

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: $\text{Set}_{\text{fin}}(\text{List}(\textit{principal_name}) \rightarrow \textit{role})$
- thread: participant paired up with a role
- action: sending, receiving, assignment, encryption, decryption, ...
- event: an action as it occurred in a particular thread:
formally: $\textit{thread_id} \times \textit{action}$

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 :=  $\pi_1$ !texta;  
  idB :=  $\pi_2$ !texta;  
  k :=  $\pi_3$ !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 :=  $\pi_1$ !texta;  
  idB :=  $\pi_2$ !texta;  
  k :=  $\pi_3$ !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”:

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

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

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

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

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

- 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.

HON,AN3, $\text{Honest}(T_0) \wedge T_0.pname = X \wedge \text{Sign}(T_0, \langle X, Y, k_0 \rangle, sk(X))$

$$\mathbf{P2,FS1} \quad \Rightarrow \text{FirstSend}\left(T_0, k_0, \left\{ \left\{ \langle X, Y, k_0 \rangle \right\}_{sk(X)} \right\}_{pk(Y)}^a\right) \quad (4)$$

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

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

$$\wedge \text{FirstSend}\left(T', k^{[T]}, \left\{ \left\{ \langle \text{idA}^{[T]}, B, k^{[T]} \rangle \right\}_{sk(\text{idA}^{[T]})} \right\}_{pk(B)}^a\right) \quad (5)$$

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

$$\wedge k^{[T]} \in \left\{ \left\{ \left\{ \langle \text{idA}^{[T]}, B, k^{[T]} \rangle \right\}_{sk(\text{idA}^{[T]})} \right\}_{pk(B)}^a \right\} \quad (6)$$

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

$$\Rightarrow \text{Send}(T', \text{enca}^{[T]}) \triangleleft \text{Receive}(T, \text{enca}^{[T]}) \quad (7)$$

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

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

$$\wedge \text{Receive}(T, \text{enca}^{[T]}) \triangleleft \text{Send}(T, \text{encb}^{[T]}) \quad (8)$$

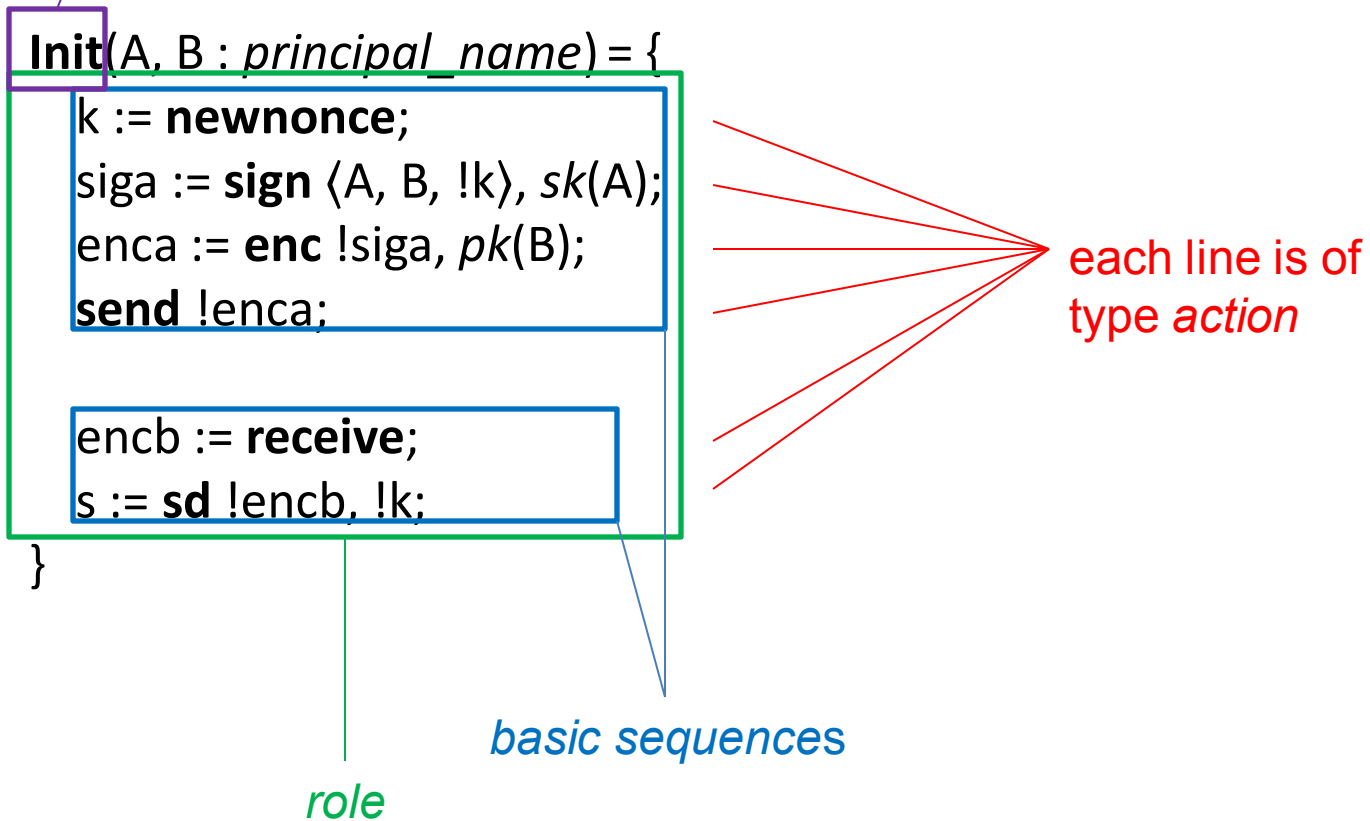
LF with subtyping

- *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
- $type ::= basic_type \mid type \times \dots \times type \mid type \rightarrow type \mid List(type)$
 $\mid Set_{fin}(type)$
- $term ::= fct_symb : type \mid variable : type$
 $\mid \langle term_1 \dots term_n \rangle \mid \pi_i term$
 $\mid \lambda variable : type_v . term \mid term term_1 \dots term_n \ (n \in \{0, 1, 2, \dots\})$
 $\mid \dots$
- *subtyping*: $type_1 \sqsubseteq type_2$
 - *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

$List(\text{principal_name}) \rightarrow \text{role}$

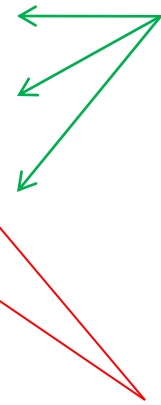
PCL syntax: *roles* and *actions*



PCL syntax: *locations* and some constant function symbols

constant function symbols of type
principal_name

```
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;  
}
```



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) = {  
  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 x *message* x *sgn_key* → *action*
location x *message* x *asym_enc_key* → *action*
message → *action*

location → *action*
location x *message* x *nonce* → *action*

PCL syntax: original types

Type name	Meta-variables (= variables)
<i>key</i>	K_1, K_2, \dots
<i>principal_name</i>	$Pname_1, Pname_2, \dots$
<i>role_name</i>	$rname_1, rname_2, \dots$
<i>action</i>	act_1, act_2, \dots
<i>thread_id</i>	tid_1, tid_2, \dots
<i>location</i>	loc_1, loc_2, \dots
<i>message</i>	msg_1, msg_2, \dots
<i>action_formula</i>	af_1, af_2, \dots
<i>nonmodal_formula</i>	$\Phi_1, \Phi_2, \dots; \Psi_1, \Psi_2, \dots$
<i>modal_formula</i>	$\Phi_1, \Phi_2, \dots; \Psi_1, \Psi_2, \dots$
<i>formula</i>	fml_1, fml_2, \dots
<i>statement</i>	stm_1, stm_2, \dots

PCL syntax: non-constructive type definitions

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

PCL syntax: subtype relations (1)

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

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

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

PCL syntax:

0-ary function symbols

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

PCL syntax:

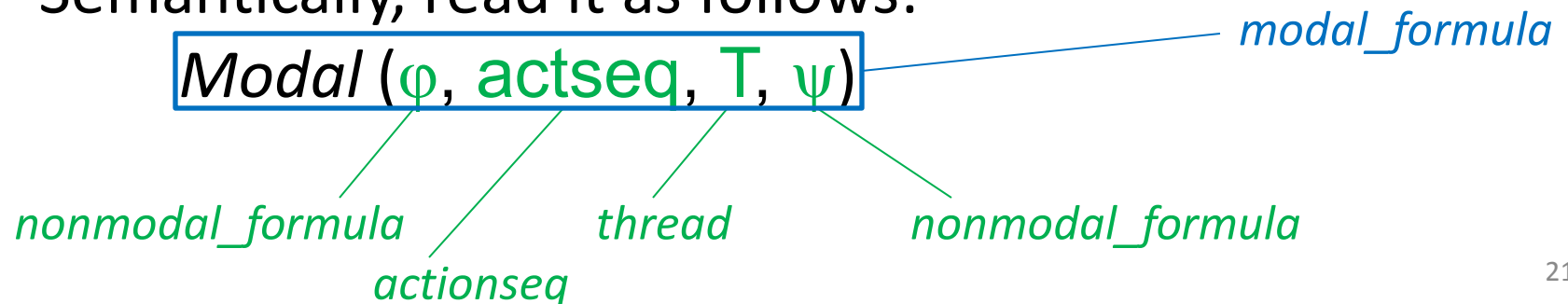
non-constant function symbols (excerpt 1)

Function symbol	Type
!	$location \rightarrow message$
send	$message \rightarrow action$
receive	$location \rightarrow action$
newnonce	$location \rightarrow action$
enc	$location \times message \times asym_enc_key \rightarrow action$
Send	$thread \times message \rightarrow action_formula$
Receive	$thread \times message \rightarrow action_formula$
Enc	$thread \times message \times key \rightarrow action_formula$
\triangleleft	$action_formula \times action_formula \rightarrow nonmodal_formula$
=	$message \times message \rightarrow nonmodal_formula$
=	$thread \times thread \rightarrow nonmodal_formula$
Honest	$principal_name \rightarrow nonmodal_formula$

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

Function symbol	Type
\neg	$nonmodal_formula \rightarrow nonmodal_formula$
\wedge	$nonmodal_formula \rightarrow nonmodal_formula$
<i>Modal</i> (shorthand:) $\cdot[\cdot].\cdot$	$nonmodal_formula \times actionseq \times thread \times nonmodal_formula \rightarrow modal_formula$
\models	$protocol \times run \times formula \rightarrow statement$
\forall_{type}	$(type \times statement) \rightarrow statement$ (for all types <i>type</i>)

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



PCL syntax vs. fully functional syntax

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

PCL syntax: equations

- $\pi_i \langle v_1, v_2, \dots, v_{i-1}, v_i, v_{i+1}, \dots, v_k \rangle = v_i$
[$k \in \mathbb{N}^+$; $i \in \{1, 2, \dots, k\}$]
- $\{ | \{ | \text{msg} | \}_K^a | \}^{-a}_K = \text{msg}$
[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*:

$$\mathcal{P}:\mathcal{R} \models \text{true} [\mathbf{Resp}(B)]_{\top} \text{Honest}(\text{idA}^{[\top]}) \wedge \text{idA}^{[\top]} \neq B \Rightarrow \exists T' . T'.pname = \text{idA}^{[\top]} \wedge \text{Send}(T', \text{enca}^{[\top]}) \triangleleft \text{Receive}(T, \text{enca}^{[\top]}) \wedge \text{Receive}(T, \text{enca}^{[\top]}) \triangleleft \text{Send}(T, \text{encb}^{[\top]})$$
- If $\mathcal{R}_{\text{prev}} = \langle S_{\text{princ}}, S_{\text{th}}, \text{eventlist}, \text{store} \rangle$ is a feasible *run* with respect to *protocol* \mathcal{P} , then

$$\mathcal{R} = \langle S_{\text{princ}}, S_{\text{th}}, \text{eventlist}:\langle \text{tid}:\text{act} \rangle, \text{store} \cup \text{newassgns} \rangle$$
 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 $\text{act} = \text{loc} := \mathbf{receive}$:
 - $\text{newassgns} = \{ \langle \langle \text{tid}, \text{loc} \rangle, \text{msg} \rangle \}$
 - condition:
 - msg was sent out previously by some thread.
 2. If $\text{act} = \mathbf{send} \text{ msg}$:
 - ...
 Etc. (14 cases)

Truth conditions (honesty)

- $\text{Auth}_{\text{Resp}}$:
 $\mathcal{P}:\mathcal{R} \models \text{true} [\mathbf{Resp}(B)]_{\top} \text{Honest}(\text{idA}^{[T]}) \wedge \text{idA}^{[T]} \neq B \Rightarrow \exists T' . T'.pname = \text{idA}^{[T]} \wedge \text{Send}(T', \text{enca}^{[T]}) \triangleleft \text{Receive}(T, \text{enca}^{[T]}) \wedge \text{Receive}(T, \text{enca}^{[T]}) \triangleleft \text{Send}(T, \text{encb}^{[T]})$
- $\mathcal{P}:\mathcal{R} \models \text{Honest}(\text{Pname})$:
 - 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_keys* are only ever used to encrypt or sign messages but not as the “payload”.

Truth conditions (modal formulas)

- $\mathcal{P}:\mathcal{R} \models \varphi [\text{actseq}]_{\top} \psi$:
 - For all divisions $(\mathcal{R}_1:\mathcal{R}_2:\mathcal{R}_3)$ of *run* \mathcal{R} :
 - If $\mathcal{P}:\mathcal{R}_1 \models \varphi$ is true, and $(\mathcal{R}_2.\text{elist}|_{\top} \setminus \mathcal{R}_1.\text{elist}|_{\top})$ matches *actseq*, then $\mathcal{P}:\mathcal{R}_2 \models \psi$ 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 sequenceare defined in our upcoming book chapter.

Axiom and rule schemas

- Axiom schema FS2:

$$\begin{aligned} & ((\text{NewNonce}(T_1, \text{msg}_1) \wedge \text{FirstSend}(T_1, \text{msg}_1, \text{msg}_2)) \\ & \quad \wedge (T_1 \neq T_2 \wedge \text{msg}_1 \in \text{msg}_3 \wedge \text{Receive}(T_2, \text{msg}_3)) \\ & \quad \Rightarrow \text{Send}(T_1, \text{msg}_2) \triangleleft \text{Receive}(T_2, \text{msg}_3) \end{aligned}$$

- Rule schema SEQ:

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

HON,AN3, $\text{Honest}(T_0) \wedge T_0.pname = X \wedge \text{Sign}(T_0, \langle X, Y, k_0 \rangle, sk(X))$

$$\mathbf{P2,FS1} \quad \Rightarrow \text{FirstSend}\left(T_0, k_0, \left\{ \left\{ \langle X, Y, k_0 \rangle \right\}_{sk(X)} \right\}_{pk(Y)}^a \right) \quad (4)$$

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

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

$$\wedge \text{FirstSend}\left(T', k^{[T]}, \left\{ \left\{ \langle \text{idA}^{[T]}, B, k^{[T]} \rangle \right\}_{sk(\text{idA}^{[T]})} \right\}_{pk(B)}^a \right) \quad (5)$$

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

$$\wedge k^{[T]} \in \left\{ \left\{ \left\{ \langle \text{idA}^{[T]}, B, k^{[T]} \rangle \right\}_{sk(\text{idA}^{[T]})} \right\}_{pk(B)}^a \right\} \quad (6)$$

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

$$\Rightarrow \text{Send}(T', \text{enca}^{[T]}) \triangleleft \text{Receive}(T, \text{enca}^{[T]}) \quad (7)$$

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

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

$$\wedge \text{Receive}(T, \text{enca}^{[T]}) \triangleleft \text{Send}(T, \text{encb}^{[T]}) \quad (8)$$

Thanks

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