$   $a$   $x$   <b>inl</b> $v$   <b>inr</b> $v$   $e_1 + e_2$   $\langle v \rangle_e$   <b>pack</b> $e$ <b>at</b> $l$   <b>unpack</b> $e$ <b>as</b> $\tau\{l\}$   ...
Commands	$c$	::=	<b>skip</b>   $x := e$   $c_1; c_2$   <b>while</b> $e$ <b>do</b> $c$   <b>case</b> $e$ <b>of</b> $a_1 \Rightarrow c_1$   $a_2 \Rightarrow c_2$

# Packing

The package  $\langle v \rangle_\ell$  is intended to have several properties.

1.  $v$  must only be read by programs with sufficient authority to read  $\ell$ .
2.  $v$  must be kept confidential in accordance with  $C(\ell)$ .
3.  $v$  must only be written to by programs with authority to write to  $\ell$ .
4.  $v$  must have only been influenced by data with integrity greater than  $I(\ell)$ .

# Booleans and Errors

Booleans and errors encoded with the following abbreviations:

```
bool = int + int
truei = inl i
falsei = inr i
error = int
insufficientAuth = 0
illegalFlow = 1
typeMismatch = 2
```

# Booleans and Errors

- ▶ Indices of `true` and `false` are usually unimportant and omitted.
- ▶ `if` statements encoded with `case` statements in the standard way.

# Expression Evaluation

- ▶ Programs are run with the authority of some set of principals.
- ▶ Expressions can read memory.

$$\frac{M(x) = v}{\bar{p}; M \vdash x \rightarrow v} \text{EE - Loc}$$

# Expression Evaluation

$$\frac{\bar{p}; M \vdash e \rightarrow e'}{\bar{p}; M \vdash \text{inl } e \rightarrow \text{inl } e'} \text{EE-Inl}$$

$$\frac{\bar{p}; M \vdash e \rightarrow e'}{\bar{p}; M \vdash \text{inr } e \rightarrow \text{inr } e'} \text{EE-Inr}$$

# Expression Evaluation

$$\frac{\bar{p}; M \vdash e_1 \rightarrow e'_2}{\bar{p}; M \vdash e_1 + e_2 \rightarrow e'_1 + e'_2} \text{EE-Plus-Struct1}$$

$$\frac{\bar{p}; M \vdash e \rightarrow e'}{\bar{p}; M \vdash v + e \rightarrow v + e'} \text{EE-Plus-Struct2}$$

$$\frac{i_3 = i_1 + i_2}{\bar{p}; M \vdash i_1 + i_2 \rightarrow i_3} \text{EE-Plus}$$

# Expression Evaluation

$$\frac{\bar{p} \text{ writes } \ell}{\bar{p}; M \vdash \text{pack } v \text{ at } \ell \rightarrow \text{inl } \langle v \rangle_{\ell}} \text{EE-Pack-OK}$$

$$\frac{\neg(\bar{p} \text{ writes } \ell)}{\bar{p}; M \vdash \text{pack } v \text{ at } \ell \rightarrow \text{inr insufficientAuth}} \text{EE-Pack-Fail}$$

# Expression Evaluation

$$\frac{\bar{p}; M \vdash e \rightarrow e'}{\bar{p}; M \vdash \text{unpack } e \text{ as } \tau\{\ell\} \rightarrow \text{unpack } e' \text{ as } \tau\{\ell\}} \text{EE-Unpack-Struct}$$

$$\frac{\neg(\bar{p} \text{ reads } \ell)}{\bar{p}; M \vdash \text{unpack } \langle v_0 \rangle_{\ell_0} \text{ as } \tau\{\ell\} \rightarrow \text{inr insufficientAuth}} \text{EE-Unpack-}$$

$$\frac{\bar{p} \text{ reads } \ell \quad \ell_0 \not\leq \ell}{\bar{p}; M \vdash \text{unpack } \langle v_0 \rangle_{\ell_0} \text{ as } \tau\{\ell\} \rightarrow \text{inr illegalFlow}} \text{EE-Unpack-Fail2}$$

# Expression Evaluation

$$\frac{\bar{p} \text{ reads } l \quad l_0 \leq l \quad \not\vdash v_0 : \tau}{\bar{p}; M \vdash \text{unpack } \langle v_0 \rangle_{l_0} \text{ as } \tau\{l\} \rightarrow \text{inr illegalFlow}} \text{EE-Unpack-Fail3}$$

$$\frac{\bar{p} \text{ reads } l \quad l_0 \leq l \quad \vdash v_0 : \tau}{\bar{p}; M \vdash \text{unpack } \langle v_0 \rangle_{l_0} \text{ as } \tau\{l\} \rightarrow \text{inl } v_0} \text{EE-Unpack-OK}$$

# Command Evaluation

$pc$  indicates the highest label assigned to locations or variables which may have influenced control flow of the command.

$$\frac{\bar{p}; M \vdash^* e \rightarrow v}{\bar{p} \vdash \langle M, x := e \rangle \rightarrow \langle M[x \mapsto v], \text{skip} \rangle} \text{EC-Assign}$$

$$\frac{}{\bar{p} \vdash \langle M, \text{skip}; c \rangle \rightarrow \langle M, c \rangle} \text{EC-Seq-Skip}$$

$$\frac{\bar{p} \vdash \langle M, c_1 \rangle \rightarrow \langle M, c_1' \rangle}{\bar{p} \vdash \langle M, c_1; c_2 \rangle \rightarrow \langle M, c_1'; c_2 \rangle} \text{EC-Seq-Struct}$$

# Command Evaluation

$$\frac{\bar{p}; M \vdash^* e \rightarrow \text{false};}{\bar{p} \vdash \langle M, \text{while } e \text{ do } c \rangle \rightarrow \langle M, \text{skip} \rangle} \text{EC-While-False}$$

$$\frac{\bar{p}; M \vdash^* e \rightarrow \text{true};}{\bar{p} \vdash \langle M, \text{while } e \text{ do } c \rangle \rightarrow \langle M, c; \text{while } e \text{ do } c \rangle} \text{EC-While-True}$$

# Command Evaluation

$$\frac{\bar{p}; M \vdash^* e \rightarrow \text{inl } v}{\bar{p} \vdash \langle M, \text{case } e \text{ of } a_1 \Rightarrow c_1 \ a_2 \Rightarrow c_2 \rangle \rightarrow \langle M, M, [v/a_1]c_1 \rangle} \text{EC-Case-Inl}$$

$$\frac{\bar{p}; M \vdash^* e \rightarrow \text{inr } v}{\bar{p} \vdash \langle M, \text{case } e \text{ of } a_1 \Rightarrow c_1 \ a_2 \Rightarrow c_2 \rangle \rightarrow \langle M, M, [v/a_2]c_2 \rangle} \text{EC-Case-Inr}$$

# Static Semantics

- ▶ Defines portion of grammar over which dynamic semantics is always defined.
- ▶ Prevents high-to-low information flows, except where permitted by pack and unpack.

# Expression Typing

$$\frac{}{\Theta; \Gamma \vdash i : \text{int}\{l\}} \text{TE-Int}$$

$$\frac{}{\Theta; \Gamma \vdash \langle v \rangle_e : \text{pkg}\{l\}} \text{TE-Package}$$

$$\frac{\Theta; \Gamma \vdash i_1 : \text{int}\{l\} \quad \Theta; \Gamma \vdash i_2 : \text{int}\{l\}}{\Theta; \Gamma \vdash i_1 + i_2 : \text{int}\{l\}} \text{TE-Plus}$$

# Expression Typing

$$\frac{\Theta; \Gamma \vdash e : \tau_1 \{l\}}{\Theta; \Gamma \vdash \text{inl } e : (\tau_1 + \tau_2) \{l\}} \text{TE-Inl}$$

$$\frac{\Theta; \Gamma \vdash e : \tau_2 \{l\}}{\Theta; \Gamma \vdash \text{inr } e : (\tau_1 + \tau_2) \{l\}} \text{TE-Inr}$$

# Expression Typing

$$\frac{\Theta; \Gamma \vdash e : \tau\{l_e\} \quad I(l_e) = I(l)}{\Theta; \Gamma \vdash \text{pack } e \text{ at } l : (\text{pkg} + \text{error})\{l\}} \text{TE-Pack}$$

$$\frac{\Theta; \Gamma \vdash e : \text{pkg}\{l_e\} \quad C(l_e) = C(l)}{\Theta; \Gamma \vdash \text{unpack } e \text{ as } \tau\{l\} : (\tau + \text{error})\{l\}} \text{TE-Unpack}$$

# Expression Typing

$$\frac{\Gamma(a) = \tau\{l\}}{\Theta; \Gamma \vdash a : \tau\{l\}} \text{TE-Var}$$

$$\frac{\Theta(x) = \tau\{l\}}{\Theta; \Gamma \vdash x : \tau\{l\}} \text{TE-Loc}$$

$$\frac{\Theta; \Gamma \vdash e : \tau\{l'\} \quad l' \leq l}{\Theta; \Gamma \vdash e : \tau\{l\}} \text{TE-Sub}$$

# Command Typing

$$\frac{}{pc; \Theta; \Gamma \vdash \text{skip}} \text{TC-Skip}$$

$$\frac{\Theta; \Gamma \vdash e : \tau\{l_e\} \quad \Theta(x) = \tau(l) \quad l_e \sqcup pc \leq l}{pc; \Theta; \Gamma \vdash x := e} \text{TC-Assign}$$

$$\frac{pc; \Theta; \Gamma \vdash c_1 \quad pc; \Theta; \Gamma \vdash c_2}{pc; \Theta; \Gamma \vdash c_1; c_2} \text{TC-Seq}$$

# Command Typing

$$\frac{\Theta; \Gamma \vdash e : \text{bool}\{l\} \quad pc \sqcup l; \Theta; \Gamma \vdash c}{pc; \Theta; \Gamma \vdash \text{while } e \text{ do } c} \text{TC-While}$$

$$\frac{\Theta; \Gamma \vdash e : (\tau_1 + \tau_2)\{l\} \quad pc \sqcup l; \Theta; \Gamma[a_1 \mapsto \tau_1\{l\}] \vdash c_1 \quad pc \sqcup l; \Theta; \Gamma[a_2 \mapsto \tau_2\{l\}] \vdash c_2}{pc; \Theta; \Gamma \vdash \text{case } e \text{ of } a_1 \Rightarrow c_1 \ a_2 \Rightarrow c_2} \text{TC-Case}$$

# Command Typing

$$\frac{pc'; \Theta; \Gamma \vdash c \quad pc \leq pc'}{pc; \Theta; \Gamma \vdash c} \text{TC-Subs}$$

# Equivalent Values

To state noninterference, we first need a notion of equivalent values.

$$\frac{}{i \cong_{\ell} i} \text{VE-Int}$$

$$\frac{v_1 \cong_{\ell} v_2}{\text{inl } v_1 \cong_{\ell} \text{inl } v_2} \text{VE-Inl}$$

$$\frac{v_1 \cong_{\ell} v_2}{\text{inr } v_1 \cong_{\ell} \text{inr } v_2} \text{VE-Inr}$$

# Equivalent Values

$$\frac{v_1 \cong_l v_2}{\langle v_1 \rangle_{l_1} \cong_l \langle v_2 \rangle_{l_1}} \text{VE-Pack-In}$$

$$\frac{l_1 \not\leq l}{\langle v_1 \rangle_{l_1} \cong_l \langle v_2 \rangle_{l_1}} \text{VE-Pack-Lab}$$

# Noninterference

**Definition 1** ( $\Theta \vdash M_1 \cong_\ell M_2$ ). In context  $\Theta$ , memories  $M_1$  and  $M_2$  are equivalent at level  $\ell$ , written  $(\Theta \vdash M_1 \cong_\ell M_2)$ , when  $\text{dom}(M_1) = \text{dom}(M_2)$  and

$$\forall x \in \text{dom}(M_1). \exists \tau, \ell'. \Theta(x) = \tau\{\ell'\} \wedge (M_1(x) \cong_\ell M_2(x) \vee \ell' \not\leq \ell) .$$

- ▶ Noninterference treats memories as identical when corresponding low security values are identical.

# Noninterference

**Definition 2** ( $\Theta \vdash M \text{ OK}$ ). A memory,  $M$ , is well typed in location context  $\Theta$ , when  $\forall x \in \text{dom}(\Theta). \Theta(x) = \tau\{ell\} \wedge \vdash M(x) : \tau$ . This property is written  $\Theta \vdash M \text{ OK}$ .

- ▶ Define a notion of well typed memories. Executing well-typed programs under well-typed memories is type-safe.

# Noninterference

A trace is a set of principals and a sequence of configurations,  $(\bar{p}, \langle M_1, c_1 \rangle, \langle M_2, c_2 \rangle, \dots, \langle M_n, c_n \rangle)$ , where for each  $i \in \{1, n-1\}$ ,  $\bar{p} \vdash \langle M_i, c_i \rangle \rightarrow \langle M_{i+1}, c_{i+1} \rangle$ .

**Lemma 2** (Trace determinacy). *There is at most one shortest trace of form  $(\bar{p}, \langle M_1, c_1 \rangle, \dots, \langle M_n, c_n \rangle)$ .*

*If such a trace exists, we will write it as  $\bar{p} \vdash \langle M_1, c_1 \rangle \rightarrow^* \langle M_n, c_n \rangle$ .*

# Noninterference

**Theorem 1** (Expression Non-interference). *If*

- ▶  $\Theta \vdash M_1 \text{ OK}$ ,  $\Theta \vdash M_2 \text{ OK}$  and  $\Theta \vdash M_1 \cong M_2$
- ▶  $p\text{c}; \Theta; \cdot \vdash c_1$  and  $c_1 \cong_\ell c_2$
- ▶  $\bar{p} \vdash \langle M_1, c_1 \rangle \rightarrow^* \langle M'_1, \text{skip} \rangle$  and  $\bar{p} \vdash \langle M_2, c_2 \rangle \rightarrow^* \langle M'_2, \text{skip} \rangle$

then  $\Theta \vdash M'_1 \cong_\ell M'_2$ .

# Cryptographic Semantics

- ▶ Noninterference ensures security for programs run in trusted environments.
- ▶ What about hostile environments?
  - ▶ Interpret labels using cryptography to ensure information flow guarantees hold in open systems.
- ▶ A formal syntax for messages, and a Dolev-Yao deduction system to reason about them.
- ▶ A compilation from  $\text{SImp}$  into messages.

# Message Syntax

Principals	$p, q, r$	::=	Alice   Bob   ...
Key Id	$\kappa, W, R$		<i>abstract</i>
Private Keys	$K^-$	::=	$K_{\kappa}^-$
Public Keys	$K^+$	::=	$K_{\kappa}^-$
Strings	$str$	::=	"a"   "b"   ...
Messages	$m, n$	::=	$str$   $K$   $p$   $(m, m')$
		::=	$enc(K, m)$   $sign(K, m)$

# Cryptographic Deduction System

$$\frac{(knows\ m) \in \sigma}{\sigma \vdash_d m} \text{D-Taut}$$

$$\frac{\sigma \vdash_u K \quad \sigma \vdash_d m}{\sigma \vdash_d enc(K, m)} \text{D-Encrypt}$$

$$\frac{\sigma \vdash_d enc(K^+, m) \quad \sigma \vdash_u K^-}{\sigma \vdash_d m} \text{D-Decrypt}$$

# Cryptographic Deduction System

$$\frac{\sigma \vdash_u K \quad \sigma \vdash_d m}{\sigma \vdash_d \text{sign}(K, m)} \text{D-Sign}$$

$$\frac{\sigma \vdash_d \text{sign}(K, m)}{\sigma \vdash_d m} \text{D-Sign-Id}$$

$$\frac{\sigma \vdash_b m}{\sigma \vdash_d m} \text{D-Lift}$$

$$\frac{\sigma \vdash_d \text{enc}(K, m)}{\sigma \vdash_d K} \text{D-Enc-Id}$$

# Cryptographic Deduction System

$$\frac{\sigma \vdash_d (m_1, m_2)}{\sigma \vdash_d m_1} \text{D-PairL}$$

$$\frac{\sigma \vdash_d (m_1, m_2)}{\sigma \vdash_d m_2} \text{D-PairR}$$

$$\frac{\sigma \vdash_d m_1 \quad \sigma \vdash_d m_2}{\sigma \vdash_d (m_1, m_2)} \text{D-Pair}$$

# Cryptographic Deduction System

$$\frac{(actswith\ m) \in \sigma}{\sigma \vdash_u m} \text{U-Taut}$$

$$\frac{\sigma \vdash_d m}{\sigma \vdash_u m} \text{U-Lift}$$

# Cryptographic Deduction System

$$\frac{(\textit{believes } m) \in \sigma}{\sigma \vdash_b m} \text{B-Taut}$$

$$\frac{\sigma \vdash_d \textit{sign}(K^-, m) \quad \sigma \vdash_b K^+}{\sigma \vdash_b m} \text{B-Sign-Verify}$$

# Compiling Policies

Translating  $\langle v \rangle_\ell$  takes three steps.

- ▶ Translate each policy in  $\ell$  into a *seal* which can be used to ensure the confidentiality and integrity of a sealed message.
- ▶ Compose all the seals into an *envelope* which can be read and written only in accordance with  $\ell$ 's meaning.
- ▶ Translate  $v$  and write its translation into the envelope.
  - ▶ Envelopes serve as a way to secure channels.

# Seals

The seal  $P[[o : \bar{r}!\bar{w}]]$  is intended to ensure an envelope sealed with  $P[[o : \bar{r}!\bar{w}]]$  should only be written to with a private key belonging to  $o$  or a member of  $\bar{w}$  and read by principals with a private key from  $o, \bar{r}$ .

# Envelopes

- ▶ The envelope  $L[[\ell]]_{\bar{\kappa}}$  translates the policies in  $\ell$  into a list of seals.
- ▶ Each of an envelope's seal's is associated with two private keys, one reading and one writing.
- ▶ Writing to an envelope requires writing private keys of all seals in envelope.

# Value Translation

$$\begin{aligned}V[[i]]_{\bar{\kappa}} &= \text{"i"} \\V[[\text{inl } v]]_{\bar{\kappa}} &= (\text{"inl"}, V[[v]]_{\bar{\kappa}}) \\V[[\text{inr } v]]_{\bar{\kappa}} &= (\text{"inr"}, V[[v]]_{\bar{\kappa}}) \\V[[\langle v \rangle \ell]]_{\overline{\kappa_1, \kappa_2}} &= (\text{doPack } \bar{\kappa}_2 \ V[[v]]_{\bar{\kappa}_1}, L[[\ell]]_{\bar{\kappa}_2})\end{aligned}$$

# Memory Translation

Entire memories are translated to cryptographic states by packing each location's contents,  $M[[M]]_{\kappa}^{\Theta}$

# Cryptographic Analysis

- ▶ Want to connect the cryptographic system to DLM.
- ▶  $\sigma \succ \bar{p}$  holds when  $\forall p \in \bar{p}. \sigma \vdash_u K_p^-$ .
- ▶  $\sigma \vdash m \cong m'$  means that in the state  $\sigma$  messages  $m$  and  $m'$  are indistinguishable.

# Contextual Message Equivalence

$$\frac{}{\sigma \vdash m \cong m} \text{ME-ID}$$

$$\frac{\sigma, \textit{knows } m_2, \textit{knows } m'_2 \vdash m_1 \cong m'_1 \quad \sigma, \textit{knows } m_1, \textit{knows } m'_1 \vdash m_2 \cong m'_2}{\sigma \vdash (m_1, m_2) \cong (m'_1, m'_2)} \text{ME - Pair}$$

# Contextual Message Equivalence

$$\frac{\sigma \vdash m_1 \cong m_2}{\sigma \vdash \text{enc}(K, m_1) \cong \text{enc}(K, m_2)} \text{ME-Enc-Struct}$$

$$\frac{\sigma \not\vdash_u K_1^- \quad \sigma \not\vdash_u K_2^-}{\sigma \vdash \text{enc}(K_1^+, m_1) \cong \text{enc}(K_2^+, m_2)} \text{ME-Enc-Secret}$$

# Contextual Message Equivalence

$$\frac{\sigma \vdash m_1 \cong m_2}{\sigma \vdash \text{sign}(K, m_1) \cong \text{sign}(K, m_2)} \text{ME-Sign-Struct}$$

$$\frac{\sigma \not\vdash_b K_1^+ \quad \sigma \not\vdash_b K_2^+}{\sigma \vdash \text{sign}(K_1^-, m_1) \cong \text{sign}(K_2^-, m_2)} \text{ME-Enc-Secret}$$

# Cryptographic Analysis

Extend notion of  $\ell$ -equivalence to messages.

- ▶ **Definition 3** ( $\sigma$  reads  $\ell$ ) If  $\exists \bar{p}. (\bar{p} \text{ reads } \ell \wedge \forall p \in \bar{p}, \not\vdash_u K_p^-)$  then  $\sigma$  reads  $\ell$ .

# Cryptographic Analysis

Extend notion of  $\ell$ -equivalence to messages.

- ▶ **Definition 3** ( $\sigma$  reads  $\ell$ ) If  $\exists \bar{p}.(\bar{p} \text{ reads } \ell \wedge \forall p \in \bar{p}, \not\vdash_u K_p^-)$  then  $\sigma$  reads  $\ell$ .
- ▶ **Definition 4** ( $\sigma$  distrusts  $\ell$ ) If  $\exists \bar{p}.(\bar{p} \text{ writes } \ell \wedge \forall p \in \bar{p}, \not\vdash_b K_p^-)$  then  $\sigma$  distrusts  $\ell$ .

# Cryptographic Analysis

Extend notion of  $\ell$ -equivalence to messages.

- ▶ **Definition 3** ( $\sigma$  reads  $\ell$ ) If  $\exists \bar{p}.(\bar{p} \text{ reads } \ell \wedge \forall p \in \bar{p}, \forall_u K_p^-)$  then  $\sigma$  reads  $\ell$ .
- ▶ **Definition 4** ( $\sigma$  distrusts  $\ell$ ) If  $\exists \bar{p}.(\bar{p} \text{ writes } \ell \wedge \forall p \in \bar{p}, \forall_b K_p^-)$  then  $\sigma$  distrusts  $\ell$ .
- ▶ **Definition 5** ( $\sigma \leq \ell$ ). If  $\neg \sigma$  reads  $\ell$  and  $\sigma$  distrusts  $\ell$  then  $\sigma \leq \ell$

# Cryptographic Analysis

Extend notion of  $\ell$ -equivalence to messages.

- ▶ **Definition 3** ( $\sigma$  reads  $\ell$ ) If  $\exists \bar{p}. (\bar{p} \text{ reads } \ell \wedge \forall p \in \bar{p}, \not\vdash_u K_p^-)$  then  $\sigma$  reads  $\ell$ .
- ▶ **Definition 4** ( $\sigma$  distrusts  $\ell$ ) If  $\exists \bar{p}. (\bar{p} \text{ writes } \ell \wedge \forall p \in \bar{p}, \not\vdash_b K_p^-)$  then  $\sigma$  distrusts  $\ell$ .
- ▶ **Definition 5** ( $\sigma \leq \ell$ ). If  $\neg \sigma$  reads  $\ell$  and  $\sigma$  distrusts  $\ell$  then  $\sigma \leq \ell$
- ▶ **Definition 6** ( $m \cong_\ell m'$ ).  $m \cong_\ell m'$  if and only if  $\forall \sigma. \sigma \leq \ell \implies \sigma, \text{ knows } m, \text{ knows } m' \vdash m \cong m'$ .

# Cryptographic Analysis

Lift  $\ell$ -equivalence to cryptographic states.

**Definition 7** ( $\sigma \cong_e \parallel \sigma'$ ). *If knows  $m \in \sigma$  implies  $\exists m'. m' \in \sigma'$  and  $m \cong_\ell m'$ , then  $\sigma' \models_\ell \sigma$ . If  $\sigma' \models_\ell \sigma$  and  $\sigma' \models \sigma$ , then  $\sigma \cong_e \parallel \sigma'$ .*

# Cryptographic Analysis

Relate SImp equivalence to cryptographic equivalence.

**Lemma 3** (Adequacy of Value Translation). *If  $v_1 \cong_\ell v_2$  and  $\bar{\kappa}$  is fresh then  $v[[v_1]]_{\bar{\kappa}} \cong_\ell v[[v_2]]_{\bar{\kappa}}$ .*

**Corollary 1** (Adequacy of Memory Translation). *If  $\Theta \vdash M_1 \cong_\ell M_2$  and  $\bar{\kappa}$  is fresh then  $M[[M_1]]_{\bar{\kappa}}^\Theta \cong_\ell M[[M_2]]_{\bar{\kappa}}^\Theta$ .*

# Cryptographic State Transitions

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

$$\frac{\sigma \vdash_d m}{\vdash \sigma \rightarrow \sigma, \text{knows } m} \text{CS-Derive}$$

$$\frac{\sigma' \subseteq \sigma}{\vdash \sigma \rightarrow \sigma'} \text{CS-Forget}$$

$$\frac{\sigma \vdash_d "i_1" \quad \sigma \vdash_d "i_2" \quad i_3 = i_1 + i_2}{\vdash \sigma \rightarrow \sigma, "i_3"} \text{CS-Compute}$$

# Cryptographic Analysis

There exists *some* cryptographic realization of the memory translation of the program.

**Theorem 3** (Feasibility). *If  $\Theta \vdash M$  OK,  $pc; \Theta; \Gamma \vdash c$ ,  $\bar{p}$  writes  $p$ , and  $\bar{p} \vdash \langle M, c \rangle \rightarrow \langle M', c' \rangle$  then*

*$\exists \bar{\kappa}_3, \bar{\kappa}_4 \vdash M[[M]]_{\bar{\kappa}_1}^{\Theta} \cup \text{state}(\bar{\kappa}_2, \bar{p}, c) \rightarrow^* M[[M']]_{\bar{\kappa}_3}^{\Theta} \cup \text{state}(\bar{\kappa}_4, \bar{p}, c)$*

.

Fin

Questions?