STL^3

Toward Security via Free Theorems*
in a **S**ession-**T**yped **L**inear **L**anguage with **L**ocations

*Work in Progress!

Andrew Wagner Amal Ahmed

February 1, 2024

Northeastern University

Secure Coin Flipping

COIN FLIPPING BY TELEPHONE
A PROTOCOL FOR SOLVING IMPOSSIBLE PROBLEMS

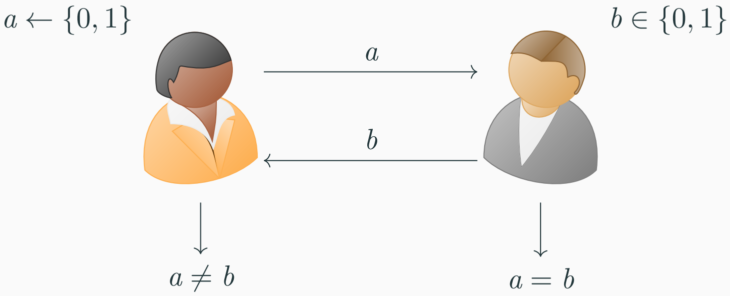
Manuel Blum*

Department of Electrical Engineering and Computer Sciences
Computer Science Division
University of California at Berkeley
November 10, 1981

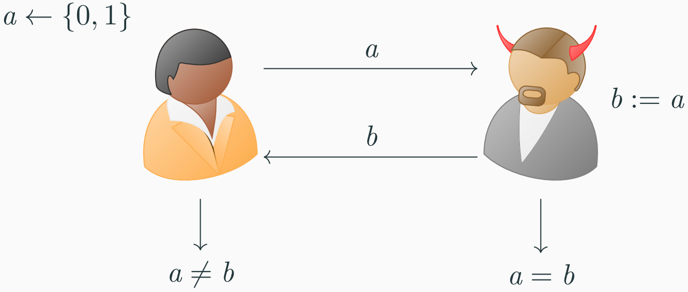
Abstract

Alice and Bob want to flip a coin by telephone. (They are very good friends , live in different cities, want to decide who will travel to see whom.) Bob would not like to tell Alice HEADS and hear Alice (at the other end of the line) say "Here goes... I'm flipping the coin.... You lost!"

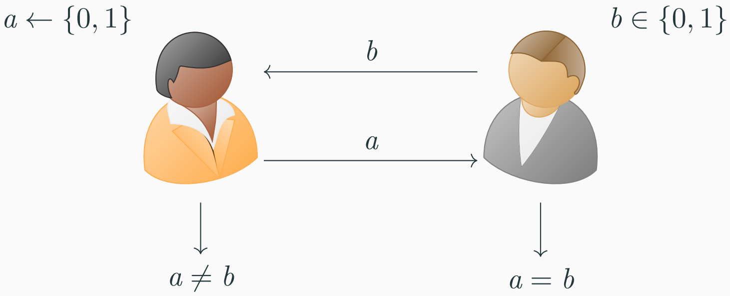
Insecure Coin Flipping



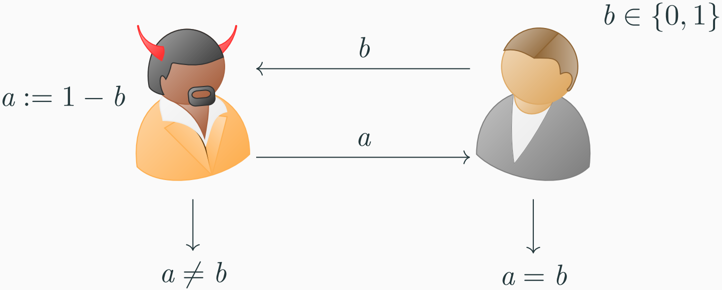
Insecure Coin Flipping



Insecure Coin Flipping



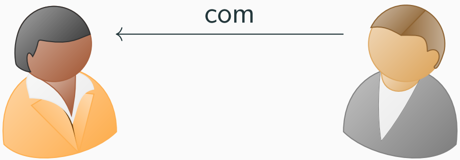
Insecure Coin Flipping



Secure Coin Flipping

$$b \in \{0, 1\}$$

$$(\mathsf{com}, \mathsf{dec}) \leftarrow \mathsf{Commit}(b)$$

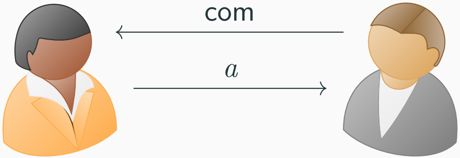


Secure Coin Flipping

$b \in \{0, 1\}$

$(\mathsf{com}, \mathsf{dec}) \leftarrow \mathsf{Commit}(b)$

$a \leftarrow \{0, 1\}$

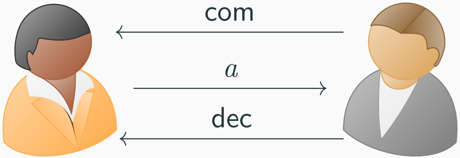


Secure Coin Flipping

$b \in \{0,1\}$

$(\mathsf{com}, \mathsf{dec}) \leftarrow \mathsf{Commit}(b)$

$a \leftarrow \{0,1\}$

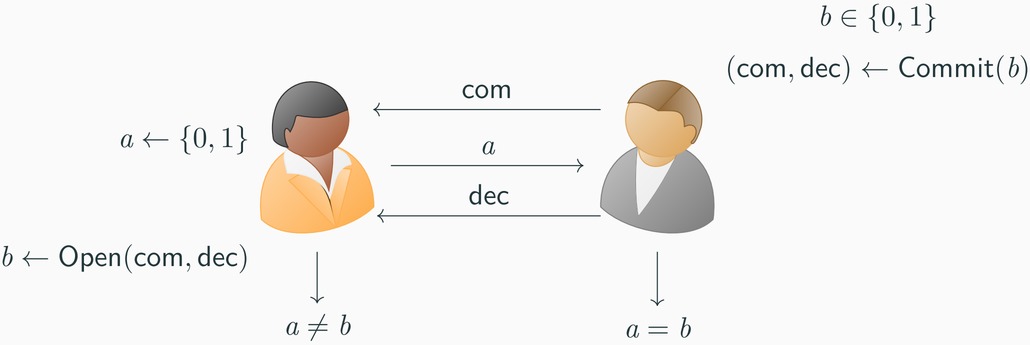


$b \leftarrow \mathsf{Open}(\mathsf{com}, \mathsf{dec})$

$a \neq b$

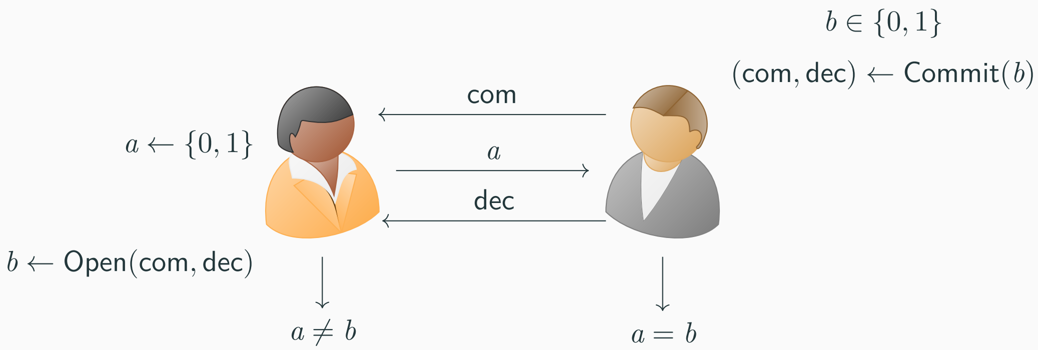
$a = b$

Secure Coin Flipping



Can we express this protocol with session types? Definitely!

Secure Coin Flipping



Can we express the security of this protocol with session types?

Key Ideas

- $\star$ Types can reference *particular* processes by name: $@_x\, T$

Key Ideas

- Types can reference *particular* processes by name: $@_x\, T$
- ... and can *quantify* over names, establishing name parametricity: $\forall\, x.\, T, \exists\, x.\, T$

Key Ideas

- Types can reference *particular* processes by name: $@_x\, T$
- … and can *quantify* over names, establishing name parametricity: $\forall\, x. T, \exists\, x. T$
- Security[1] of some protocols can be expressed as free theorems

[1] information flow

Background: Intuitionistic Binary Sessions[2]

$\boxed{\Delta \vdash \mathcal{P} :: (x : T)}$ Δ a lin. ctx. of channels; $\mathcal{P}$ a process; x a name; T a type

[2] Caires and Pfenning 2010.

Background: Intuitionistic Binary Sessions[2]

$\boxed{\Delta \vdash \mathcal{P} :: (x : T)}$ Δ a lin. ctx. of channels; $\mathcal{P}$ a process; x a name; T a type

$\multimap R$

$$\frac{\Delta, y : S \vdash \mathcal{P} :: (x : T)}{\Delta \vdash y \leftarrow x.\textbf{recv}; \mathcal{P} :: (x : S \multimap T)}$$

[2] Caires and Pfenning 2010.

Background: Intuitionistic Binary Sessions[2]

$\boxed{\Delta \vdash \mathcal{P} :: (x : T)}$ Δ a lin. ctx. of channels; $\mathcal{P}$ a process; x a name; T a type

$\multimap R$

$$\frac{\Delta, y : S \vdash \mathcal{P} :: (x : T)}{\Delta \vdash y \leftarrow x.\textbf{recv}; \mathcal{P} :: (x : S \multimap T)}$$

$\multimap L^*$

$$\frac{\Delta, y : S_2 \vdash \mathcal{P} :: (z : T)}{\Delta, x : S_1, y : S_1 \multimap S_2 \vdash y.\textbf{send}(x); \mathcal{P} :: (z : T)}$$

[2] Caires and Pfenning 2010.

Background: Intuitionistic Binary Sessions[2]

$\boxed{\Delta \vdash \mathcal{P} :: (x : T)}$ Δ a lin. ctx. of channels; $\mathcal{P}$ a process; x a name; T a type

$\multimap R$

$$\frac{\Delta, y : S \vdash \mathcal{P} :: (x : T)}{\Delta \vdash y \leftarrow x.\textbf{recv}; \mathcal{P} :: (x : S \multimap T)}$$

$\multimap L^*$

$$\frac{\Delta, y : S_2 \vdash \mathcal{P} :: (z : T)}{\Delta, x : S_1, y : S_1 \multimap S_2 \vdash y.\textbf{send}(x); \mathcal{P} :: (z : T)}$$

$\boxed{\delta \to \delta'}$ δ a list of processes

[2] Caires and Pfenning 2010.

Background: Intuitionistic Binary Sessions[2]

$\boxed{\Delta \vdash \mathcal{P} :: (x : T)}$ Δ a lin. ctx. of channels; $\mathcal{P}$ a process; x a name; T a type

$\multimap R$
$$\frac{\Delta, y : S \vdash \mathcal{P} :: (x : T)}{\Delta \vdash y \leftarrow x.\textbf{recv}; \mathcal{P} :: (x : S \multimap T)}$$

$\multimap L^*$
$$\frac{\Delta, y : S_2 \vdash \mathcal{P} :: (z : T)}{\Delta, x : S_1, y : S_1 \multimap S_2 \vdash y.\textbf{send}(x); \mathcal{P} :: (z : T)}$$

$\boxed{\delta \to \delta'}$ δ a list of processes

$\multimap / \otimes$
$$\langle \hat{x}.\textbf{send}(\hat{y}); \mathcal{P} \rangle \quad \langle y \leftarrow \hat{x}.\textbf{recv}; \mathcal{Q} \rangle \to \langle \mathcal{P} \rangle \quad \langle \mathcal{Q}[\hat{y}/y] \rangle$$

[2] Caires and Pfenning 2010.

Background[3]: L^3

$$\boxed{\mathsf{ref}\ T \cong \exists\,\ell.\ !\mathsf{ptr}\,\ell \otimes \mathsf{cap}\,\ell\ T}$$

Decompose a traditional *reference* into an unrestricted *pointer* and a linear *capability*, tied together by a type-level name, ℓ.

[3] Ahmed, Fluet, and Morrisett 2007.

STL^3 Extensions: Exchanging Names

$\boxed{\Gamma; \Delta \vdash \mathcal{P} :: (x : T)}$ Γ an unrestricted context of names, $\Gamma \supseteq$ freenames(Δ, T)

STL^3 Extensions: Exchanging Names

$\boxed{\Gamma; \Delta \vdash \mathcal{P} :: (x : T)}$ Γ an unrestricted context of names, $\Gamma \supseteq \text{freenames}(\Delta, T)$

$$\multimap R \quad \frac{\Gamma, y; \Delta, y : S \vdash \mathcal{P} :: (x : T)}{\Gamma; \Delta \vdash y \leftarrow x.\textbf{recv}; \mathcal{P} :: (x : S \multimap T)}$$

STL^3 Extensions: Exchanging Names

$\boxed{\Gamma; \Delta \vdash \mathcal{P} :: (x : T)}$ Γ an unrestricted context of names, $\Gamma \supseteq$ freenames(Δ, T)

$$\multimap R \quad \frac{\Gamma, y; \Delta, y : S \vdash \mathcal{P} :: (x : T)}{\Gamma; \Delta \vdash y \leftarrow x.\textbf{recv}; \mathcal{P} :: (x : S \multimap T)}$$

$$\text{cut} \quad \frac{\Gamma; \Delta_1 \vdash \mathcal{P} :: (x : S) \qquad \Gamma, x; \Delta_2, x : S \vdash \mathcal{Q} :: (y : T)}{\Gamma; \Delta_1, \Delta_2 \vdash x \leftarrow \langle \mathcal{P} \rangle ;\ \mathcal{Q} :: (y : T)}$$

STL^3 Extensions: Exchanging Names

$\boxed{\Gamma; \Delta \vdash \mathcal{P} :: (x : T)}$ Γ an unrestricted context of names, $\Gamma \supseteq$ freenames(Δ, T)

$$\multimap R \quad \frac{\Gamma, y; \Delta, y : S \vdash \mathcal{P} :: (x : T)}{\Gamma; \Delta \vdash y \leftarrow x.\textbf{recv}; \mathcal{P} :: (x : S \multimap T)}$$

$$\text{cut} \quad \frac{\Gamma; \Delta_1 \vdash \mathcal{P} :: (x : S) \qquad \Gamma, x; \Delta_2, x : S \vdash \mathcal{Q} :: (y : T)}{\Gamma; \Delta_1, \Delta_2 \vdash x \leftarrow \langle \mathcal{P} \rangle ;\ \mathcal{Q} :: (y : T)}$$

$$\forall R \quad \frac{\Gamma, y; \Delta \mathcal{P} :: (x : T)}{\Gamma; \Delta \vdash [y] \leftarrow x.\textbf{recv}; \mathcal{P} :: (x : \forall\, y. T)}$$

STL^3 Extensions: Exchanging Names

$\boxed{\Gamma; \Delta \vdash \mathcal{P} :: (x : T)}$ Γ an unrestricted context of names, $\Gamma \supseteq$ freenames(Δ, T)

$$\multimap R \quad \frac{\Gamma, y; \Delta, y : S \vdash \mathcal{P} :: (x : T)}{\Gamma; \Delta \vdash y \leftarrow x.\textbf{recv}; \mathcal{P} :: (x : S \multimap T)}$$

$$\text{cut} \quad \frac{\Gamma; \Delta_1 \vdash \mathcal{P} :: (x : S) \qquad \Gamma, x; \Delta_2, x : S \vdash \mathcal{Q} :: (y : T)}{\Gamma; \Delta_1, \Delta_2 \vdash x \leftarrow \langle \mathcal{P} \rangle ;\ \mathcal{Q} :: (y : T)}$$

$$\forall R \quad \frac{\Gamma, y; \Delta \mathcal{P} :: (x : T)}{\Gamma; \Delta \vdash [y] \leftarrow x.\textbf{recv}; \mathcal{P} :: (x : \forall\, y. T)}$$

$$\forall L \quad \frac{\Gamma; \Delta, y : S[x/x'] \vdash \mathcal{P} :: (z : T) \qquad \Gamma \ni x}{\Gamma; \Delta, y : \forall\, x'.S \vdash y.\textbf{send}[x]; \mathcal{P} :: (z : T)}$$

STL[3] Extensions: Jumping[4]

$\boxed{\Gamma; \Delta \vdash \mathcal{P} :: (x : T)}$ Γ an unrestricted context of names, $\Gamma \supseteq \text{freenames}(\Delta, T)$

$@_x R$

$\Gamma; x : T \vdash y.\textbf{send}(@x) :: (y : @_x T)$

[4] Braüner and Paiva 2006.

STL[3] Extensions: Jumping[4]

$\boxed{\Gamma; \Delta \vdash \mathcal{P} :: (x : T)}$ Γ an unrestricted context of names, $\Gamma \supseteq \mathsf{freenames}(\Delta, T)$

$@_x R$

$$\Gamma; x : T \vdash y.\mathbf{send}(@x) :: (y : @_x\, T)$$

$@_x L$

$$\frac{\Gamma; x : S \vdash \mathcal{P} :: (z : T)}{\Gamma; y : @_x\, S \vdash @x \leftarrow y.\mathbf{recv}; \mathcal{P} :: (z : T)}$$

[4] Braüner and Paiva 2006.

STL[3] Extensions: Jumping[4]

$\boxed{\Gamma; \Delta \vdash \mathcal{P} :: (x : T)}$ Γ an unrestricted context of names, $\Gamma \supseteq$ freenames(Δ, T)

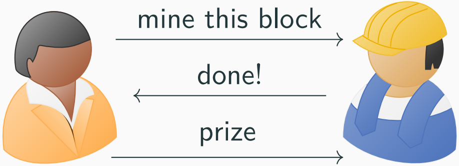
$$\dfrac{}{\Gamma; x : T \vdash y.\mathbf{send}(@x) :: (y : @_x\, T)}\ @_x R \qquad \dfrac{\Gamma; x : S \vdash \mathcal{P} :: (z : T)}{\Gamma; y : @_x\, S \vdash @x \leftarrow y.\mathbf{recv}; \mathcal{P} :: (z : T)}\ @_x L$$

$\boxed{\delta \to \delta'}$

$$\dfrac{}{\langle \hat{y}.\mathbf{send}(@\hat{x})\rangle \ \ \langle @\hat{x} \leftarrow \hat{y}.\mathbf{recv}; \mathcal{P}\rangle \to \langle \mathcal{P}\rangle}\ @$$

[4] Braüner and Paiva 2006.

Example: Proof of Work

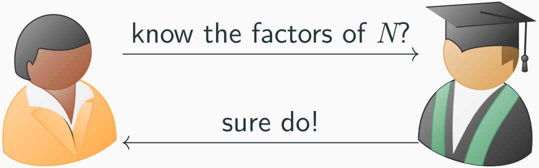


Example: Proof of Work

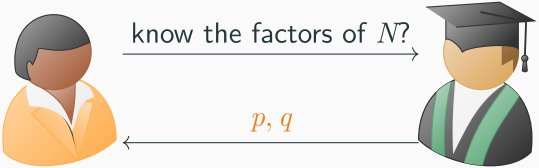
Example: Proof of Work

$$\boxed{\mathsf{PoW} \triangleq \forall\, i.\ \underbrace{@_i\, T}_{\text{task}} \multimap \underbrace{@_i\, \mathbb{1}}_{\text{cert}} \otimes \underbrace{\mathbb{N}}_{\text{prize}} \multimap \mathbb{1}}$$

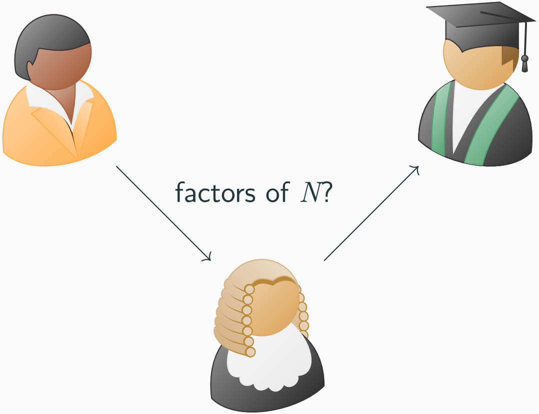
Example: Zero-Knowledge Proof



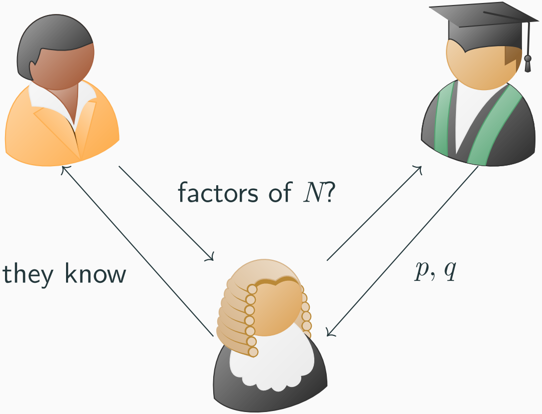
Example: Zero-Knowledge Proof



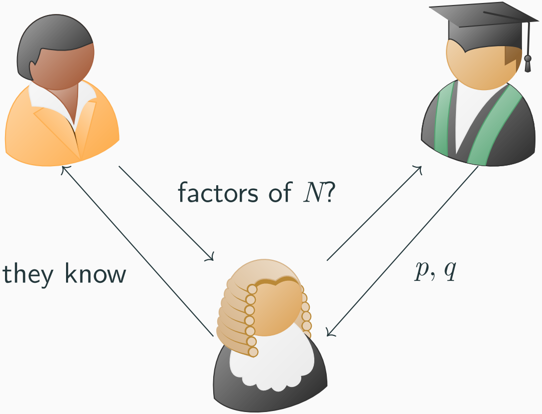
Example: Zero-Knowledge Proof



Example: Zero-Knowledge Proof



Example: Zero-Knowledge Proof



$$\boxed{\mathsf{ZKP} \triangleq \forall\, i.\ \underbrace{@_i\,(\mathbb{N}^2 \multimap \mathbb{B})}_{\text{test}} \multimap \underbrace{@_i\,\mathbb{B}}_{\text{cert}}}$$

Security Property: Authenticity

$$\boxed{\mathsf{Adv} \triangleq \forall\, e.\ \underbrace{@_e \mathbb{B}}_{\text{sign}} \multimap \underbrace{@_e \mathbb{B}}_{\text{forge?}}}$$

“Can an adversary forge a named process?”

Security Property: Authenticity

$$\boxed{\mathsf{Adv} \triangleq \forall\, e.\ \underbrace{@_e\,\mathbb{B}}_{\text{sign}} \multimap \underbrace{@_e\,\mathbb{B}}_{\text{forge?}}}$$

"Can an adversary forge a named process?"

Conjecture (Authenticity)

For all $\mathcal{E} : \mathbb{B}$, $\mathcal{A} : \mathsf{Adv}$,

$$\mathcal{E} \approx e \leftarrow \mathcal{E}; \mathsf{adv} \leftarrow \mathcal{A}; \mathsf{exp}(\mathsf{adv}, e)$$

Security Property: Authenticity

$$\boxed{\mathsf{Adv} \triangleq \forall\, e.\ \underbrace{@_e\mathbb{B}}_{\text{sign}} \multimap \underbrace{@_e\mathbb{B}}_{\text{forge?}}}$$

"Can an adversary forge a named process?"

$$
\begin{array}{l}
\textbf{proc}\ \mathsf{exp}(\mathsf{adv} : \mathsf{Adv}, e : \mathbb{B}) : \mathbb{B} \triangleq \\
\quad \mathsf{adv}.\textbf{send}[e]; \\
\quad \mathsf{sig} \leftarrow \langle \mathsf{sig.send}(@e) \rangle\ ; \\
\quad \mathsf{adv}.\textbf{send}(\mathsf{sig}); \\
\quad @e \leftarrow \mathsf{adv}.\textbf{recv}; \\
\quad \mathsf{exp}.\textbf{fwd}(e)
\end{array}
$$

Conjecture (Authenticity)

For all $\mathcal{E} : \mathbb{B}$, $\mathcal{A} : \mathsf{Adv}$,

$$\mathcal{E} \approx e \leftarrow \mathcal{E}; \mathsf{adv} \leftarrow \mathcal{A}; \mathsf{exp}(\mathsf{adv}, e)$$

Security Property: Hiding

$$\boxed{\mathsf{Adv} \triangleq \forall\, e.\ \underbrace{@_e\, \mathbb{B}}_{\text{enc}} \multimap \underbrace{@_e\, \mathbb{B}}_{\text{ack}} \otimes \underbrace{\mathbb{B}}_{\text{guess?}}}$$

"Can an adversary peek into a named process?"

Security Property: Hiding

$$\boxed{\mathsf{Adv} \triangleq \forall\, e.\ \underbrace{@_e\,\mathbb{B}}_{\mathsf{enc}} \multimap \underbrace{@_e\,\mathbb{B}}_{\mathsf{ack}} \otimes \underbrace{\mathbb{B}}_{\mathsf{guess?}}}$$

"Can an adversary peek into a named process?"

Conjecture (Hiding)

For all $\mathcal{A} : \mathsf{Adv}$,

$$\mathsf{adv} \leftarrow \mathcal{A}; \mathsf{exp}(\mathsf{adv}, 0) \approx \mathsf{adv} \leftarrow \mathcal{A}; \mathsf{exp}(\mathsf{adv}, 1)$$

Security Property: Hiding

$$\boxed{\mathsf{Adv} \triangleq \forall\, e.\ \underbrace{@_e \mathbb{B}}_{\mathsf{enc}} \multimap \underbrace{@_e \mathbb{B}}_{\mathsf{ack}} \otimes \underbrace{\mathbb{B}}_{\mathsf{guess?}}}$$

"Can an adversary peek into a named process?"

$\textbf{proc}\ \mathsf{exp}(\mathsf{adv} : \mathsf{Adv}, e : \mathbb{B}) : \mathbb{B} \triangleq$
$\quad \mathsf{adv}.\textbf{send}[e];$
$\quad \mathsf{enc} \leftarrow \langle \mathsf{enc}.\mathsf{send}(@e) \rangle\ ;$
$\quad \mathsf{adv}.\textbf{send}(\mathsf{enc});$
$\quad \mathsf{enc} \leftarrow \mathsf{adv}.\textbf{recv};$
$\quad @e \leftarrow \mathsf{enc}.\textbf{recv};$
$\quad e.\textbf{recv}\{\overline{\cdot \Rightarrow \mathsf{exp}.\textbf{fwd}(\mathsf{adv})}\}$

Conjecture (Hiding)

For all $\mathcal{A} : \mathsf{Adv}$,

$$\mathsf{adv} \leftarrow \mathcal{A}; \mathsf{exp}(\mathsf{adv}, 0) \approx \mathsf{adv} \leftarrow \mathcal{A}; \mathsf{exp}(\mathsf{adv}, 1)$$

Security Property: Binding

$$\boxed{\mathsf{Adv} \triangleq \exists\, a.\ \underbrace{@_a \mathbb{B}}_{\text{com}} \otimes \underbrace{@_a \mathbb{B}}_{\text{ack}} \multimap \underbrace{\mathbb{B}}_{\text{test}} \multimap \underbrace{@_a \mathbb{B}}_{\text{switch?}}}$$

"Can an adv. switch a named process?"

Security Property: Binding

$$\boxed{\mathsf{Adv} \triangleq \exists\, a.\ \underbrace{@_a \mathbb{B}}_{\text{com}} \otimes \underbrace{@_a \mathbb{B}}_{\text{ack}} \multimap \underbrace{\mathbb{B}}_{\text{test}} \multimap \underbrace{@_a \mathbb{B}}_{\text{switch?}}}$$

"Can an adv. switch a named process?"

Conjecture (Binding)

For all $\mathcal{A} : \mathsf{Adv}$,

$$\mathsf{adv} \leftarrow \mathcal{A}; \mathsf{exp}(\mathsf{adv}, 0) \approx \mathsf{adv} \leftarrow \mathcal{A}; \mathsf{exp}(\mathsf{adv}, 1)$$

Security Property: Binding

$$\boxed{\mathsf{Adv} \triangleq \exists\, a.\ \underbrace{@_a \mathbb{B}}_{\mathsf{com}} \otimes \underbrace{@_a \mathbb{B}}_{\mathsf{ack}} \multimap \underbrace{\mathbb{B}}_{\mathsf{test}} \multimap \underbrace{@_a \mathbb{B}}_{\mathsf{switch?}}}$$

"Can an adv. switch a named process?"

$\textbf{proc}\ \mathsf{exp}(\mathsf{adv} : \mathsf{Adv}, e : \mathbb{B}) : \mathbb{B} \triangleq$

$\quad [a] \leftarrow \mathsf{adv}.\textbf{recv};$

$\quad \mathsf{com} \leftarrow \mathsf{adv}.\textbf{recv};$

$\quad \mathsf{adv}.\textbf{send}(\mathsf{com});$

$\quad \mathsf{adv}.\textbf{send}(e);$

$\quad @a \leftarrow \mathsf{adv}.\textbf{recv};$

$\quad \mathsf{exp}.\textbf{fwd}(a)$

Conjecture (Binding)

For all $\mathcal{A} : \mathsf{Adv}$,

$$\mathsf{adv} \leftarrow \mathcal{A}; \mathsf{exp}(\mathsf{adv}, 0) \approx \mathsf{adv} \leftarrow \mathcal{A}; \mathsf{exp}(\mathsf{adv}, 1)$$

Secure Coin Flipping in STL[3]

$$\mathsf{Flip} \triangleq \forall\, c.\ \underbrace{@_c\,\mathbb{B}}_{\mathsf{com}} \multimap \underbrace{@_c\,\mathbb{B}}_{\mathsf{ack}} \otimes \underbrace{\mathbb{B}}_{\mathsf{flip}} \otimes \underbrace{@_c\,\mathbb{B}}_{\mathsf{opn}} \multimap \mathbb{1}$$

Secure Coin Flipping in STL[3]

$$\mathsf{Flip} \triangleq \forall\, c.\ \underbrace{@_c\,\mathbb{B}}_{\mathsf{com}} \multimap \underbrace{@_c\,\mathbb{B}}_{\mathsf{ack}} \otimes \underbrace{\mathbb{B}}_{\mathsf{flip}} \otimes \underbrace{@_c\,\mathbb{B}}_{\mathsf{opn}} \multimap \mathbb{1}$$

proc flip() : Flip $\triangleq$

  $[c]$(com) $\leftarrow$ flip.**recv**;

  flip.**send**(com);

  flip.**send** $\langle f \leftarrow$ sample()$\rangle$;

  com $\leftarrow$ flip.**recv**;

  $@c \leftarrow$ com.**recv**;

  $\ldots$ *use the result,* c

  flip.**close**

Secure Coin Flipping in STL[3]

$\text{Flip} \triangleq \forall\, c.\ \underbrace{@_c\,\mathbb{B}}_{\text{com}} \multimap \underbrace{@_c\,\mathbb{B}}_{\text{ack}} \otimes \underbrace{\mathbb{B}}_{\text{flip}} \otimes \underbrace{@_c\,\mathbb{B}}_{\text{opn}} \multimap \mathbb{1}$ $\quad \text{Call} \triangleq \exists\, c.\ \underbrace{@_c\,\mathbb{B}}_{\text{com}} \otimes \underbrace{@_c\,\mathbb{B}}_{\text{ack}} \multimap \underbrace{\mathbb{B}}_{\text{flip}} \multimap \underbrace{@_c\,\mathbb{B}}_{\text{opn}}$

proc flip() : Flip $\triangleq$

$[c]$(com) $\leftarrow$ flip.**recv**;

flip.**send**(com);

flip.**send** $\langle f \leftarrow$ sample()$\rangle$;

com $\leftarrow$ flip.**recv**;

$@c \leftarrow$ com.**recv**;

. . . *use the result,* c

flip.**close**

Secure Coin Flipping in STL[3]

$$\mathsf{Flip} \triangleq \forall\, c.\ \underbrace{@_c\,\mathbb{B}}_{\mathsf{com}} \multimap \underbrace{@_c\,\mathbb{B}}_{\mathsf{ack}} \otimes \underbrace{\mathbb{B}}_{\mathsf{flip}} \otimes \underbrace{@_c\,\mathbb{B}}_{\mathsf{opn}} \multimap \mathbb{1}$$

$\textbf{proc}\ \mathsf{flip}()\ :\ \mathsf{Flip} \triangleq$

- $[c](\mathsf{com}) \leftarrow \mathsf{flip}.\textbf{recv};$
- $\mathsf{flip}.\textbf{send}(\mathsf{com});$
- $\mathsf{flip}.\textbf{send}\,\langle f \leftarrow \mathsf{sample}()\rangle\,;$
- $\mathsf{com} \leftarrow \mathsf{flip}.\textbf{recv};$
- $@c \leftarrow \mathsf{com}.\textbf{recv};$
- $\ldots$ *use the result,* c
- $\mathsf{flip}.\textbf{close}$

$$\mathsf{Call} \triangleq \exists\, c.\ \underbrace{@_c\,\mathbb{B}}_{\mathsf{com}} \otimes \underbrace{@_c\,\mathbb{B}}_{\mathsf{ack}} \multimap \underbrace{\mathbb{B}}_{\mathsf{flip}} \multimap \underbrace{@_c\,\mathbb{B}}_{\mathsf{opn}}$$

$\textbf{proc}\ \mathsf{call}()\ :\ \mathsf{Call} \triangleq$

- $c \leftarrow \mathsf{guess}();$
- $\mathsf{call}.\textbf{send}[c]\,\langle \mathsf{com} \leftarrow \mathsf{com}.\textbf{send}(@c)\rangle\,;$
- $\mathsf{com} \leftarrow \mathsf{call}.\textbf{recv};$
- $f \leftarrow \mathsf{call}.\textbf{recv};$
- $\ldots$ *use the result,* f
- $\mathsf{call}.\textbf{fwd}(\mathsf{com})$

Coin Flipping: Flipper Security

$$\text{Call} \triangleq \exists\, c.\ @_c\mathbb{B} \otimes @_c\mathbb{B} \multimap \mathbb{B} \multimap @_c\mathbb{B}$$

$$
\begin{aligned}
&\textbf{proc}\ \text{exp}(\text{call} : \text{Call}, b : \mathbb{B}) : \mathbb{B} \triangleq \\
&\quad [c](\text{com}) \leftarrow \text{call}.\textbf{recv}; \\
&\quad \text{call}.\textbf{send}(\text{com}); \\
&\quad \text{call}.\textbf{send}(b); \\
&\quad \text{com} \leftarrow \text{call}.\textbf{recv}; \\
&\quad @c \leftarrow \text{com}.\textbf{recv}; \\
&\quad \text{exp}.\textbf{fwd}(c)
\end{aligned}
$$

Conjecture (Flipper Security)

Forall $\mathcal{C}$ :: (call : Call),

$$\text{call} \leftarrow \mathcal{C}; \text{exp}(\text{call}, 0) \approx \text{call} \leftarrow \mathcal{C}; \text{exp}(\text{call}, 1)$$

Coin Flipping: Caller Security

$$\text{Flip} \triangleq \forall\, c.\ @_c\,\mathbb{B} \multimap @_c\,\mathbb{B} \otimes \mathbb{B} \otimes @_c\,\mathbb{B} \multimap \mathbb{1}$$

$$
\begin{array}{l}
\textbf{proc}\ \text{exp}(\text{flip} : \text{Flip}, c : \mathbb{B}) : \mathbb{B} \triangleq \\
\quad \text{flip}.\textbf{send}[c]; \\
\quad \text{com} \leftarrow \langle \text{com}.\textbf{send}(@c) \rangle\ ; \\
\quad \text{flip}.\textbf{send}(\text{com}); \\
\quad \text{com} \leftarrow \text{flip}.\textbf{recv}; \\
\quad f \leftarrow \text{flip}.\textbf{recv}; \\
\quad \text{flip}.\textbf{send}(\text{com}); \\
\quad \text{flip}.\textbf{wait}; \\
\quad \text{exp}.\textbf{fwd}(f)
\end{array}
$$

Conjecture (Caller Security)

Forall $\mathcal{F}$ :: (flip : Flip),

$$\text{flip} \leftarrow \mathcal{F}; \text{exp}(\text{flip}, 0) \approx \text{flip} \leftarrow \mathcal{F}; \text{exp}(\text{flip}, 1)$$

But... we can’t prove it yet :)

Relationship to Type Polymorphism

No Reveal $\forall\, i.\ @_i\,\mathbb{B}^n \multimap @_i\,\mathbb{B}^n \quad \rightsquigarrow \quad \forall\, \alpha.\ \alpha \multimap \alpha$

Relationship to Type Polymorphism

No Reveal	$\forall\, i.\ @_i\,\mathbb{B}^n \multimap @_i\,\mathbb{B}^n$	$\rightsquigarrow$	$\forall\, \alpha.\ \alpha \multimap \alpha$
Partial Reveal	$\forall\, i.\ @_i\,\mathbb{B}^{m+n} \multimap @_i\,\mathbb{B}^n$	$\rightsquigarrow$	$\forall\, \alpha.\ \mathbb{B}^m[\alpha] \multimap \alpha$

Relationship to Type Polymorphism

No Reveal	$\forall\, i.\ @_i\,\mathbb{B}^n \multimap @_i\,\mathbb{B}^n$	$\rightsquigarrow$	$\forall\, \alpha.\ \alpha \multimap \alpha$
Partial Reveal	$\forall\, i.\ @_i\,\mathbb{B}^{m+n} \multimap @_i\,\mathbb{B}^n$	$\rightsquigarrow$	$\forall\, \alpha.\ \mathbb{B}^m[\alpha] \multimap \alpha$

Cond. Reveal	$\forall\, i.\ @_i\,\mathbb{B}^n \multimap \mathbb{1} \mathbin{\&} @_i\,\mathbb{B}^n$	$\not\rightsquigarrow$	$\forall\, \alpha.\ \alpha \multimap \big(\underbrace{(\alpha \multimap \mathbb{B})}_{\text{not binding!}} \multimap \mathbb{1}\big) \mathbin{\&} \alpha$

Relationship to Type Polymorphism

No Reveal	$\forall\, i.\ @_i\,\mathbb{B}^n \multimap @_i\,\mathbb{B}^n$	$\rightsquigarrow$	$\forall\, \alpha.\ \alpha \multimap \alpha$
Partial Reveal	$\forall\, i.\ @_i\,\mathbb{B}^{m+n} \multimap @_i\,\mathbb{B}^n$	$\rightsquigarrow$	$\forall\, \alpha.\ \mathbb{B}^m[\alpha] \multimap \alpha$

Cond. Reveal	$\forall\, i.\ @_i\,\mathbb{B}^n \multimap \mathbb{1}\ \&\ @_i\,\mathbb{B}^n$	$\not\rightsquigarrow$	$\forall\, \alpha.\ \alpha \multimap \Big(\underbrace{(\alpha \multimap \mathbb{B})}_{\text{not binding!}} \multimap \mathbb{1}\Big)\ \&\ \alpha$
Type-Sensitive	$\forall\, i.\ @_i\,(\mathbb{N} \multimap \mathbb{B}) \multimap @_i\,\mathbb{B}$	$\not\rightsquigarrow$	$\forall\, \alpha.\ \underbrace{(\mathbb{N} \multimap \alpha)}_{\text{not ZK!}} \multimap \alpha$

Logical Relations[5]

$$
\boxed{\begin{array}{lclll}
(v_1, v_2) & \in & \mathcal{V}[\![\forall\,\alpha.\,T]\!]\rho & \text{iff} & \text{for all types } S_1, S_2 \text{ and candidates } R, \\
 & & & & (v_1[S_1], v_2[S_2]) \in \mathcal{E}[\![\,T]\!]\rho[\alpha \mapsto R] \\
(v_1, v_2) & \in & \mathcal{V}[\![\alpha]\!]\rho & \text{iff} & (v_1, v_2) \in \rho(\alpha)
\end{array}}
$$

Conditional revealing means candidacy must be more strict: it is constrained by how the name is used in the continuation type.

[5] Reynolds 1983.

Candidates?

No Restriction $\forall\, i.\ @_i\, \mathbb{B}^n \multimap @_i\, \mathbb{B}^n$

Candidates?

No Restriction	$\forall\, i.\ @_i\, \mathbb{B}^n \multimap @_i\, \mathbb{B}^n$
Equivalent	$\forall\, i.\ @_i\, \mathbb{B}^n \multimap \mathbb{1}$

Candidates?

No Restriction	$\forall\, i.\ @_i\, \mathbb{B}^n \multimap @_i\, \mathbb{B}^n$
Equivalent	$\forall\, i.\ @_i\, \mathbb{B}^n \multimap \mathbb{1}$
	$\forall\, i.\ @_i\, \mathbb{B}^n \multimap \mathbb{1} \oplus @_i\, \mathbb{B}^n$

Candidates?

No Restriction	$\forall\, i.\ @_i\,\mathbb{B}^n \multimap @_i\,\mathbb{B}^n$
Equivalent	$\forall\, i.\ @_i\,\mathbb{B}^n \multimap \mathbb{1}$
	$\forall\, i.\ @_i\,\mathbb{B}^n \multimap \mathbb{1} \oplus @_i\,\mathbb{B}^n$
$\frac{1}{2}$ **Equivalent**	$\forall\, i.\ @_i\,\mathbb{B}^{2n} \multimap @_i\,\mathbb{B}^n$

Candidates?

No Restriction	$\forall\, i.\ @_i\,\mathbb{B}^n \multimap @_i\,\mathbb{B}^n$
Equivalent	$\forall\, i.\ @_i\,\mathbb{B}^n \multimap \mathbb{1}$
	$\forall\, i.\ @_i\,\mathbb{B}^n \multimap \mathbb{1} \oplus @_i\,\mathbb{B}^n$
$\frac{1}{2}$ **Equivalent**	$\forall\, i.\ @_i\,\mathbb{B}^{2n} \multimap @_i\,\mathbb{B}^n$
Conditionally Equivalent	$\forall\, i.\ @_i\,\mathbb{B}^n \multimap \mathbb{1}\ \&\ @_i\,\mathbb{B}^n$

Other Proof Ideas

- Computation focusing (Rioux and Zdancewic 2020)
- Theorems for free from separation logic specifications (Birkedal et al. 2021)
- Session logical relations for noninterference (Derakhshan, Balzer, and Jia 2021)
- Suggestions?