Protocol Composition Logic

A logic for proving security properties of network protocols

Stephan Stiller
Stanford Computer Security Lab
Post-presentation notes

• To address a question about confidential keys and honesty: Nonces are by default not confidential (in fact we currently treat nonce as a separate type from key). So sending out a nonce never violates honesty. The type confidential key is meant only for long-term keys (that are possibly shared between a small number of principals as part of the setup assumptions of a protocol), so the honesty condition on principals that they not send out confidential keys does not cause any problems. Keys that are intended to be sent out are not designated as confidential. (I have clarified our explanation of honesty in the slides below.)

• There is an upcoming book chapter (in a book edited by Steve Kremer and Véronique Cortier) with all the details.

• No proofs for axiom and rule schemas are shown in this presentation (but some can be found in the book chapter).

• Most tables shown in this presentation are excerpts only (full tables are in the book chapter).
About PCL

• “direct” logic (for proving things about network protocols) that does not involve explicit reasoning about attacker
• designed to reason about an unbounded number of protocol executions
• used for proving authentication and secrecy properties of network protocols
• PCL has a symbolic and a computational model, and we do not prove their equivalence.
• This talk is about a full formalization of PCL in LF with subtyping.
  – sketches how one can spell out PCL in detail
  – formalism is fully explicit, so that one could in principle feed it to a theorem prover or automated proof checker
  – talk is introduction; ask me for details
Terminology

- principal: a protocol participant
- role: sequence of actions prescribed by the protocol for well-behaved principals
- protocol: set of parameterized roles
  formally: $\text{Set}_{\text{fin}}(\text{List}(<\text{principal\_name}>) \rightarrow \text{role})$
- thread: participant paired up with a role
- action: sending, receiving, assignment, encryption, decryption, ...
- event: an action as it occurred in a particular thread:
  formally: $\text{thread\_id} \times \text{action}$
Handshake protocol in PCL

\textbf{Init}(A, B : principal\_name) = \{ \\
    k := \text{newnonce}; \\
    siga := \text{sign} \langle A, B, !k\rangle, sk(A); \\
    enca := \text{enc} !siga, pk(B); \\
    \text{send} !enca; \\

    encb := \text{receive}; \\
    s := \text{sd} !encb, !k; \\
\} \\

\textbf{Resp}(B : principal\_name) = \{ \\
    enca := \text{receive}; \\
    siga := \text{dec} !enca, dk(B); \\
    texta := \text{unsign} !siga; \\
    idA := π_1!texta; \\
    idB := π_2!texta; \\
    k := π_3!texta; \\
    \text{verify} !siga, vk(!idA); \\
    \text{assert}: !idB = B; \\
    s := \text{newsym}; \\
    encb := \text{se} !s, !k; \\
    \text{send} !encb; \\
\}
Explicit vs. abbreviated notation

**fully explicit version:**

\[
\text{Resp}(B : \text{principal}\_name) = \begin{cases} 
\text{enca} := \text{receive}; \\
\text{siga} := \text{dec} \text{!enca, } dk(B); \\
\text{texta} := \text{unsigned} \text{!siga}; \\
\text{idA} := \pi_1 \text{!texta}; \\
\text{idB} := \pi_2 \text{!texta}; \\
k := \pi_3 \text{!texta}; \\
\text{verify} \text{!siga, } vk(\text{!idA}); \\
\text{assert: } \text{!idB} = B; \\
\text{s} := \text{newsym}; \\
\text{encb} := \text{se} \text{!s, !k}; \\
\text{send} \text{!encb}; 
\end{cases}
\]

**abbreviated version (“PCL notation”):**

\[
\text{Resp}(B : \text{principal}\_name) = \begin{cases} 
\text{enca} := \text{receive}; \\
\text{siga} := \text{dec} \text{enca, } dk(B); \\
\text{texta} := \text{unsigned} \text{siga}; \\
\langle \text{idA, idB, k} \rangle := \text{texta}; \\
\text{verify} \text{siga, } vk(\text{idA}); \\
\text{assert: } \text{idB} = B; \\
\text{s} := \text{newsym}; \\
\text{encb} := \text{se} \text{s, } k; \\
\text{send} \text{encb}; 
\end{cases}
\]
What is “honesty”? Where is the attacker in PCL?

• **honesty:** When we consider any point during execution of a protocol in PCL, exactly those principals are considered “honest” that have executed a number of *basic sequences* of their assigned role.

• Thus: 2 ways to be dishonest:
  1. not executing the assigned role
  2. stopping in the middle of a basic sequence

• **attacker:** somebody who executes a thread but doesn’t follow the prescribed action sequence (item 1 above)
Matching conversations

• authentication as “matching conversations”:

• \( \text{Auth}_{\text{Resp}}: \)

  \[
  \text{true} \left[ \text{Resp}^T(B) \right] \land \text{Honest}(\text{id}_A^T) \land \text{id}_A^T \neq B
  \Rightarrow \exists T'. T'.\text{pname} = \text{id}_A^T
  \land \text{Send}(T', \text{enca}^T) \nless \text{Receive}(T, \text{enca}^T)
  \land \text{Receive}(T, \text{enca}^T) \nless \text{Send}(T, \text{encb}^T)
  \]

• Note: \( \varphi [\text{actseq}]_T \psi \) is a modal statement:
  – If \( \varphi \) is true and thread T executes exactly some action sequence actseq (and other threads may do other things meanwhile), then afterwards \( \psi \) will be true.
\[ \text{HON,AN3, } \text{Honest}(T_0) \land T_0.pname = X \land \text{Sign}(T_0, \langle X, Y, k_0 \rangle, sk(X)) \]

\[ \Rightarrow \text{FirstSend}(T_0, k_0, \left[ \left[ \langle X, Y, k_0 \rangle \right]_{sk(X)} \right]_{pk(Y)}) \]

(4)

\[ \text{G3 } \text{true [enca := receive; \cdots assert: idB = B;]_T Honest(idA}^{[T]} \text{)} \]

\[ \Rightarrow \exists T'. T'.pname = idA^{[T]} \]

\[ \land \text{FirstSend}(T', k^{[T]}, \left[ \left[ \langle idA^{[T]}, B, k^{[T]} \rangle \right]_{sk(idA^{[T]})} \right]_{pk(B)}) \]

(5)

\[ \text{AA0, AA1 } \text{true [enca := receive; \cdots assert: idB = B;]_T Receive(T, \left[ \left[ \langle idA^{[T]}, B, k^{[T]} \rangle \right]_{sk(idA^{[T]})} \right]_{pk(B)})} \]

\[ \land k^{[T]} \subseteq \left[ \left[ \langle idA^{[T]}, B, k^{[T]} \rangle \right]_{sk(idA^{[T]})} \right]_{pk(B)} \]

(6)

\[ \text{FS2 } \text{true [Resp(B)]_T Honest(idA}^{[T]} \text{)} \Rightarrow \exists T'. T'.pname = idA^{[T]} \land T' \neq T \]

\[ \Rightarrow \text{Send(T', enca}^{[T]} \text{)} \triangleleft \text{Receive(T, enca}^{[T]} \text{)} \]

(7)

\[ \text{AA4 } \text{true [Resp(B)]_T Honest(idA}^{[T]} \land idA}^{[T]} \neq B \Rightarrow \exists T'. T'.pname = idA^{[T]} \]

\[ \land \text{Send(T', enca}^{[T]} \text{)} \triangleleft \text{Receive(T, enca}^{[T]} \text{)} \]

\[ \land \text{Receive(T, enca}^{[T]} \text{)} \triangleleft \text{Send(T, encb}^{[T]} \text{)} \]

(8)

---

Formal proof of Auth\_resp
LF with subtyping

- **logical framework**: provides a uniform way of encoding a logical language, its inference rules, and its proofs

- “LF with subtyping”:
  - aka: *order-sorted logical framework* (OSLF)
  - typed language of expressions that includes tupling (pairing), functions (variable binding), (finite) lists, finite sets
  - subtyping (\(\sqsubseteq\)) that is similar to order-sorted algebra
    - every expression in PCL has a type, and we don’t need explicit type conversion functions

- **type**: \(\text{basic\_type} \mid \text{type} \times \ldots \times \text{type} \mid \text{type} \rightarrow \text{type} \mid \text{List} \langle \text{type} \rangle \mid \text{Set}\_\text{fin} \langle \text{type} \rangle\)

- **term**: \(\text{fct\_symb} : \text{type} \mid \text{variable} : \text{type} \mid \langle \text{term}_1 \ldots \text{term}_n \rangle \mid \pi_i \text{term} \mid \lambda \text{variable} : \text{type}_v . \text{term} \mid \text{term} \text{term}_1 \ldots \text{term}_n \ (n \in \{0, 1, 2, \ldots\}) \mid \ldots\)

- **subtyping**: \(\text{type}_1 \sqsubseteq \text{type}_2\)
  - *type retracts*: An expression of supertype type\(_2\) can be used as an expression of subtype type\(_1\), if the value so permits. This is really just implicit type casting.
PCL syntax: roles and actions

constant function symbol of type
List(principal_name) → role

Init(A, B : principal_name) = {
  k := newnonce;
  siga := sign (A, B, !k), sk(A);
  enca := enc !siga, pk(B);
  send !enca;

  encb := receive;
  s := sd !encb, !k;
}
PCL syntax: *locations* and some constant function symbols

constant function symbols of type `principal_name`

Init\(\langle A, B \rangle : \text{principal\_name}\) = {
    \[\begin{align*}
    k & := \text{newnonce}; \\
    \text{siga} & := \text{sign} \langle A, B, !k \rangle, \ sk(A); \\
    \text{enca} & := \text{enc} !\text{siga}, \ \text{pk}(B); \\
    \text{send} !\text{enca}; \\
    \text{encb} & := \text{receive}; \\
    s & := \text{sd} !\text{encb}, !k;
    \end{align*}\]
}

variables of type *location*:
- single-assignment
- thread-internal
- values managed by *store*, where *store* is of type:
  \[\text{thread\_id} \times \text{location} \to \text{message}\]

constant function symbols of types
- \((sk:\) \text{principal\_name} \to \text{conf\_sgn\_key})
- \((pk:\) \text{principal\_name} \to \text{asym\_enc\_key}\)
PCL syntax: actions

```
Init(A, B : principal_name) = {
  k := newnonce;
  siga := sign(A, B, !k), sk(A);
  enca := enc(!siga, pk(B));
  send(!enca);

  encb := receive;
  s := sd(!encb, !k);
}
```

read this as:

- newnonce(k)
- sign(siga, (A, B, !k), sk(A))
- enc(enca, !siga, pk(B))
- send(!enca)
- receive(encb)
- sd(s, !encb, !k)

type of green constant function symbol:

- location → action
- location × message × sgn_key → action
- location × message × asymmetric_enc_key → action
- message → action
- location → action
- location × message × nonce → action
## PCL syntax: original types

<table>
<thead>
<tr>
<th>Type name</th>
<th>Meta-variables (= variables)</th>
</tr>
</thead>
<tbody>
<tr>
<td>key</td>
<td>$K_1, K_2, ...$</td>
</tr>
<tr>
<td>principal_name</td>
<td>$\text{Pname}_1, \text{Pname}_2, ...$</td>
</tr>
<tr>
<td>role_name</td>
<td>$\text{rname}_1, \text{rname}_2, ...$</td>
</tr>
<tr>
<td>action</td>
<td>$\text{act}_1, \text{act}_2, ...$</td>
</tr>
<tr>
<td>thread_id</td>
<td>$\text{tid}_1, \text{tid}_2, ...$</td>
</tr>
<tr>
<td>location</td>
<td>$\text{loc}_1, \text{loc}_2, ...$</td>
</tr>
<tr>
<td>message</td>
<td>$\text{msg}_1, \text{msg}_2, ...$</td>
</tr>
<tr>
<td>action_formula</td>
<td>$\text{af}_1, \text{af}_2, ...$</td>
</tr>
<tr>
<td>nonmodal_formula</td>
<td>$\varphi_1, \varphi_2, ...; \psi_1, \psi_2, ...$</td>
</tr>
<tr>
<td>modal_formula</td>
<td>$\Phi_1, \Phi_2, ...; \Psi_1, \Psi_2, ...$</td>
</tr>
<tr>
<td>formula</td>
<td>$\text{fml}_1, \text{fml}_2, ...$</td>
</tr>
<tr>
<td>statement</td>
<td>$\text{stm}_1, \text{stm}_2, ...$</td>
</tr>
</tbody>
</table>
## PCL syntax: non-constructive type definitions

<table>
<thead>
<tr>
<th>Alias</th>
<th>Definition</th>
<th>Meta-variables</th>
</tr>
</thead>
<tbody>
<tr>
<td>thread</td>
<td>principal_name x role_name x List(principal_name) x thread_id</td>
<td>T₁, T₂, ...</td>
</tr>
<tr>
<td>principal</td>
<td>principal_name x List(key)</td>
<td>Princ₁, Princ₂, ...</td>
</tr>
<tr>
<td>actionseq</td>
<td>List(action)</td>
<td>actseq₁, actseq₂, ...</td>
</tr>
<tr>
<td>basicseq</td>
<td>List(action)</td>
<td>basicseq₁, basicseq₂, ...</td>
</tr>
<tr>
<td>role</td>
<td>role_name x List(basicseq)</td>
<td>role₁, role₂, ...</td>
</tr>
<tr>
<td>protocol</td>
<td>Set_fin(List(principal_name) → role)</td>
<td>P₁, P₂, ...</td>
</tr>
<tr>
<td>event</td>
<td>thread_id x action</td>
<td>ev₁, ev₂, ...</td>
</tr>
<tr>
<td>store</td>
<td>thread_id x location → message</td>
<td>st₁, st₂, ...</td>
</tr>
<tr>
<td>run</td>
<td>Set_fin(principal) x Set_fin(thread) x List(event) x store</td>
<td>R₁, R₂, ...</td>
</tr>
</tbody>
</table>
PCL syntax: subtype relations (1)

• `conf_sym_key ⊑ sym_key ⊑ key`
  ...
• `conf_sym_key, conf_asym_key, ..., conf_ver_key ⊑ conf_key`
• `conf_key ⊑ key`
• `principal_name, nonce, key, \( x \) message ⊑ message`
  \( \prod_{i=1}^{n} \)
• `action_formula ⊑ nonmodal_formula`
• `nonmodal_formula, modal_formula ⊑ formula`
PCL syntax: subtype relations (2)

• ⪯-relation: reflexive and transitive closure
• strong typing: all terms are implicitly type-annotated
• retracts:
  – key ⪯ message
  – key K may be sent out ("send K" action) because key ⪯ message.
  – But if we receive something, what we receive is of type message. However, since the received message is underlyingly of type key, it is okay to use it for actions that specifycally require type key.
PCL syntax: notational shortcuts

- $\text{Princ}.pname := \pi_1 \text{Princ}$, $\text{Princ}.klist := \pi_2 \text{Princ}$
- $\text{role}.rname := \pi_1 \text{role}$, $\text{role}.bseqs := \pi_2 \text{role}$
- $\text{ev}.tid := \pi_1 \text{ev}$, $\text{ev}.act := \pi_2 \text{ev}$
- $\mathcal{R}.pset := \pi_1 \mathcal{R}$, $\mathcal{R}.tset := \pi_2 \mathcal{R}$,
  $\mathcal{R}.elist := \pi_3 \mathcal{R}$, $\mathcal{R}.st := \pi_4 \mathcal{R}$
- $\text{pk}(\text{Pname}) = \text{Princ}.klist.1$, $\text{dk}(\text{Pname}) = \text{Princ}.klist.2$, ...
- ...

18
PCL syntax:
0-ary function symbols

<table>
<thead>
<tr>
<th>Function symbol</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>$k_1, k_2, \ldots$</td>
<td>key</td>
</tr>
<tr>
<td>Alice, Bob, Charlie, ...; A, B, C, ...</td>
<td>principal_name</td>
</tr>
<tr>
<td>Init, Resp, ...</td>
<td>role_name and List(principal_name) → role</td>
</tr>
<tr>
<td>1, 2, ...</td>
<td>thread_id</td>
</tr>
<tr>
<td>x, y, siga, encb, texta, idB, k, s, ...</td>
<td>location</td>
</tr>
<tr>
<td>‘‘, ‘a’, ‘ca’, ‘hello’, ...</td>
<td>message</td>
</tr>
<tr>
<td>true, false</td>
<td>nonmodal_formula</td>
</tr>
</tbody>
</table>
# PCL syntax:
non-constant function symbols (excerpt 1)

<table>
<thead>
<tr>
<th>Function symbol</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>!</td>
<td><code>location \to message</code></td>
</tr>
<tr>
<td>send</td>
<td><code>message \to action</code></td>
</tr>
<tr>
<td>receive</td>
<td><code>location \to action</code></td>
</tr>
<tr>
<td>newnonce</td>
<td><code>location \to action</code></td>
</tr>
<tr>
<td>enc</td>
<td><code>location \times message \times asym_enc_key \to action</code></td>
</tr>
<tr>
<td>Send</td>
<td><code>thread \times message \to action_formula</code></td>
</tr>
<tr>
<td>Receive</td>
<td><code>thread \times message \to action_formula</code></td>
</tr>
<tr>
<td>Enc</td>
<td><code>thread \times message \times key \to action_formula</code></td>
</tr>
<tr>
<td>$\triangleleft$</td>
<td><code>action_formula \times action_formula \to nonmodal_formula</code></td>
</tr>
<tr>
<td>$=$</td>
<td><code>message \times message \to nonmodal_formula</code></td>
</tr>
<tr>
<td>$=$</td>
<td><code>thread \times thread \to nonmodal_formula</code></td>
</tr>
<tr>
<td>Honest</td>
<td><code>principal_name \to nonmodal_formula</code></td>
</tr>
</tbody>
</table>
PCL syntax:
non-constant function symbols (excerpt 2)

<table>
<thead>
<tr>
<th>Function symbol</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>¬</td>
<td>nonmodal_formula → nonmodal_formula</td>
</tr>
<tr>
<td>∧</td>
<td>nonmodal_formula → nonmodal_formula</td>
</tr>
<tr>
<td>Modal</td>
<td>nonmodal_formula x actionseq x thread x nonmodal_formula → modal_formula</td>
</tr>
<tr>
<td>(shorthand:)</td>
<td></td>
</tr>
<tr>
<td>⊨</td>
<td>protocol x run x formula → statement</td>
</tr>
<tr>
<td>∀_type</td>
<td>(type x statement) → statement (for all types type)</td>
</tr>
</tbody>
</table>

- $\varphi [\text{actseq}]_T \psi$ is a `modal_formula`.
- Semantically, read it as follows:

$$\text{Modal } (\varphi, \text{actseq}, T, \psi)$$

nonmodal_formula nonmodal_formula thread nonmodal_formula

modal_formula
# PCL syntax vs. fully functional syntax

<table>
<thead>
<tr>
<th>PCL syntax</th>
<th>Syntax as it reflects the formal types</th>
</tr>
</thead>
<tbody>
<tr>
<td>{</td>
<td>msg</td>
</tr>
<tr>
<td>send msg</td>
<td>send(msg)</td>
</tr>
<tr>
<td>loc := receive</td>
<td>receive(loc)</td>
</tr>
<tr>
<td>loc := enc msg, K</td>
<td>enc(loc, msg, K)</td>
</tr>
<tr>
<td>verify msg_corrected, K</td>
<td>verify(msg_corrected, K)</td>
</tr>
<tr>
<td>af₁ &lt; af₂</td>
<td>≲(af₁, af₂)</td>
</tr>
<tr>
<td>φ [actseq]ᵣ ψ</td>
<td>Modal(φ, actseq, T, ψ)</td>
</tr>
<tr>
<td>P : R ⊨ fml</td>
<td>⊨(P, R, fml)</td>
</tr>
</tbody>
</table>
PCL syntax: equations

- \( \pi_i \langle v_1, v_2, \ldots, v_{i-1}, v_i, v_{i+1}, \ldots, v_k \rangle = v_i \)
  \[ k \in \mathbb{N}^+; \ i \in \{1, 2, \ldots, k\} \]

- \( \{| \{| \text{msg} \mid \}^a_K \mid \}^{-a}_K = \text{msg} \)
  \[ K: \text{key}; \ \text{msg}: \text{message} \]
What is a run / Feasibility of runs

• The Auth\textsubscript{Resp} property makes sense only in the context of a protocol and a run:
  \[ P:R \models true [\text{Resp}(B)]_T \text{ Honest}(idA[T]) \land idA[T] \neq B \Rightarrow \exists T'. T'.pname = idA[T] \land \text{Send}(T', \text{enca}[T]) \]
  \[ \prec \text{Receive}(T, \text{enca}[T]) \land \text{Receive}(T, \text{enca}[T]) \prec \text{Send}(T, \text{encb}[T]) \]

• If \( R_{\text{prev}} = \langle S_{\text{princ}}, S_{\text{th}}, \text{eventlist}, \text{store} \rangle \) is a feasible run with respect to protocol \( P \), then
  \[ R = \langle S_{\text{princ}}, S_{\text{th}}, \text{eventlist:}\langle\text{tid:act}\rangle, \text{store} \cup \text{newassgns} \rangle \]
  is a feasible run with respect to \( P \) where : indicates list concatenation, tid is a thread\_id among the threads in \( S_{\text{th}} \) and tid, act, and \( \text{newassgns} \) satisfy the following conditions:
  1. If act = loc := receive:
     – \( \text{newassgns} = \{\langle\langle\text{tid,loc},\text{msg}\rangle\} \)
     – condition:
       • msg was sent out previously by some thread.
  2. If act = send msg:
     ...
  Etc. (14 cases)
Truth conditions (honesty)

- **Auth\_Resp:**
  \[ P:R \models true [\text{Resp}(B)]_T \land idA[T] \neq B \Rightarrow \exists T'. T'.pname = idA[T] \land Send(T', enca[T]) \not\preceq Receive(T, enca[T]) \land Receive(T, enca[T]) \not\preceq Send(T, encb[T]) \]

- **P:R \models \text{Honest}(Pname):**
  - Pname is a *principal_name* of R
  - Each thread of this *principal* has executed precisely some number of basic sequences of its designated role. In particular, no thread can deviate from or stop in the middle of a basic sequence.
  - For the canonical forms of all messages that Pname sent out, it must be the case that any occurrences of keys of type *conf_key* may only occur in positions that require type *key* (or a subtype thereof). In other words, *conf_keys* are only ever used to encrypt or sign messages but not as the “payload”.
Truth conditions (modal formulas)

- $P:R \models \varphi \ [\text{actseq}]_T \psi$:
  - For all divisions $(R_1:R_2:R_3)$ of run $R$:
    - If $P:R_1 \models \varphi$ is true, and $(R_2.\text{elist}|_T \setminus R_1.\text{elist}|_T)$ matches actseq,
      then $P:R_2 \models \psi$ is true.
  - Btw, things such as
    - the notion of a division of a run,
    - the projection of an eventlist onto a thread, and
    - what it means for an eventlist to match an action sequence
      are defined in our upcoming book chapter.
Axiom and rule schemas

• Axiom schema FS2:

\[(\text{NewNonce}(T_1, \text{msg}_1) \land \text{FirstSend}(T_1, \text{msg}_1, \text{msg}_2))\]
\[\land (T_1 \neq T_2 \land \text{msg}_1 \subseteq \text{msg}_3 \land \text{Receive}(T_2, \text{msg}_3))\]
\[\Rightarrow \text{Send}(T_1, \text{msg}_2) \ll \text{Receive}(T_2, \text{msg}_3)\]

• Rule schema SEQ:

\[\varphi_1 [\text{actseq}]_T \varphi_2 \quad \varphi_2 [\text{actseq'}]_T \varphi_3\]
\[\varphi_1 [\text{actseq} : \text{actseq'}]_T \varphi_3\]
HON.AN3. \quad \text{Honest}(T_0) \land T_0.pname = X \land \text{Sign}(T_0, \langle X, Y, k_0 \rangle, sk(X))

P2,FS1 \quad \Rightarrow \text{FirstSend} \left( T_0, k_0, \{[\langle X, Y, k_0 \rangle]_{sk(X)}\}_{pk(Y)}^a \right)

\hspace{1cm} (4)

G3 \quad \text{true [enca := receive; \cdots assert: idB = B;} \rangle T \\text{Honest(idA}^T]\)

\quad \Rightarrow \exists T'. T'. pname = idA^T

\quad \land \text{FirstSend} \left( T', k^T, \{[\langle idA^T, B, k^T \rangle]_{sk(idA^T)}\}_{pk(B)}^a \right)

\hspace{1cm} (5)

AA0, AA1 \quad \text{true [enca := receive; \cdots assert: idB = B;} \rangle T \\text{Receive} \left( T, \{[\langle idA^T, B, k^T \rangle]_{sk(idA^T)}\}_{pk(B)}^a \right)

\quad \land k^T \subseteq \{[\langle idA^T, B, k^T \rangle]_{sk(idA^T)}\}_{pk(B)}

\hspace{1cm} (6)

FS2 \quad \text{true [Resp(B)} \rangle T \\text{Honest(idA}^T]\) \Rightarrow \exists T'. T'. pname = idA^T \land T' \neq T

\quad \Rightarrow \text{Send}(T', \text{enca}^T) \triangleq \text{Receive}(T, \text{enca}^T)

\hspace{1cm} (7)

AA4 \quad \text{true [Resp(B)} \rangle T \\text{Honest(idA}^T]\) \land idA^T \neq B \Rightarrow \exists T'. T'. pname = idA^T

\quad \land \text{Send}(T', \text{enca}^T) \triangleq \text{Receive}(T, \text{enca}^T)

\quad \land \text{Receive}(T, \text{enca}^T) \triangleq \text{Send}(T, \text{encb}^T)

\hspace{1cm} (8)

Formal proof of Auth_{resp}
Thanks

• Joint work with:
  – John C. Mitchell (Stanford)
  – Arnab Roy (now at IBM)
  – Anupam Datta (CMU)

• numerous others before