Session Logical Relation for Noninterference

Farzaneh Derakhshan
Joint work with Stephanie Balzer and Limin Jia
PLAS 2021

Session types in a nutshell

Propagation of sensitive information

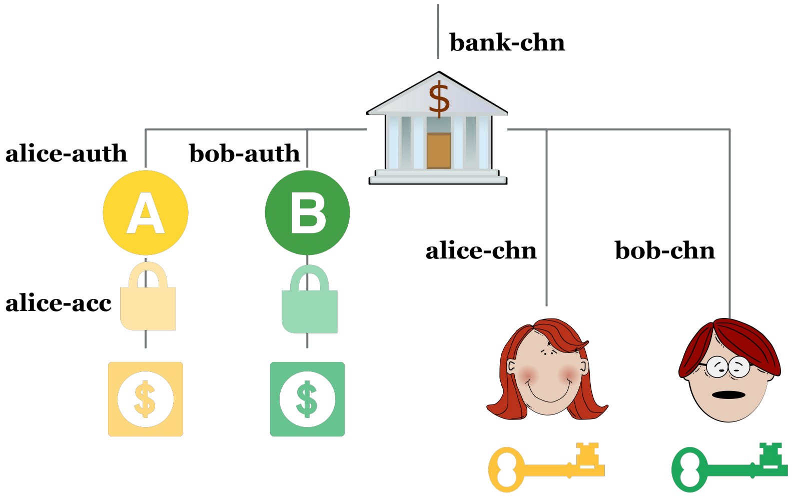
Session type: Protocol for message exchange along channels

?int;!bool;1

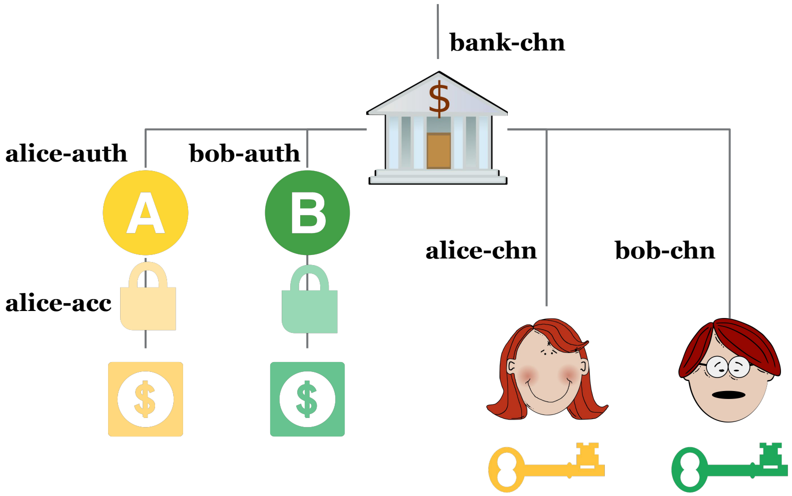
Message passing concurrency paradigm

Erlang, Go, Rust

Information flow control



Information flow control

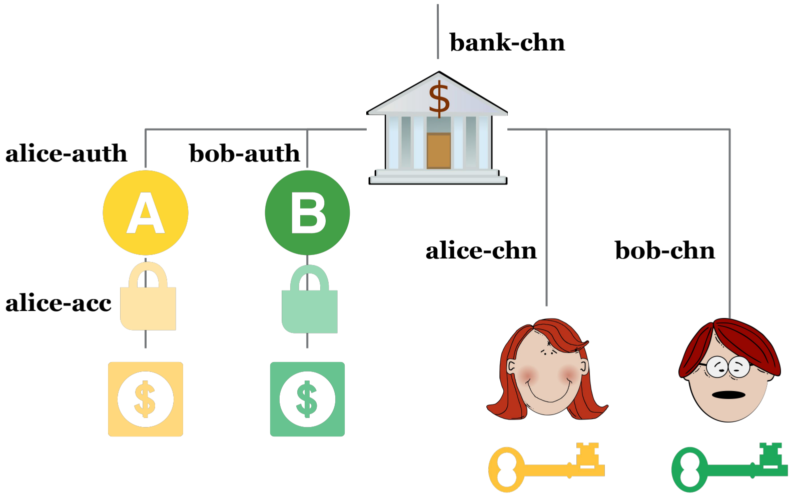


[*Protocols*]

$$\texttt{alice-chn} : \; \mathsf{customer} = \mathsf{auth} \multimap 1$$

$$\texttt{alice-auth} : \; \mathsf{auth} = \&\{tok_1 : \oplus\{succ : \mathsf{account} \otimes 1, fail : 1\}, \ldots, tok_n : \oplus\{succ : \mathsf{account} \otimes 1, fail : 1\}\}$$

Information flow control



Process term

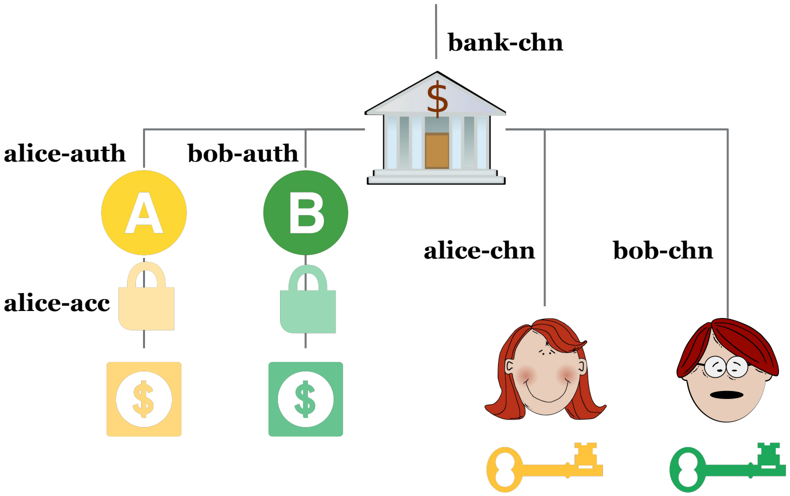
$$\textbf{Bank}: \\ \quad \texttt{send alice-auth alice-chn;} \\ \quad \texttt{send bob-auth bob-chn;}$$

Protocols

$$\texttt{alice-chn}: \ \textsf{customer} = \textsf{auth} \multimap 1$$

$$\texttt{alice-auth}: \ \textsf{auth} = \&\{tok_1 : \oplus\{succ: \textsf{account} \otimes 1, fail: 1\}, \ldots, \\ tok_n : \oplus\{succ: \textsf{account} \otimes 1, fail: 1\}\}$$

Information flow control



Process term

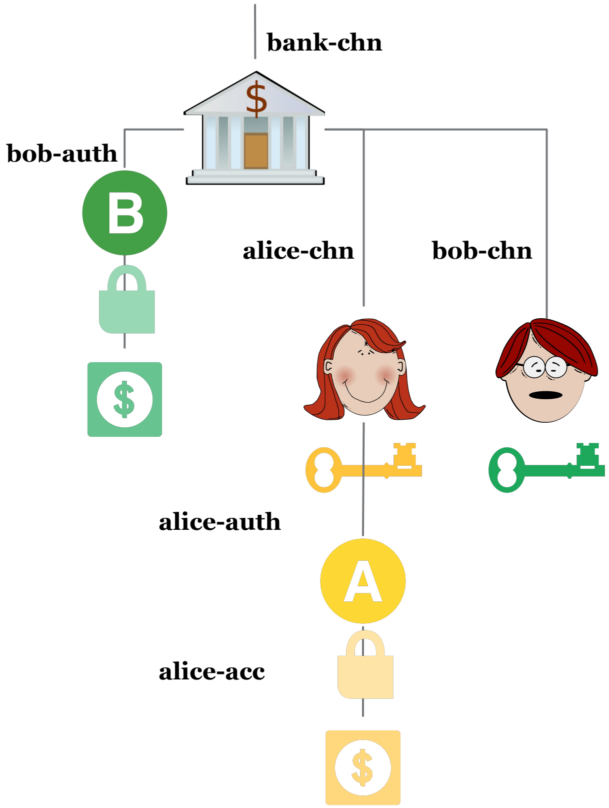
Bank :
    send alice–auth alice–chn;
    send bob–auth bob–chn;

Protocols

$$\texttt{alice–chn}: \ \mathsf{customer} = \mathsf{auth} \multimap 1$$

$$\texttt{alice–auth}: \ \mathsf{auth} = \&\{tok_1 \colon \oplus \{succ \colon \mathsf{account} \otimes 1, fail \colon 1\}, \ldots, tok_n \colon \oplus \{succ \colon \mathsf{account} \otimes 1, fail \colon 1\}\}$$

Information flow control



Process term

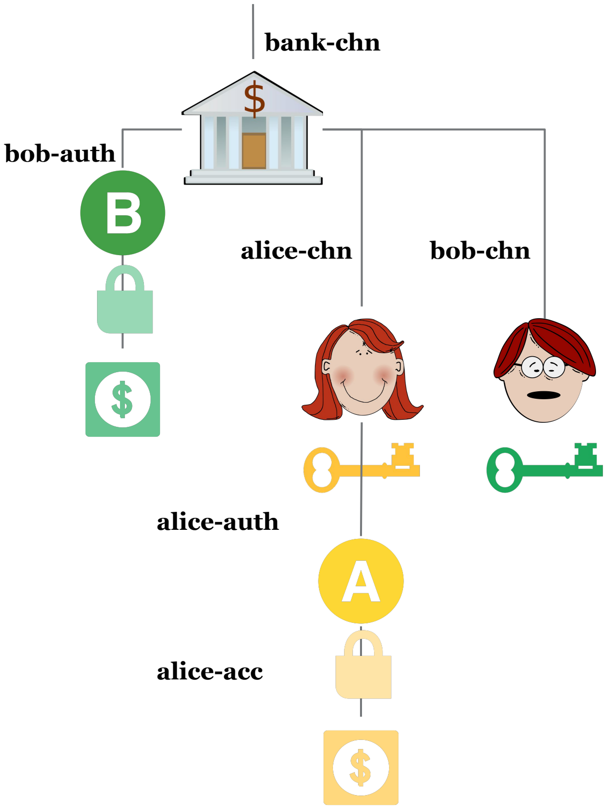
Bank :
  send alice−auth alice−chn;
  send bob−auth bob−chn;

Protocols

$\texttt{alice-chn} : \; \mathsf{customer} = \mathsf{auth} \multimap 1$

$\texttt{alice-auth} : \; \mathsf{auth} = \&\{tok_1 : \oplus\{succ : \mathsf{account} \otimes 1, fail : 1\}, \ldots, tok_n : \oplus\{succ : \mathsf{account} \otimes 1, fail : 1\}\}$

Information flow control



Process term

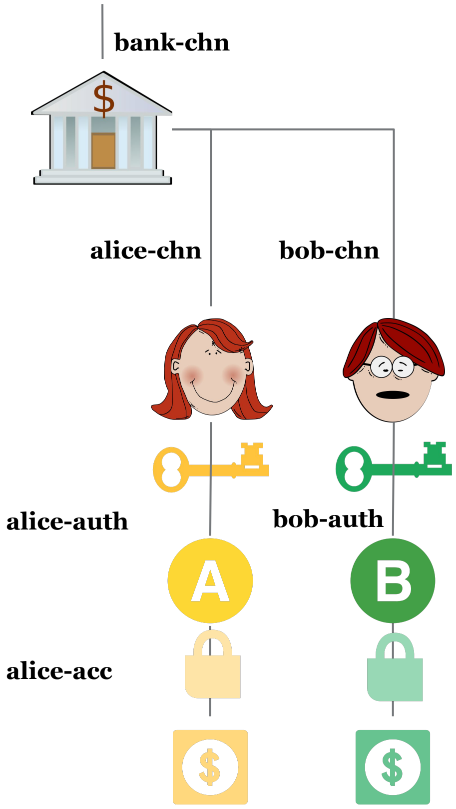
$$\textbf{Bank}: \; \texttt{send alice-auth alice-chn}; \; \texttt{send bob-auth bob-chn};$$

Protocols

$$\texttt{alice-chn}: \; \mathsf{customer} = \mathsf{auth} \multimap 1$$

$$\texttt{alice-auth}: \; \mathsf{auth} = \&\{tok_1 : \oplus\{succ : \mathsf{account} \otimes 1, fail : 1\}, \ldots, tok_n : \oplus\{succ : \mathsf{account} \otimes 1, fail : 1\}\}$$

Information flow control



Process term

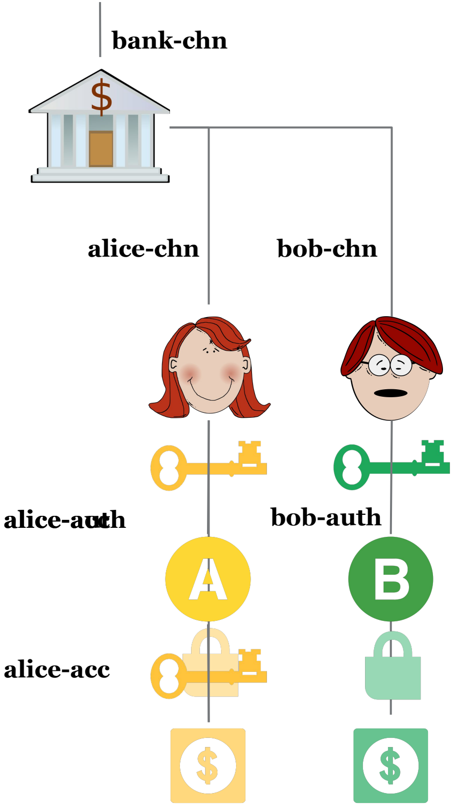
Bank :
  send alice−auth alice−chn;
  send bob−auth bob−chn;

Protocols

$$\texttt{alice-chn} : \ \mathsf{customer} = \mathsf{auth} \multimap 1$$

$$\texttt{alice-auth} : \ \mathsf{auth} = \&\{tok_1 : \oplus\{succ : \mathsf{account} \otimes 1, fail : 1\}, \ldots, tok_n : \oplus\{succ : \mathsf{account} \otimes 1, fail : 1\}\}$$

Information flow control



Process term

Alice–Auth–Process :

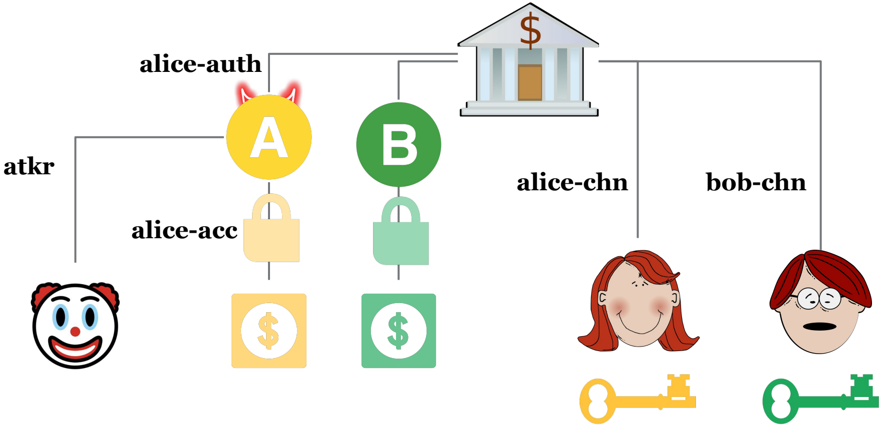
$$\textbf{case}\ \texttt{alice–auth}\ (tok_{yellow} \Rightarrow \texttt{alice–auth}.succ;\ \textbf{send}\ \texttt{alice–acc}\ \texttt{alice–auth}; \cdots \mid tok_{i \neq yellow} \Rightarrow \texttt{alice–auth}.fail; \cdots))$$

Protocols

$$\texttt{alice–chn} : \ \mathsf{customer} = \mathsf{auth} \multimap 1$$

$$\texttt{alice–auth} : \ \mathsf{auth} = \&\{tok_1 : \oplus\{succ : \mathsf{account} \otimes 1, fail : 1\}, \ldots, tok_n : \oplus\{succ : \mathsf{account} \otimes 1, fail : 1\}\}$$

Information leakage: indirect flow

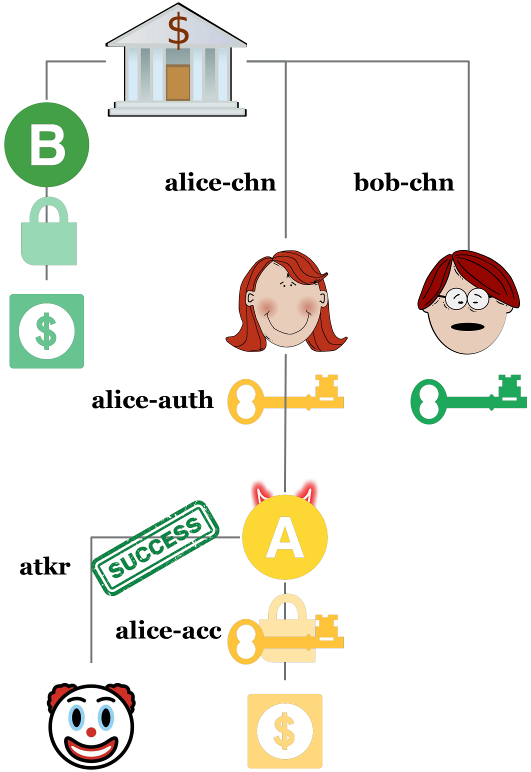


[Process term](#)

SneakyaAuth :

$$\textbf{case}\; x\; (tok_{yellow} \Rightarrow x.succ; u.s; z.s; \text{// insecure send}$$
$$\mid tok_{i \neq yellow} \Rightarrow x.fail; u.f; z.f; \text{// insecure send})$$

Information leakage: indirect flow

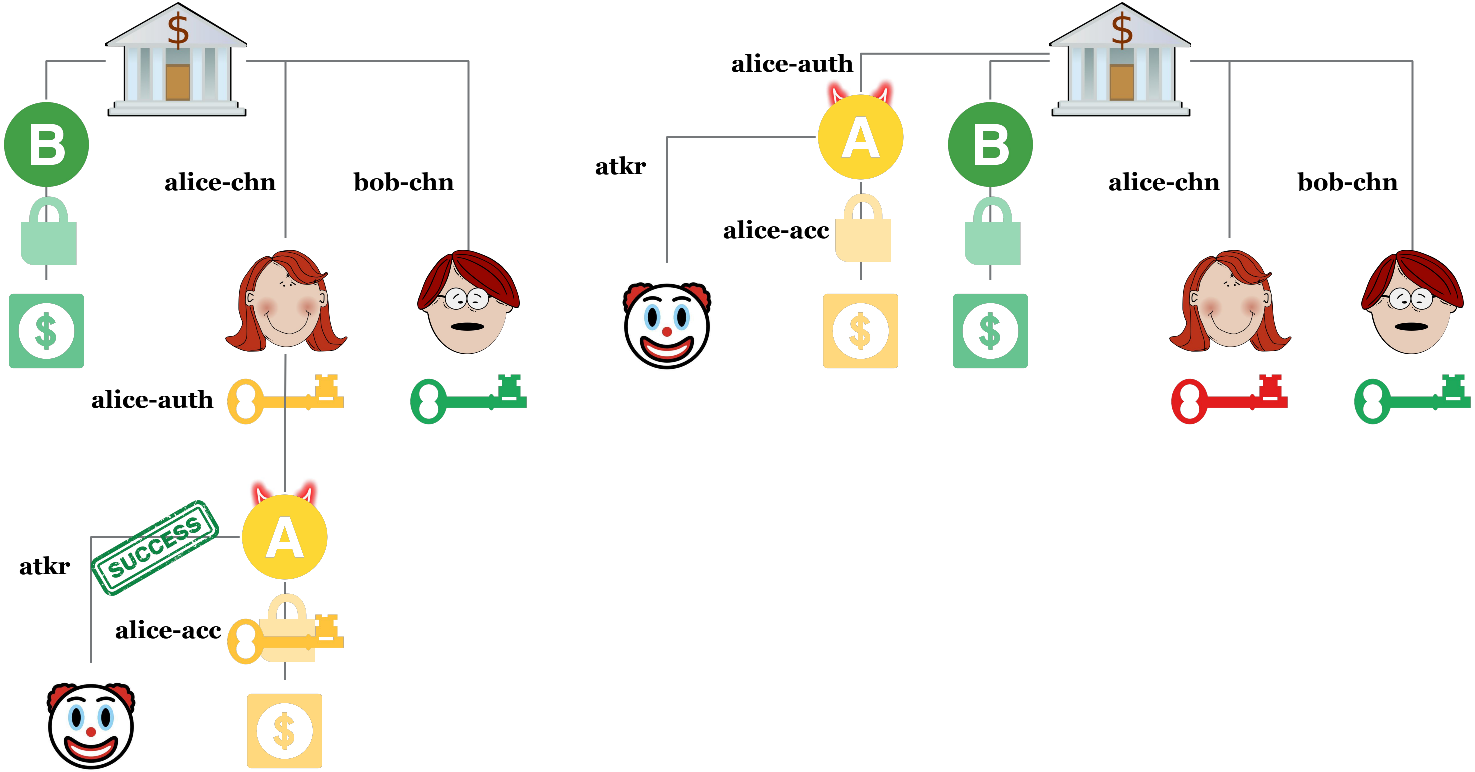


[Process term]

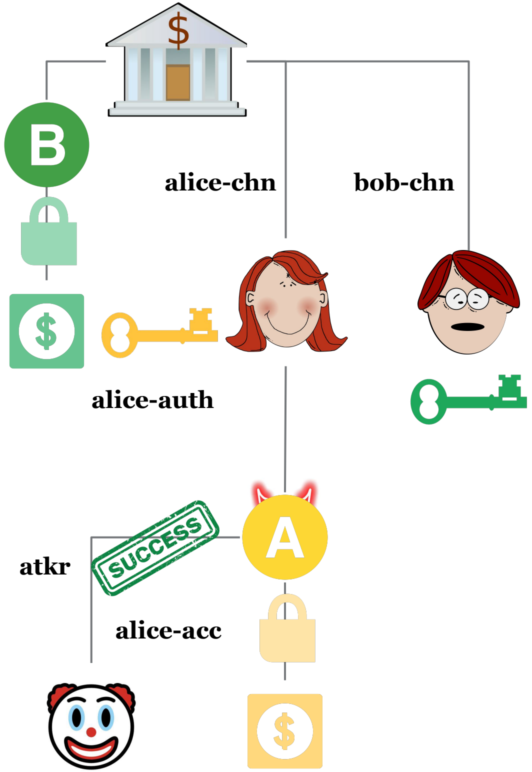
SneakyaAuth :

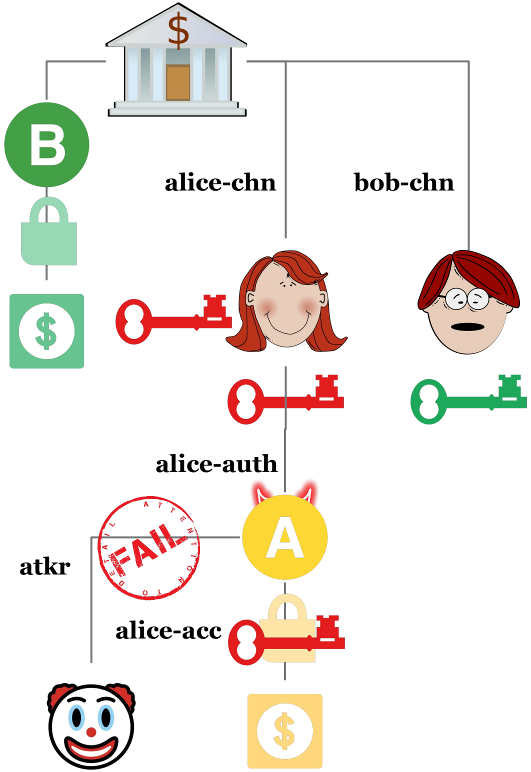
$$\textbf{case}\, x\, (tok_{yellow} \Rightarrow x.succ; u.s; z.s; \; // \text{ insecure send} \mid tok_{i \neq yellow} \Rightarrow x.fail; u.f; z.f; \; // \text{ insecure send})$$

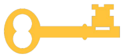
Information leakage: indirect flow



Information leakage: indirect flow





The secret  vs 

Not secret  vs 

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.***

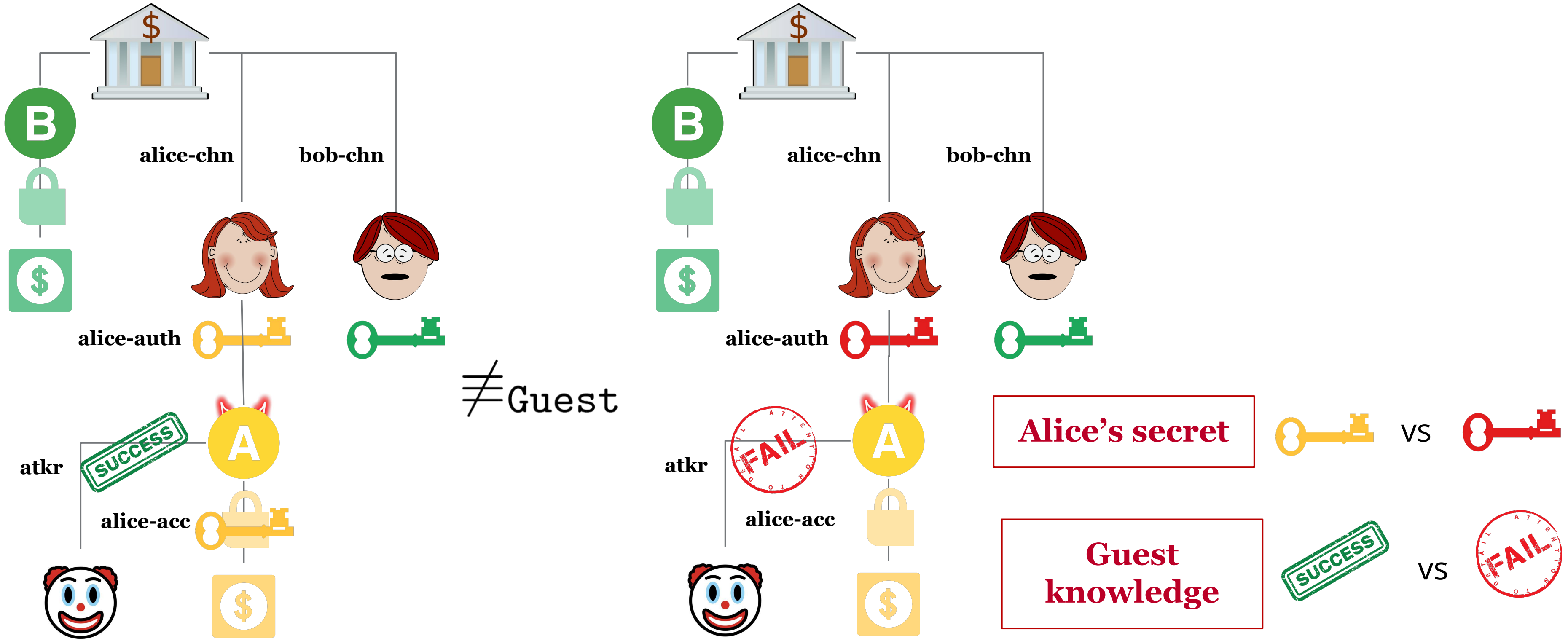
➔ *Capture* **noninterference** *with* ***a novel logical relation.***

Noninterference

Program equivalence up to observable messages

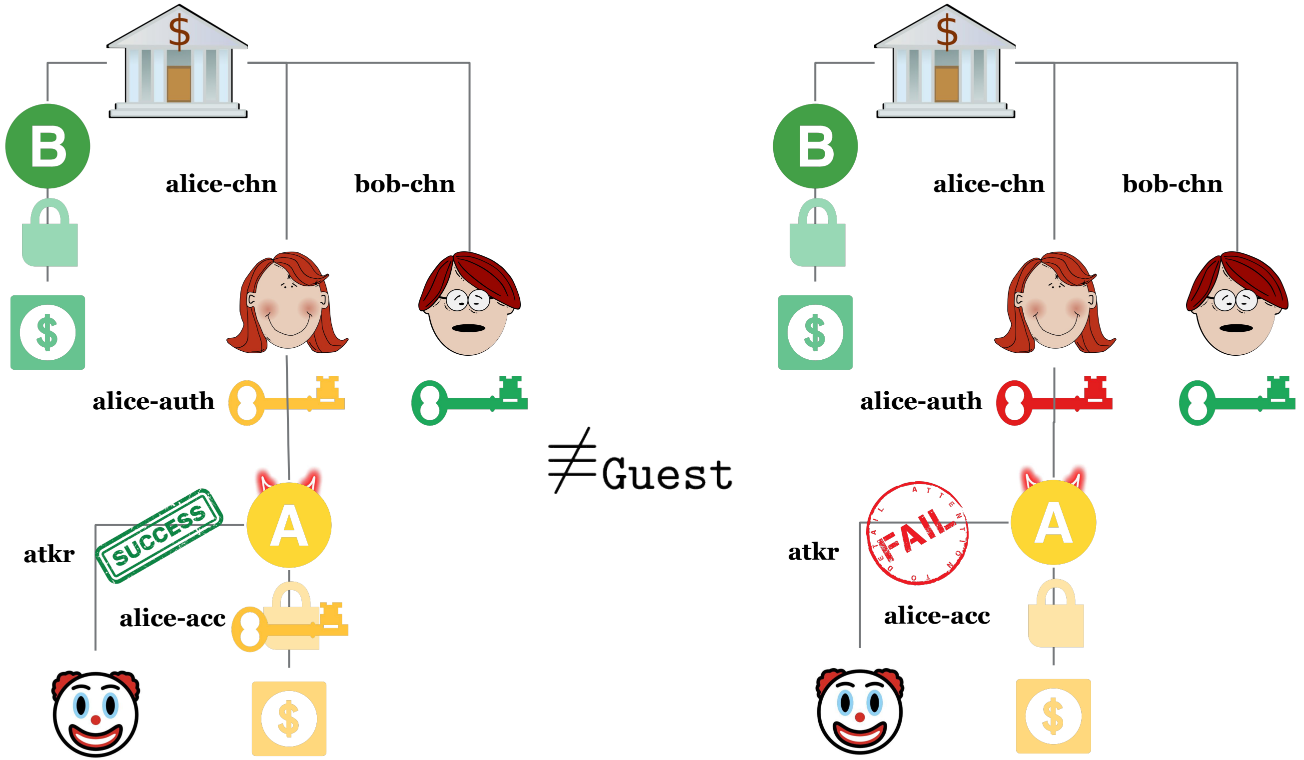
- ➔ ***Assume***: a process receives related messages along low-secrecy channels.
- ➔ ***Assert***: it sends the same messages along those channels.

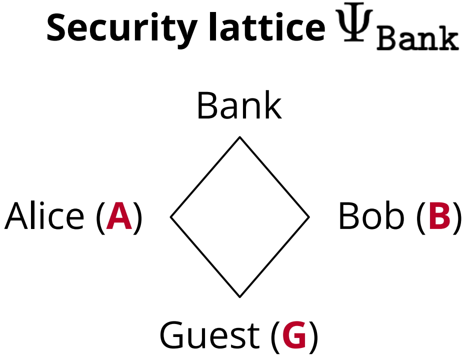
Noninterference: Program equivalence up to observable messages



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

Noninterference: Program equivalence up to observable messages



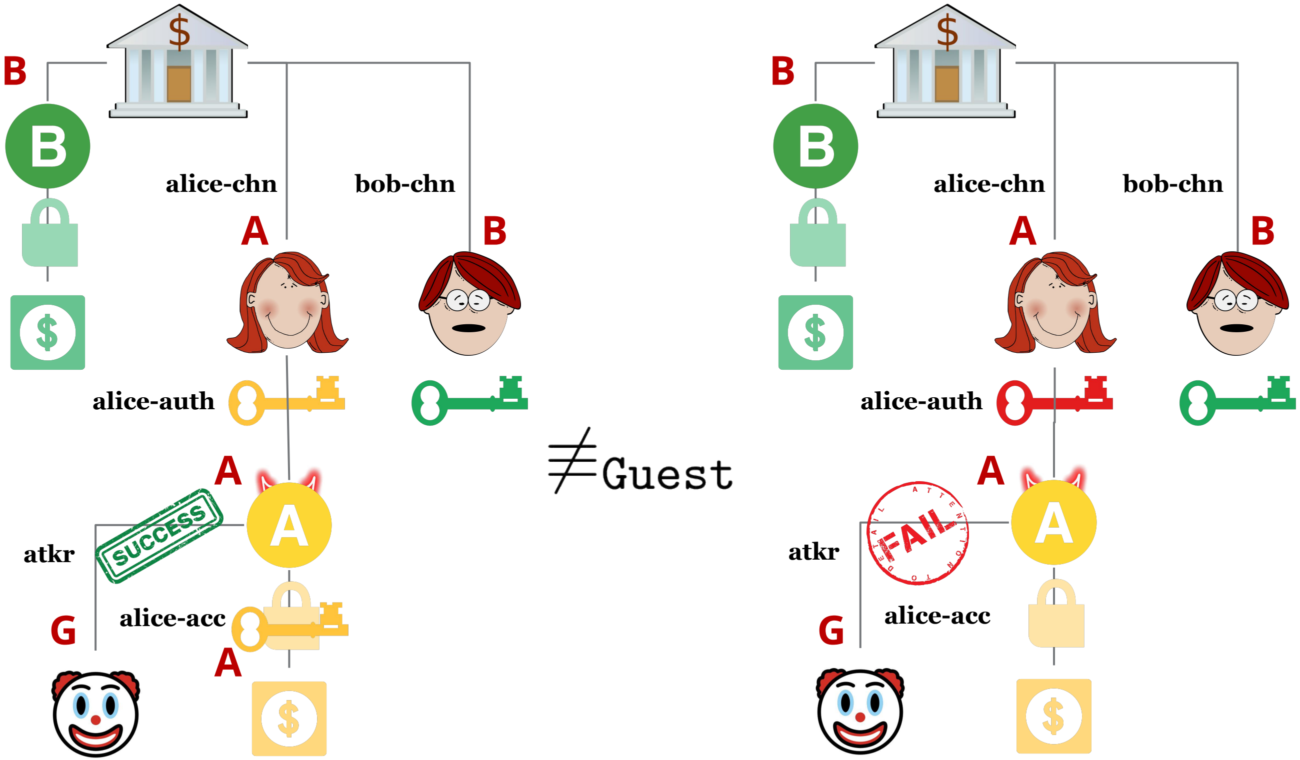


- ➔ ***Assume***: a process receives related messages along low-secrecy channels.
- ➔ ***Assert***: it sends the same messages along those channels.

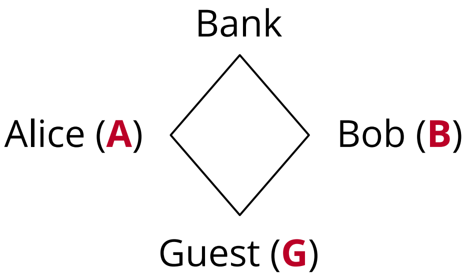
Maximal secrecy

- ★ 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



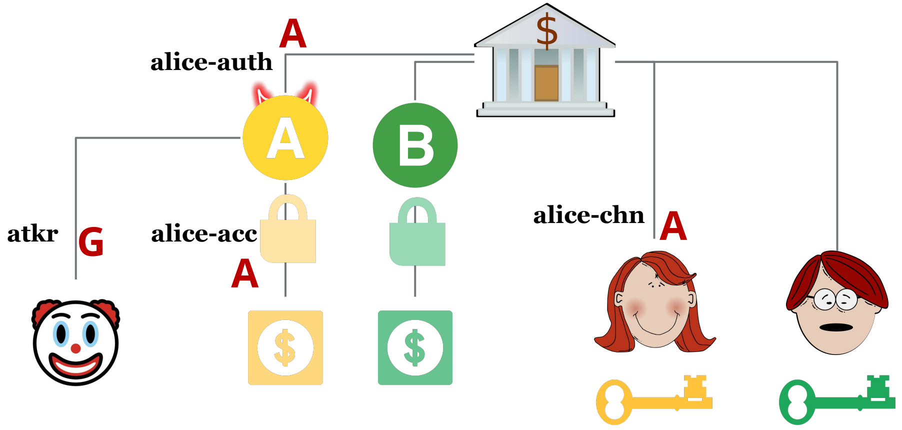
Security lattice $\Psi_{\mathtt{Bank}}$

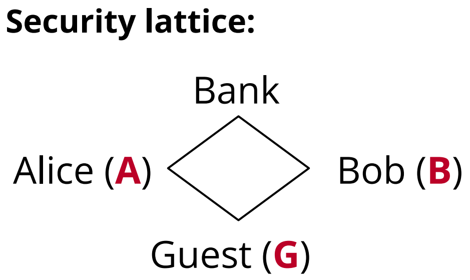


Running secrecy

Reflection of **the level of secret information a process has obtained** so far.

Running secrecy: the highest level of secret information obtained so far



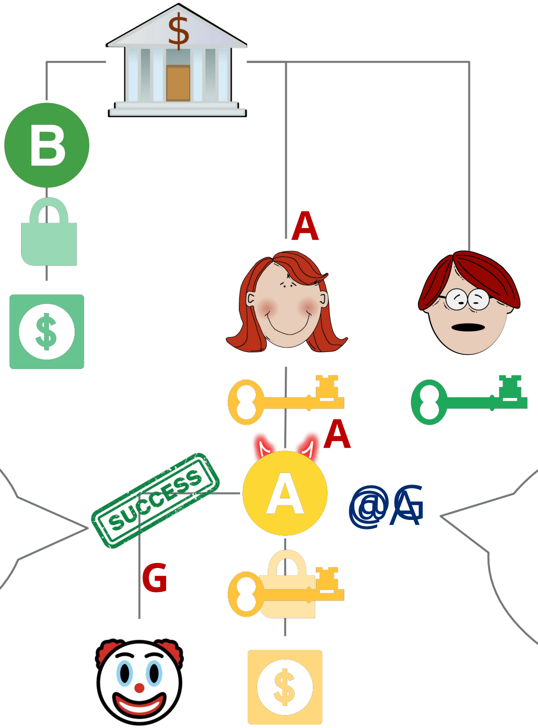


SneakyaAuth :

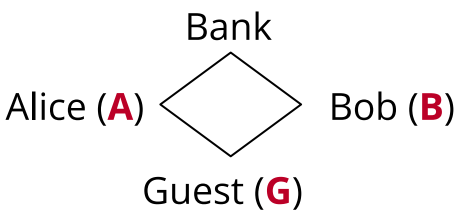
$$\mathbf{case}\, x \,(tok_{yellow} \Rightarrow x.succ; u.s; z.s; \\ \mid tok_{i \neq yellow} \Rightarrow x.fail; u.f; z.f;)$$

Running secrecy: the highest level of secret information obtained so far

Indirect flow 🚫



Security lattice:



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

Running secrecy of A is increased to Alice after receiving from x of secrecy Alice.

SneakyaAuth :

@Alice

$\textbf{case}\, x\, (tok_{yellow} \Rightarrow x.succ; u.s; z.s;$ // insecure send

error: Guest<Alice

@Guest

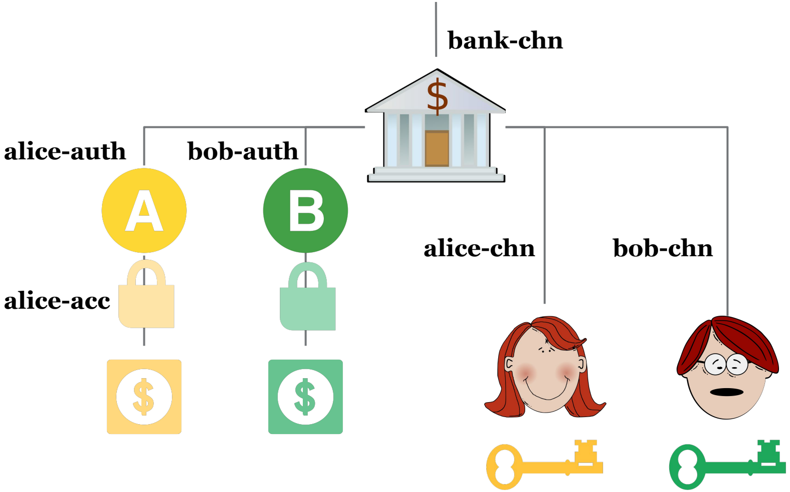
$\mid tok_{i \neq yellow} \Rightarrow x.fail; u.f; z.f;$ // insecure send$)$

Typing judgments with possible worlds

$$\Delta \vdash P \qquad :: (x{:}A)$$

Typing judgments with possible worlds

$$\Delta \vdash P \qquad :: (x{:}A)$$



$\texttt{alice-auth:auth}, \texttt{bob-auth:auth}, \texttt{alice-chn:customer}, \texttt{bob-chn:customer} \vdash \textbf{Bank} :: (\texttt{bank-chn}:1)$

Typing judgments with possible worlds

$$\Delta \vdash P \qquad :: (x{:}A)$$

Typing judgments with possible worlds

Security lattice

Maximal secrecy

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

Running secrecy

Typing judgments with possible worlds

Security lattice

Maximal secrecy

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

Running secrecy

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

Security lattice

Maximal secrecy

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

Running secrecy

Typing judgments with possible worlds

Security lattice

Maximal secrecy

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

Running secrecy

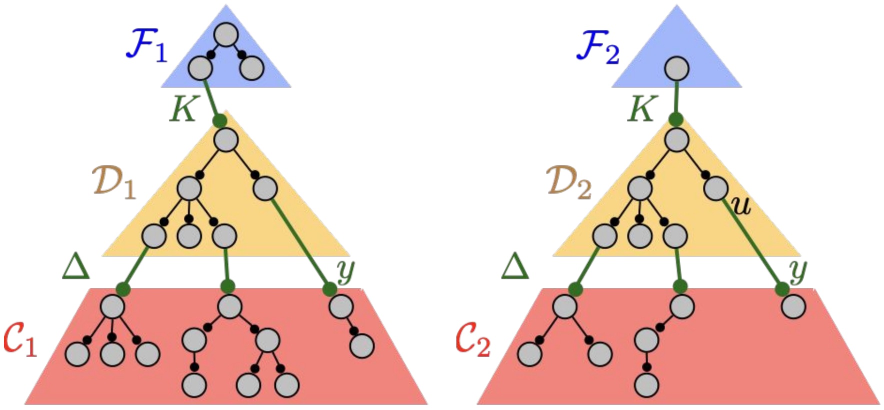
Tree invariant:

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



$$(\mathcal{C}_1\mathcal{D}_1\mathcal{F}_1, \mathcal{C}_2\mathcal{D}_2\mathcal{F}_2) \in \mathcal{E}^{\xi}_{\Psi}[\![\Delta \Vdash K]\!]$$

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*