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: $\mathrm{Set}_{\mathrm{fin}}(\mathrm{List}(\mathit{principal\_name}) \rightarrow \mathit{role})$
- thread: participant paired up with a role
- action: sending, receiving, assignment, encryption, decryption, ...
- event: an action as it occurred in a particular thread:

  formally: $\mathit{thread\_id} \times \mathit{action}$

Handshake protocol in PCL

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;
}

Resp(B : *principal_name*) = {
  enca := **receive**;
  siga := **dec** !enca, *dk*(B);
  texta := **unsign** !siga;
  idA := π_1!texta;
  idB := π_2!texta;
  k := π_3!texta;
  verify !siga, *vk*(!idA);
  assert: !idB = B;
  s := **newsym**;
  encb := **se** !s, !k;
  send !encb;
}

Explicit vs. abbreviated notation

fully explicit version:

Resp(B : *principal_name*) = {
enca := **receive**;
siga := **dec** !enca, *dk*(B);
texta := **unsign** !siga;
idA := π_1!texta;
idB := π_2!texta;
k := π_3!texta;
verify !siga, *vk*(!idA);
assert: !idB = B;
s := **newsym**;
encb := **se** !s, !k;
send !encb;
}

abbreviated version ("PCL notation"):

Resp(B : *principal_name*) = {
enca := **receive**;
siga := **dec** enca, *dk*(B);
texta := **unsign** siga;
⟨idA, idB, k⟩ := texta;
verify siga, *vk*(idA);
assert: idB = B;
s := **newsym**;
encb := **se** s, k;
send encb;
}

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":
- $\text{Auth}_{\text{Resp}}$:

  $\mathit{true}\ [\textbf{Resp}(B)]_T\ \text{Honest}(idA^{[T]}) \wedge idA^{[T]} \neq B$

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

  $\wedge\ \text{Send}(T', enca^{[T]}) \lhd \text{Receive}(T, enca^{[T]})$

  $\wedge\ \text{Receive}(T, enca^{[T]}) \lhd \text{Send}(T, encb^{[T]})$

- Note: $\varphi\ [\text{actseq}]_T\ \psi$ is a *modal statement*:
  - If φ is true and thread T executes exactly some action sequence actseq (and other threads may do other things meanwhile), then afterwards ψ will be true.

$$
\begin{array}{rl}
\textbf{HON,AN3,} & \mathsf{Honest}(\mathrm{T}_0) \wedge \mathrm{T}_0\,.\mathit{pname} = \mathrm{X} \wedge \mathsf{Sign}(\mathrm{T}_0, \langle \mathrm{X}, \mathrm{Y}, \mathrm{k}_0 \rangle, sk\,(\mathrm{X})) \\
\textbf{P2,FS1} & \quad \Rightarrow \mathsf{FirstSend}\left(\mathrm{T}_0, \mathrm{k}_0, \left\{\!\left| [\![\langle \mathrm{X}, \mathrm{Y}, \mathrm{k}_0 \rangle]\!]_{sk(\mathrm{X})} \right|\!\right\}^{\mathsf{a}}_{pk(\mathrm{Y})}\right) \qquad (4)
\end{array}
$$

$$
\begin{array}{rl}
\textbf{G3} & \mathit{true}\ [\mathsf{enca} := \textbf{receive}; \cdots\ \textbf{assert:}\ \mathtt{idB} = \mathrm{B};\,]_{\mathrm{T}}\ \mathsf{Honest}(\mathtt{idA}^{[\mathrm{T}]}) \\
& \quad \Rightarrow \exists \mathrm{T}'.\ \mathrm{T}'\,.\mathit{pname} = \mathtt{idA}^{[\mathrm{T}]} \\
& \quad \wedge\ \mathsf{FirstSend}\left(\mathrm{T}', \mathrm{k}^{[\mathrm{T}]}, \left\{\!\left| [\![\left\langle \mathtt{idA}^{[\mathrm{T}]}, \mathrm{B}, \mathrm{k}^{[\mathrm{T}]} \right\rangle]\!]_{sk(\mathtt{idA}^{[\mathrm{T}]})} \right|\!\right\}^{\mathsf{a}}_{pk(\mathrm{B})}\right) \qquad (5) \\
\textbf{AA0, AA1} & \mathit{true}\ [\mathsf{enca} := \textbf{receive};\ \cdots\ \textbf{assert:}\ \mathtt{idB} = \mathrm{B};\,]_{\mathrm{T}}\ \mathsf{Receive}\left(\mathrm{T}, \left\{\!\left| [\![\left\langle \mathtt{idA}^{[\mathrm{T}]}, \mathrm{B}, \mathrm{k}^{[\mathrm{T}]} \right\rangle]\!]_{sk(\mathtt{idA}^{[\mathrm{T}]})} \right|\!\right\}^{\mathsf{a}}_{pk(\mathrm{B})}\right) \\
& \quad \wedge\ \mathrm{k}^{[\mathrm{T}]} \Subset \left\{\!\left| [\![\left\langle \mathtt{idA}^{[\mathrm{T}]}, \mathrm{B}, \mathrm{k}^{[\mathrm{T}]} \right\rangle]\!]_{sk(\mathtt{idA}^{[\mathrm{T}]})} \right|\!\right\}^{\mathsf{a}}_{pk(\mathrm{B})} \qquad (6) \\
\textbf{FS2} & \mathit{true}\ [\textbf{Resp}(\mathrm{B})]_{\mathrm{T}}\ \mathsf{Honest}(\mathtt{idA}^{[\mathrm{T}]}) \Rightarrow \exists \mathrm{T}'.\ \mathrm{T}'\,.\mathit{pname} = \mathtt{idA}^{[\mathrm{T}]} \wedge \mathrm{T}' \neq \mathrm{T} \\
& \quad \Rightarrow \mathsf{Send}(\mathrm{T}', \mathsf{enca}^{[\mathrm{T}]}) \lhd \mathsf{Receive}(\mathrm{T}, \mathsf{enca}^{[\mathrm{T}]}) \qquad (7) \\
\textbf{AA4} & \mathit{true}\ [\textbf{Resp}(\mathrm{B})]_{\mathrm{T}}\ \mathsf{Honest}(\mathtt{idA}^{[\mathrm{T}]}) \wedge \mathtt{idA}^{[\mathrm{T}]} \neq \mathrm{B} \Rightarrow \exists \mathrm{T}'.\ \mathrm{T}'\,.\mathit{pname} = \mathtt{idA}^{[\mathrm{T}]} \\
& \quad \wedge\ \mathsf{Send}(\mathrm{T}', \mathsf{enca}^{[\mathrm{T}]}) \lhd \mathsf{Receive}(\mathrm{T}, \mathsf{enca}^{[\mathrm{T}]}) \\
& \quad \wedge\ \mathsf{Receive}(\mathrm{T}, \mathsf{enca}^{[\mathrm{T}]}) \lhd \mathsf{Send}(\mathrm{T}, \mathsf{encb}^{[\mathrm{T}]}) \qquad (8)
\end{array}
$$

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* ::= *basic_type* | *type* x ... x *type* | *type* → *type* | List(*type*)

  | $\mathrm{Set}_{\mathrm{fin}}$(*type*)

  term ::= *fct_symb* : *type* | *variable* : *type*

  | ⟨$term_1$ ... $term_n$⟩ | π_i *term*

  | λ *variable* : $type_v$. *term* | *term* $term_1$ ... $term_n$ ($n \in \{0, 1, 2, ...\}$)

  | ...
- *subtyping:* $type_1 \sqsubseteq \mathrm{type}_2$
  - *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.

constant function symbol of type
List(*principal_name*) → *role*

PCL syntax: *roles* and *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;

}

each line is of type *action*

basic sequences

role

PCL syntax: *locations* and some constant function symbols

constant function symbols of type *principal_name*

Init($\boxed{A, B}$: *principal_name*) = {
  k := **newnonce**;
  siga := **sign** ⟨A, B, !k⟩, *sk*(A);
  enca := **enc** !siga, *pk*(B);
  send !enca;

  encb := **receive**;
  s := **sd** !encb, !k;
}

variables of type *location*:

- single-assignment
- thread-internal
- values managed by *store*, where store is of type:

  thread_id x *location* → *message*

constant function symbols of types
(*sk*:) *principal_name* → *conf_sgn_key*
(*pk*:) *principal_name* → *asym_enc_key*

PCL syntax: *actions*

Init(A, B : *principal_name*) = {	*read this as:*	*type of green constant function symbol:*
k := **newnonce**;	➔ **newnonce**(k)	*location* → *action*
siga := **sign** ⟨A, B, !k⟩, *sk*(A);	➔ **sign**(siga, ⟨A, B, !k⟩, *sk*(A))	*location* x *message* x *sgn_key* → *action*
enca := **enc** !siga, *pk*(B);	➔ **enc** (enca, !siga, *pk*(B))	*location* x *message* x *asym_enc_key* → *action*
send !enca;	➔ **send**(!enca)	*message* → *action*
encb := **receive**;	➔ **receive**(encb)	*location* → *action*
s := **sd** !encb, !k;	➔ **sd**(s, !encb, !k)	*location* x *message* x *nonce* → *action*
}		

PCL syntax: original types

Type name	Meta-variables (= variables)
key	K_1, K_2, ...
principal_name	Pname_1, Pname_2, ...
role_name	rname_1, rname_2, ...
action	act_1, act_2, ...
thread_id	tid_1, tid_2, ...
location	loc_1, loc_2, ...
message	msg_1, msg_2, ...
action_formula	af_1, af_2, ...
nonmodal_formula	φ_1, φ_2, ...; ψ_1, ψ_2, ...
modal_formula	Φ_1, Φ_2, ...; Ψ_1, Ψ_2, ...
formula	fml_1, fml_2, ...
statement	stm_1, stm_2, ...

PCL syntax: non-constructive type definitions

Alias	Definition	Meta-variables
thread	*principal_name* x *role_name* x List(principal_name) x *thread_id*	T_1, T_2, ...
principal	*principal_name* x List(*key*)	$Princ_1$, $Princ_2$, ...
actionseq	List(*action*)	$actseq_1$, $actseq_2$, ...
basicseq	List(*action*)	$basicseq_1$, $basicseq_2$, ...
role	*role_name* x List(*basicseq*)	$role_1$, $role_2$, ...
protocol	Set_{fin}(List(*principal_name*) → *role*)	$\mathcal{P}_1$, $\mathcal{P}_2$, ...
event	*thread_id* x *action*	ev_1, ev_2, ...
store	*thread_id* x *location* → *message*	st_1, st_2, ...
run	Set_{fin}(*principal*) x Set_{fin}(*thread*) x List(*event*) x *store*	$\mathcal{R}_1$, $\mathcal{R}_2$, ...

PCL syntax: subtype relations (1)

- *conf_sym_key* $\sqsubseteq$ *sym_key* $\sqsubseteq$ *key*

  ...
- *conf_sym_key*, *conf_asym_key*, ..., *conf_ver_key* $\sqsubseteq$ *conf_key*
- *conf_key* $\sqsubseteq$ *key*
- *principal_name*, *nonce*, *key*, $\times_{i=1}^{n}$ *message* $\sqsubseteq$ *message*
- *action_formula* $\sqsubseteq$ *nonmodal_formula*
- *nonmodal_formula*, *modal_formula* $\sqsubseteq$ *formula*

PCL syntax: subtype relations (2)

- $\sqsubseteq$-relation: reflexive and transitive closure
- strong typing: all terms are implicitly type-annotated
- retracts:
  - *key* $\sqsubseteq$ *message*
  - *key* K may be sent out ("**send** K" action) because *key* $\sqsubseteq$ *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 specifically require type *key*.

PCL syntax: notational shortcuts

- Princ.*pname* := $\boldsymbol{\pi}_{\mathbf{1}}$Princ, Princ.*klist* := $\boldsymbol{\pi}_{\mathbf{2}}$Princ
- role.*rname* := $\boldsymbol{\pi}_{\mathbf{1}}$role, role.*bseqs* := $\boldsymbol{\pi}_{\mathbf{2}}$role
- ev.*tid* := $\boldsymbol{\pi}_{\mathbf{1}}$ev, ev.*act* := $\boldsymbol{\pi}_{\mathbf{2}}$ev
- $\mathscr{R}$.*pset* := $\boldsymbol{\pi}_{\mathbf{1}}\mathscr{R}$, $\mathscr{R}$.*tset* := $\boldsymbol{\pi}_{\mathbf{2}}\mathscr{R}$,
  $\mathscr{R}$.*elist* := $\boldsymbol{\pi}_{\mathbf{3}}\mathscr{R}$, $\mathscr{R}$.*st* := $\boldsymbol{\pi}_{\mathbf{4}}\mathscr{R}$
- *pk*(Pname) = Princ.*klist.1*, *dk*(Pname) = Princ.*klist.2*, ...
- ...

PCL syntax: 0-ary function symbols

Function symbol	Type
k_1, k_2, ...	*key*
Alice, Bob, Charlie, ...; A, B, C, ...	*principal_name*
Init, **Resp**, ...	*role_name* <u>and</u> List(*principal_name*) → *role*
1, 2, ...	*thread_id*
x, y, siga, encb, texta, idB, k, s, ...	*location*
‘’, ‘a’, ‘ca’, ‘hello’, ...	*message*
true, *false*	*nonmodal_formula*

PCL syntax: non-constant function symbols (excerpt 1)

Function symbol	Type
!	*location* → *message*
send	*message* → *action*
receive	*location* → *action*
newnonce	*location* → *action*
enc	*location* x *message* x *asym_enc_key* → *action*
Send	*thread* x *message* → *action_formula*
Receive	*thread* x *message* → *action_formula*
Enc	*thread* x *message* x *key* → *action_formula*
⊲	*action_formula* x *action_formula* → *nonmodal_formula*
=	*message* x *message* → *nonmodal_formula*
=	*thread* x *thread* → *nonmodal_formula*
Honest	*principal_name* → *nonmodal_formula*

PCL syntax: non-constant function symbols (excerpt 2)

Function symbol	Type
¬	*nonmodal_formula* → *nonmodal_formula*
∧	*nonmodal_formula* → *nonmodal_formula*
Modal (*shorthand:*) $\cdot[\cdot]_{\cdot}\cdot$	*nonmodal_formula* x *actionseq* x *thread* x *nonmodal_formula* → *modal_formula*
⊨	*protocol* x *run* x *formula* → *statement*
$\forall_{type}$	(*type* x *statement*) → *statement* (for all types *type*)

- φ [actseq]$_T$ ψ is a *modal_formula*.
- Semantically, read it as follows:

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

modal_formula

nonmodal_formula

thread

nonmodal_formula

actionseq

PCL syntax vs. fully functional syntax

PCL syntax	Syntax as it reflects the formal types
$\{\| \text{msg} \|\}^{a}_{K}$	*encmsg*(msg, K)
send msg	**send**(msg)
loc := **receive**	**receive**(loc)
loc := **enc** msg, K	**enc**(loc, msg, K)
verify $\text{msg}_{\text{signed}}$, K	**verify**($\text{msg}_{\text{signed}}$, K)
$\text{af}_1 \lhd \text{af}_2$	$\lhd(\text{af}_1, \text{af}_2)$
$\phi\ [\text{actseq}]_T\ \psi$	*Modal*(ϕ, actseq, T, ψ)
$\mathcal{P} : \mathcal{R} \vDash \text{fml}$	$\vDash(\mathcal{P}, \mathcal{R}, \text{fml})$

PCL syntax: equations

- $\boldsymbol{\pi}_{\mathbf{i}}\langle v_1, v_2, \ldots, v_{i-1}, v_i, v_{i+1}, \ldots, v_k\rangle = v_i$

  $[k \in \mathbb{N}^+; i \in \{1, 2, \ldots, k\}]$

- $\{\!|\ \{\!|\ msg\ |\!\}^a_K\ |\!\}^{-a}_K = msg$

  [K: *key*; msg: *message*]

What is a run / Feasibility of runs

- The $\text{Auth}_{\text{Resp}}$ property makes sense only in the context of a *protocol* and a *run*:

  $\mathcal{P}:\mathcal{R} \vDash \textit{true}\ [\textbf{Resp}(B)]_T\ \text{Honest}(idA^{[T]}) \wedge idA^{[T]} \neq B \Rightarrow \exists T' .\ T'.\textit{pname} = idA^{[T]} \wedge \text{Send}(T', enca^{[T]})$
  $\lhd \text{Receive}(T, enca^{[T]}) \wedge \text{Receive}(T, enca^{[T]}) \lhd \text{Send}(T, encb^{[T]})$

- If $\mathcal{R}_{prev} = \langle S_{princ}, S_{th}, \textit{eventlist}, \textit{store}\rangle$ is a feasible *run* with respect to *protocol* $\mathcal{P}$, then

  $\mathcal{R} = \langle S_{princ}, S_{th}, \textit{eventlist}:\langle tid:act\rangle, \textit{store} \cup \textit{newassgns}\rangle$

  is a feasible *run* with respect to $\mathcal{P}$ where : indicates list concatenation, tid is a *thread_id* among the *threads* in S_{th} and tid, act, and *newassgns* satisfy the following conditions:

  1. If act = loc := **receive**:
     - *newassgns* = {⟨⟨tid,loc⟩,msg⟩}
     - condition:
       - msg was sent out previously by some thread.
  2. If act = **send** msg:

     ...

  Etc. (14 cases)

Truth conditions (honesty)

- $\text{Auth}_{\text{Resp}}$:

  $\mathcal{P}:\mathcal{R} \vDash \textit{true}\ [\mathbf{Resp}(B)]_T\ \text{Honest}(idA^{[T]}) \wedge idA^{[T]} \neq B \Rightarrow \exists T' .\ T'.\textit{pname} = idA^{[T]} \wedge \text{Send}(T', enca^{[T]}) \lhd \text{Receive}(T, enca^{[T]}) \wedge \text{Receive}(T, enca^{[T]}) \lhd \text{Send}(T, encb^{[T]})$

- $\mathcal{P}:\mathcal{R} \vDash \text{Honest}(\text{Pname})$:
  - Pname is a *principal_name* of $\mathcal{R}$
  - 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_key*s are only ever used to encrypt or sign messages but not as the "payload".

Truth conditions (modal formulas)

- $\mathcal{P}:\mathcal{R} \vDash \varphi\ [\text{actseq}]_T\ \psi$:
  - For all divisions ($\mathcal{R}_1:\mathcal{R}_2:\mathcal{R}_3$) of *run* $\mathcal{R}$:

    If $\mathcal{P}:\mathcal{R}_1 \vDash \varphi$ is true, and ($\mathcal{R}_2.\mathit{elist}|_T \setminus \mathcal{R}_1.\mathit{elist}|_T$) matches actseq,

    then $\mathcal{P}:\mathcal{R}_2 \vDash \psi$ 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:

$$((\mathrm{NewNonce}(T_1, msg_1) \wedge \mathrm{FirstSend}(T_1, msg_1, msg_2)) \\ \wedge (T_1 \neq T_2 \wedge msg_1 \Subset msg_3 \wedge \mathrm{Receive}(T_2, msg_3)) \\ \Rightarrow \mathrm{Send}(T_1, msg_2) \lhd \mathrm{Receive}(T_2, msg_3)$$

- Rule schema SEQ:

$$\frac{\varphi_1\ [actseq]_T\ \varphi_2 \quad \varphi_2\ [actseq']_T\ \varphi_3}{\varphi_1\ [actseq : actseq']_T\ \varphi_3}$$

$$\textbf{HON,AN3,} \quad \mathsf{Honest}(\mathrm{T}_0) \wedge \mathrm{T}_0.\mathit{pname} = \mathrm{X} \wedge \mathsf{Sign}(\mathrm{T}_0, \langle \mathrm{X}, \mathrm{Y}, \mathrm{k}_0 \rangle, \mathit{sk}(\mathrm{X}))$$

$$\textbf{P2,FS1} \quad \Rightarrow \mathsf{FirstSend}\left(\mathrm{T}_0, \mathrm{k}_0, \left\{\!\left| [\![\langle \mathrm{X}, \mathrm{Y}, \mathrm{k}_0 \rangle]\!]_{\mathit{sk}(\mathrm{X})} \right|\!\right\}^{\mathsf{a}}_{\mathit{pk}(\mathrm{Y})}\right) \tag{4}$$

$$\textbf{G3} \quad \mathit{true}\ [\mathsf{enca} := \textbf{receive}; \cdots\ \textbf{assert:}\ \mathtt{idB} = \mathrm{B};\ ]_{\mathrm{T}}\ \mathsf{Honest}(\mathtt{idA}^{[\mathrm{T}]})$$

$$\Rightarrow \exists \mathrm{T}'.\ \mathrm{T}'.\mathit{pname} = \mathtt{idA}^{[\mathrm{T}]}$$

$$\wedge\ \mathsf{FirstSend}\left(\mathrm{T}', \mathrm{k}^{[\mathrm{T}]}, \left\{\!\left| [\![\left\langle \mathtt{idA}^{[\mathrm{T}]}, \mathrm{B}, \mathrm{k}^{[\mathrm{T}]} \right\rangle]\!]_{\mathit{sk}(\mathtt{idA}^{[\mathrm{T}]})} \right|\!\right\}^{\mathsf{a}}_{\mathit{pk}(\mathrm{B})}\right) \tag{5}$$

$$\textbf{AA0, AA1} \quad \mathit{true}\ [\mathsf{enca} := \textbf{receive};\ \cdots\ \textbf{assert:}\ \mathtt{idB} = \mathrm{B};\ ]_{\mathrm{T}}\ \mathsf{Receive}\left(\mathrm{T}, \left\{\!\left| [\![\left\langle \mathtt{idA}^{[\mathrm{T}]}, \mathrm{B}, \mathrm{k}^{[\mathrm{T}]} \right\rangle]\!]_{\mathit{sk}(\mathtt{idA}^{[\mathrm{T}]})} \right|\!\right\}^{\mathsf{a}}_{\mathit{pk}(\mathrm{B})}\right)$$

$$\wedge\ \mathrm{k}^{[\mathrm{T}]} \Subset \left\{\!\left| [\![\left\langle \mathtt{idA}^{[\mathrm{T}]}, \mathrm{B}, \mathrm{k}^{[\mathrm{T}]} \right\rangle]\!]_{\mathit{sk}(\mathtt{idA}^{[\mathrm{T}]})} \right|\!\right\}^{\mathsf{a}}_{\mathit{pk}(\mathrm{B})} \tag{6}$$

$$\textbf{FS2} \quad \mathit{true}\ [\textbf{Resp}(\mathrm{B})]_{\mathrm{T}}\ \mathsf{Honest}(\mathtt{idA}^{[\mathrm{T}]}) \Rightarrow \exists \mathrm{T}'.\ \mathrm{T}'.\mathit{pname} = \mathtt{idA}^{[\mathrm{T}]} \wedge \mathrm{T}' \neq \mathrm{T}$$

$$\Rightarrow \mathsf{Send}(\mathrm{T}', \mathsf{enca}^{[\mathrm{T}]}) \lhd \mathsf{Receive}(\mathrm{T}, \mathsf{enca}^{[\mathrm{T}]}) \tag{7}$$

$$\textbf{AA4} \quad \mathit{true}\ [\textbf{Resp}(\mathrm{B})]_{\mathrm{T}}\ \mathsf{Honest}(\mathtt{idA}^{[\mathrm{T}]}) \wedge \mathtt{idA}^{[\mathrm{T}]} \neq \mathrm{B} \Rightarrow \exists \mathrm{T}'.\ \mathrm{T}'.\mathit{pname} = \mathtt{idA}^{[\mathrm{T}]}$$

$$\wedge\ \mathsf{Send}(\mathrm{T}', \mathsf{enca}^{[\mathrm{T}]}) \lhd \mathsf{Receive}(\mathrm{T}, \mathsf{enca}^{[\mathrm{T}]})$$

$$\wedge\ \mathsf{Receive}(\mathrm{T}, \mathsf{enca}^{[\mathrm{T}]}) \lhd \mathsf{Send}(\mathrm{T}, \mathsf{encb}^{[\mathrm{T}]}) \tag{8}$$

Formal proof of $Auth_{resp}$

Thanks

- Joint work with:
  - John C. Mitchell (Stanford)
  - Arnab Roy (now at IBM)
  - Anupam Datta (CMU)
- numerous others before