Session Logical Relation for Noninterference

Farzaneh Derakhshani
Joint work with Stephanie Balzer and Limin Jia
PLAS 2021
Session types in a nutshell

Propagation of sensitive information

Session type: Protocol for message exchange along channels

?int;!bool;1

Message passing concurrency paradigm

Erlang, Go, Rust
Information flow control
Information flow control

Protocols

alice-chn: customer = auth → 1

alice-auth: auth = \&\{tok_1: \oplus \{succ: account \otimes 1, fail: 1\}, \ldots, tok_n: \oplus \{succ: account \otimes 1, fail: 1\}\}
Information flow control

**Process term**

Bank:

send alice-auth alice-chn;
send bob-auth bob-chn;

Protocols

alice-chn: customer = auth \rightarrow 1

alice-auth: auth = \&\{ tok_1: \oplus \{ succ: account \otimes 1, fail: 1 \}, \ldots, 
               tok_n: \oplus \{ succ: account \otimes 1, fail: 1 \} \}
Information flow control

Process term

Bank:
send alice-auth alice-chn;
send bob-auth bob-chn;

Protocols

alice-chn: customer = auth → 1
alice-auth: auth = &{tok_1: \oplus \{succ: account \otimes 1, fail: 1\}, \ldots, tok_n: \oplus \{succ: account \otimes 1, fail: 1\}}
Information flow control

Bank:
send alice-auth alice-chn;
send bob-auth bob-chn;

Protocols
alice-chn: customer = auth → 1
alice-auth: auth = \&\{tok_1: \oplus \{succ: account \otimes 1, fail: 1\}, ..., \\
tok_n: \oplus \{succ: account \otimes 1, fail: 1\}\}
Information flow control

Process term

Bank:
send alice-auth alice-chn;
send bob-auth bob-chn;

Protocols

alice-chn: customer = auth → 1

alice-auth: auth = &{tok₁: ⊕ {succ: account ⊗ 1, fail: 1},..., tokₙ: ⊕ {succ: account ⊗ 1, fail: 1}}
Information flow control

Process term

Bank:

send alice-auth alice-chn;
send bob-auth bob-chn;

Protocols

alice-chn: customer = auth → 1

alice-auth: auth = \&\{tok_1: \oplus \{succ: account \otimes 1, fail: 1\}, ..., 
               tok_n: \oplus \{succ: account \otimes 1, fail: 1\}\}
Information flow control

**Process term**

Alice–Auth–Process:
\[
\text{case alice-auth}(tok_{\text{yellow}} \Rightarrow \text{alice-auth}.\text{succ}; \\
\quad \text{send alice-acc alice-auth}; \cdots \\
\mid tok_{i \neq \text{yellow}} \Rightarrow \text{alice-auth}.\text{fail}; \cdots )
\]

**Protocols**

\[
alice-chn: \text{customer} = \text{auth} \to 1
\]

\[
alice-auth: \text{auth} = \&\{tok_1: \oplus \{\text{succ: account } \otimes 1, \text{fail: 1}\}, \ldots, \\
\quad tok_n: \oplus \{\text{succ: account } \otimes 1, \text{fail: 1}\}\}
\]
Information leakage: indirect flow

\[
\text{SneakyaAuth} : \\
\text{case } x \cdot \begin{cases} 
\text{tok}_{\text{yellow}} \Rightarrow x.\text{succ}; u.s; z.s; // \text{insecure send} \\
\text{tok}_{i \neq \text{yellow}} \Rightarrow x.\text{fail}; u.f; z.f; // \text{insecure send}
\end{cases}
\]
Information leakage: indirect flow

**Process term**

SneakyaAuth:

\[
\text{case } x \begin{cases} 
(tok_{\text{yellow}} \Rightarrow x.\text{succ}; u.s; z.s; // insecure send) \\
(tok_{i \neq \text{yellow}} \Rightarrow x.\text{fail}; u.f; z.f; // insecure send)
\end{cases}
\]
Information leakage: indirect flow
Information leakage: indirect flow

The secret vs Not secret

VS

SUCCESS

FAIL
IFC for message passing concurrency

Direct and indirect malicious leakages can be prevented by an information flow control (IFC) type system.
IFC for message passing concurrency

Direct and indirect malicious leakages can be prevented by an information flow control (IFC) type system.

Session types to prescribe the protocols of message passing systems.
IFC for message passing concurrency

Direct and indirect malicious leakages can be prevented by an information flow control (IFC) type system.

Session types to prescribe the protocols of message passing systems.

➔ Enrich session types to prevent information leakage.

➔ Capture noninterference with a novel logical relation.
Noninterference

Program equivalence up to observable messages

→ **Assume**: a process receives related messages along low-secrecy channels.

→ **Assert**: it sends the same messages along those channels.
Assume: a process receives related messages along low-secrecy channels.

Assert: it sends the same messages along those channels.
Noninterference: Program equivalence up to observable messages

- **Assume**: a process receives related messages along low-secrecy channels.
- **Assert**: it sends the same messages along those channels.
Maximal secrecy

★ The **security clearance** of a process.

★ The *maximum secrecy* that a process can receive *w/o violating the security lattice*. 


Maximal secrecy: The security clearance of the process

Security lattice $\Psi_{Bank}$

Bank
Alice (A)       Bob (B)
Guest (G)
Running secrecy

Reflection of **the level of secret information a process has obtained** so far.
Running secrecy: the highest level of secret information obtained so far

Security lattice:

SneakyaAuth:
\[
\text{case } x (\text{tok}_{\text{yellow}} \Rightarrow x.\text{succ}; u.s; z.s; \\
\quad \mid \text{tok}_{i \neq \text{yellow}} \Rightarrow x.\text{fail}; u.f; z.f;)\]
Running secrecy: the highest level of secret information obtained so far

Indirect flow

Does not type check Process of running secrecy cannot send along z of lower secrecy Guest.

SneakyaAuth:

\[
\begin{align*}
\text{case } x (t_{\text{yellow}}) & \rightarrow x.\text{succ}; u.s; z.s; // \text{ insecure send} \\
\text{im} & \rightarrow x.\text{fail}; u.f; z.f; // \text{ insecure send}
\end{align*}
\]
Typing judgments with possible worlds

\[ \Delta \vdash P \quad :: \quad (x : A) \]
Typing judgments with possible worlds

\[ \Delta \vdash P :: (x:A) \]

\[ \text{alice-auth}, \text{bob-auth}, \text{alice-chn}, \text{bob-chn} \vdash \text{Bank :: (bank-chn:1)} \]
Typing judgments with possible worlds

\[ \Delta \vdash P :: (x:A) \]
Typing judgments with possible worlds

\[ \Psi; \Delta \vdash P_{@c} :: (x:A[d]) \]

Security lattice

Maximal secrecy

Running secrecy
Typing judgments with possible worlds

\[ \Psi; \Delta \vdash P[@c :: (x:A[d])] \]

The running secrecy is a reflection of the level of secret information a process has obtained so far.

1. **Adjust on receives:** increase the running secrecy to **at least** the secrecy of the channel you receive from,

2. **Guard on sends:** the running secrecy of the sending process is **at most** the secrecy of the channel you send to.
Typing judgments with possible worlds

\[ \Psi; \Delta \vdash P^{\@c} :: (x : A^{[d]}) \]

- Security lattice
- Maximal secrecy
- Running secrecy
Typing judgments with possible worlds

\[ \Psi; \Delta \vdash P@c :: (x:A[d]) \]

Tree invariant:

1. the maximal secrecy of a child is at most as high as the parent's node,
2. the running secrecy of the parent node is capped by its maximal secrecy (d).

A node can never obtain more secrets than it is licensed to.
Main contributions

- **IFC type system** for intuitionistic linear binary session types using possible worlds
- **Session logical relation** for noninterference supporting open programs

\[(C_1 D_1 F_1, C_2 D_2 F_2) \in E^\xi_\psi [\Delta \vdash K]\]
Future work

- **Noninterference of recursive session-types**
  - Progress sensitive
  - **Progress insensitive** system with **certified downgrading**
    - more flexible but not as safe!

- Integrate our results with **sharing**
Conclusions

Summary:

- **IFC type system** for intuitionistic linear binary session types using possible worlds
- **Session logical relation** for noninterference supporting open programs

Observations:

- Session types make explicit knowledge of information learned through *message exchange*
- Session logical relation allows for more *nuanced equality expression*, possibly paving the way for other investigations
- Possible worlds bear resemblance to **Kripke logical relations**, yet *internalizing the worlds into the type system*