Protocol analysis using ProVerif, 2nd part

ProVerif's input language

- ProVerif internally represents protocols as sets of Horn clauses.
- The protocol can be entered as Horn clauses, or as a process in a language similar to applied π-calculus.
- Invoking the analyzer:
  - `./analyzer` *file*, if *file* contains the protocol specification as Horn clauses;
  - `./analyzer -in pi` *file*, if *file* contains the protocol specification in applied π-calculus.

A process

A process P is one of

0	does nothing
$\texttt{new}\ n; P'$	create new atom n, then P'
$\texttt{in}(c, p); P'$	bind a msg from chan. c to var. p, then P'
$\texttt{out}(c, m); P'$	send the msg m on chan. c, then P'
$\texttt{let}\ p = M\ \texttt{in}\ P'\ \texttt{else}\ P''$	bind p to M, do P' if success, P'' otherwise
$P_1 \mid P_2$	do P_1 and P_2 in parallel
$!P'$	replicate P'. $!P' \equiv P' \mid !P'$
$\texttt{event}\ M; P'$	emit event ! M, then P'

A channel can be read (i.e. intercepted) and written by a party that knows its name.
A process represents all sessions of all parties.

Protocol specification

Declare

- message constructors;
  - constants, channel names, event names, constructors, etc.
  - whether adversary has access to them or not
- message destructors;
  - whether adversary has access to them or not
  - In the ProVerif language, terms cannot be “automatically” taken apart or parsed
    - like we did with Horn clauses
- predicates (if you need them);
- queries;
- the process.

Demo. . .

TODO:

- `proverif1.82/examples/pi/secr-auth/piyahalom`
  - Analysis of the code and execution result
- `proverif1.82/examples/pi/secr-auth/piyahalom-bid`

Useful trick: procedures / functions

Function implementation

```
private free f_in

let f =
  in(f_in, (f_out,arg));
    ......
  out(f_out, result).
```

Function call:

```
...
new f_out;
out(f_in, (f_out, arg));
in(f_out, result);
...
```

The Process contains:

```
process ...| !f | ...
```

Other properties: non-interference

- Let $P(\vec{x})$ be a process depending on variables $\vec{x}$.
- Informally, P does not preserve secrecy of $\vec{x}$, if
  - for some $\vec{M}$, $\vec{N}$
  - some attacker can observe the difference in behaviour of $P(\vec{M})$ and $P(\vec{N})$.
- e.g. $P(x, y) \equiv \texttt{new}\ k; \texttt{out}(c, (\{x\}_k, \{y\}_k))$ does not preserve the secrecy of (x, y).
- Indeed, the outputs made by $P(M, M)$ and $P(M, N)$ look different.
- Non-interference should be used if the set where the secrets come from is small.
- example: `proverif1.82/examples/pi/noninterf/piyahalom`

Global synchronization — phases

- ProVerif's process definition allows the construct

  $$\texttt{phase}\ n; P$$

  where n is an integer.
- P executes after the time point n has been reached. The commands preceeding `phase` n execute before that point.
- Some applications, e.g. voting, have such synchronization points.

Observational equivalence

- ProVerif's messages may contain the construct

$$\texttt{choice}[M_1, M_2]$$

- This defines two processes:
  - One, where all `choice`-constructs are replaced with their left arguments.
  - Another, where all `choice`-constructs are replaced with their right arguments.
- ProVerif tries to find whether some attacker can observe the difference in behaviour of these two processes.
- example: `proverif1.82/examples/pi/choice/pivote`
- A form of offline guessing attack