COL876: SPECIAL TOPICS IN FORMAL METHODS

Formal verification of security protocols

Lecture 4, 7 August 2023

RECAP

- Saw a high-level overview of the active intruder problem
- Alternative presentation for inference: equational theories

$$t := m \mid \mathrm{pk}(k) \mid (t_1, t_2) \mid \mathrm{aenc}(t, \mathrm{pk}(k))$$

$$\mathrm{fst}((t_1, t_2)) = t_1$$

$$\mathrm{snd}((t_1, t_2)) = t_2$$

$$\mathrm{adec}(\mathrm{aenc}(t, \mathrm{pk}(k)), k) = t$$

TODAY

- A programming-style representation of protocols
- Helps formalize some details we kept implicit so far
- Needs us to utilize equational theories in the description
- See how to write out protocols in this, the *applied-pi calculus*

APPLIED-PI CALCULUS: GRAMMAR

$P, Q :=$	plain process	
	0	[null process]
	$P \mid Q$	[parallel composition]
	$!P$	[replication]
	$\nu n.P$	[name restriction]
	if $t_1 = t_2$ then P else Q	[conditional branching]
	in(c, x).P	[receive action]
	out(c, t).P	[send action]
	let x = t in P	[let binding]

ALICE-BOB VS APPLIED-PI

Alice-Bob Notation	**Applied-Pi Calculus**
Known set of agents	Generate agents dynamically
Agents identified by name	Agents identified by key
Only constructors for terms	Both constructors & destructors
New terms: Inference system	New terms: *recipes*
Received message is a *pattern*	Received message is a variable

FORMALIZING EXECUTIONS

$$A \rightarrow B : A, \mathsf{enc}(m, \mathsf{pk}(B))$$
$$B \rightarrow A : \mathsf{enc}(m, \mathsf{pk}(A))$$

```
init(ski: skey, pkr: pkey) {
   new n: bytes;
   send(pk(ski), aenc(n,pkr));
   recv(x: bytes);
   if (adec(x,ski) =/= n)
     error;
}
```

```
resp(skr: skey) {
   recv(k: pkey, y: bytes);
   let
      z = adec(y, skr)
   in
      send(aenc(z,k));
}
```

FORMALIZING EXECUTIONS

```
init(ski: skey, pkr: pkey) {
   new n: bytes;
   send(pk(ski), aenc(n,pkr));
   recv(x: bytes);
   if (adec(x,ski) =≠ n)
     error;
}
```

$P_i(ski, pkr) \triangleq$

$\nu n.\ out(c, aenc(n, pkr)).$

$in(c, x).$

$if(adec(x, ski) == n)$ then

SUCCESS

FORMALIZING EXECUTIONS

```
resp(skr: skey) {
   recv(k: pkey, y: bytes);
   let
      z = adec(y, skr)
   in
      send(aenc(z,k));
}
```

$P_r(skr) \triangleq$

$$\begin{aligned}
&\mathrm{in}(c, y). \\
&\mathrm{let}\ pka = \mathrm{fst}(y)\ \mathrm{in} \\
&\mathrm{let}\ z = \mathrm{adec}(y, skr)\ \mathrm{in} \\
&\mathrm{out}(c, \mathrm{aenc}(z, pka))
\end{aligned}$$

A FIRST ATTEMPT

- $P_i(ski, pkr) \triangleq \nu n.\ \mathrm{out}(c, \mathrm{aenc}(n, pkr)).\ \mathrm{in}(c, x).\ \mathrm{if}(\mathrm{adec}(x, ski) == n)$ then SUCCESS
- $P_r(skr) \triangleq \mathrm{in}(c, y).\ \mathrm{let}\ pka = \mathrm{fst}(y)\ \mathrm{in}.\ \mathrm{let}\ z = \mathrm{adec}(y, skr)\ \mathrm{in}.\ \mathrm{out}(c, \mathrm{aenc}(z, pka))$
- Have to put these two roles together to get an execution of the overall protocol?
- Agent with key $\mathrm{pk}(sk_a)$ executes an instance of P_i, while the agent with key $\mathrm{pk}(sk_b)$ executes an instance of P_r
- We also output the agents' public keys to make them available to the intruder
- $Pr^I \triangleq \nu sk_a.\ \nu sk_b.\ (\ P_i(sk_a, \mathrm{pk}(sk_b))\ |\ P_r(sk_b)\ |\ \mathrm{out}(c, \mathrm{pk}(sk_a))\ |\ \mathrm{out}(c, \mathrm{pk}(sk_b))\ )$

INTRUDER? WHAT INTRUDER?

- Okay, so we captured the MitM attack on that protocol.
- Recall that the adversary has a wide array of abilities
  - Most of these are not formalized in Pr^2!
  - We do not a priori know the attack on a given protocol
  - Formalism needs to be able to find any *possible* attack
- What about some attack where
  - the intruder mixes-and-matches terms, and
  - maybe requires A to talk to someone else? The intruder themselves, maybe?

A SECOND ATTEMPT

- $P_i(ski, pkr) \triangleq \nu n.\ \mathrm{out}(c, \mathrm{aenc}(n, pkr)).\ \mathrm{in}(c, x).\ \mathrm{if}(\mathrm{adec}(x, ski) == n)$ then SUCCESS
- $P_r(skr) \triangleq \mathrm{in}(c, y).$ let $pka = \mathrm{fst}(y)$ in. let $z = \mathrm{adec}(y, skr)$ in. $\mathrm{out}(c, \mathrm{aenc}(z, pka))$
- Explicitly model an instance of P_i where the agent with key sk_a talks to the intruder (who has key sk_c)
  - sk_c is just a free name; free names by default accessible to the intruder
- If the intruder starts a P_i instance, we only need to model a P_r instance by an honest agent
- $Pr^2 \triangleq \nu sk_a.\ \nu sk_b.\ (\ P_i(sk_a, \mathrm{pk}(sk_b))\ |\ P_i(sk_a, \mathrm{pk}(sk_c))\ |\ P_r(sk_b)\ |$

  $\mathrm{out}(c, \mathrm{pk}(sk_a))\ |\ \mathrm{out}(c, \mathrm{pk}(sk_b))\ )$

A THIRD ATTEMPT

- $P_i(ski, pkr) \triangleq \nu n.\ \mathrm{out}(c, \mathrm{aenc}(n, pkr)).\ \mathrm{in}(c, x).\ \mathrm{if}(\mathrm{adec}(x, ski) == n)$ then SUCCESS
- $P_r(skr) \triangleq \mathrm{in}(c, y).$ let $pka = \mathrm{fst}(y)$ in. let $z = \mathrm{adec}(y, skr)$ in. $\mathrm{out}(c, \mathrm{aenc}(z, pka))$
- Allow the intruder to pick who starts a session with the agent executing P_i
  - Add an input to have the intruder "feed" any public key to the P_i role
  - Could be $\mathrm{pk}(sk_a)$ or $\mathrm{pk}(sk_b)$, or even the intruder's own public key $\mathrm{pk}(sk_c)$
- $Pr^3 \triangleq \nu sk_a.\ \nu sk_b.\ (\ \mathrm{in}(c, x_{pk}).\ P_i(sk_a, x_{pk}) \mid P_r(sk_b) \mid \mathrm{out}(c, \mathrm{pk}(sk_a)) \mid \mathrm{out}(c, \mathrm{pk}(sk_b))\ )$

MORE MISSING ELEMENTS

- Can have unboundedly many sessions in parallel
- Need to add replication

A FOURTH ATTEMPT

- $P_i(ski, pkr) \triangleq \nu n.\ \mathrm{out}(c, \mathrm{aenc}(n, pkr)).\ \mathrm{in}(c, x).\ \mathrm{if}(\mathrm{adec}(x, ski) == n)\ \mathrm{then\ SUCCESS}$

- $P_r(skr) \triangleq \mathrm{in}(c, y).\ \mathrm{let}\ pka = \mathrm{fst}(y)\ \mathrm{in}.\ \mathrm{let}\ z = \mathrm{adec}(y, skr)\ \mathrm{in}.\ \mathrm{out}(c, \mathrm{aenc}(z, pka))$

- $Pr^4 \triangleq \nu sk_a.\ \nu sk_b.(\ !\mathrm{in}(c, x_{pk}).\ P_i(sk_a, x_{pk})\ |\ !P_r(sk_b)\ |\ \mathrm{out}(c, \mathrm{pk}(sk_a))\ |\ \mathrm{out}(c, \mathrm{pk}(sk_b))\ )$

- Allow unboundedly many copies of the initiator role (talking to anyone the intruder picks), and the responder role

- Still not enough! What's wrong now?

A FIFTH (FINAL?) ATTEMPT

- $P_i(ski, pkr) \triangleq \nu n.\ \mathrm{out}(c, \mathrm{aenc}(n, pkr)).\ \mathrm{in}(c, x).\ \mathrm{if}(\mathrm{adec}(x, ski) == n)$ then SUCCESS
- $P_r(skr) \triangleq \mathrm{in}(c, y).\ \mathrm{let}\ pka = \mathrm{fst}(y)\ \mathrm{in}.\ \mathrm{let}\ z = \mathrm{adec}(y, skr)\ \mathrm{in}.\ \mathrm{out}(c, \mathrm{aenc}(z, pka))$
- $$\begin{aligned} Pr^5 \triangleq\ !\nu sk_a.!\ \nu sk_b.(\ & !\mathrm{in}(c, x_{pk}).\ P_i(sk_a, x_{pk})\ |\ !P_r(sk_b)\ | \\ & !\mathrm{in}(c, x_{pk}).\ P_i(sk_b, x_{pk})\ |\ !P_r(sk_a)\ | \\ & \mathrm{out}(c, \mathrm{pk}(sk_a))\ |\ \mathrm{out}(c, \mathrm{pk}(sk_b))\ ) \end{aligned}$$
- Allow the same agent to play either role; allow unboundedly many honest agents
- Can write this out more succinctly as follows:

$$Pr \triangleq\ !\nu sk.(\ !\mathrm{in}(c, x_{pk}).\ P_i(sk, x_{pk})\ |\ !P_r(sk)\ |\ \mathrm{out}(c, \mathrm{pk}(sk))\ )$$

INTRUDER KNOWLEDGE

- Intruder controls network
- Messages sent onto channel c added to intruder knowledge
- Intruder stores every message along with a variable pointing to it
  - Denoted by a *substitution* $\sigma = [x_1 \mapsto t_1, \ldots, x_n \mapsto t_n]$
  - $dom(\sigma) = \{x_1, \ldots, x_n\}$ and $rng(\sigma) = \{t_1, \ldots, t_n\}$
  - Each t_i a term without variables or destructors
- Messages received from c should be derivable from σ
- t is derivable from σ iff $rng(\sigma) \vdash t$ *

OTHER BOOKKEEPING

- νn.P evolves like the process P; uses a fresh name m in place of n
- Fresh names are private; cannot be accessed by the intruder
- Names outside the scope of a ν are assumed to be public
- Have to keep track of all fresh names generated during a process
- Processes involve replication; track a multiset of processes

CONFIGURATIONS

- A *configuration* of a process is a triple $\mathscr{C} := (\mathscr{P}, \tilde{n}, \sigma)$ where
  - $\mathscr{P}$ is a finite multiset of processes
  - $\tilde{n}$ is a finite set of freshly generated names
  - σ is a finite substitution mapping variables to messages
- An *extended process* is a configuration $(\{P_1, \ldots, P_n\}, \tilde{n}, \sigma)$
  - For simplicity, we will write this as $\nu\tilde{n} \,.\, (P_1 \mid \ldots \mid P_n \mid \sigma)$
- Process evolution: transition (*reduction*) rules on configurations

REDUCTION RULES 1

$$
\begin{array}{llllllll}
(\mathcal{P} \cup \{0\}, & \tilde{n}, & \sigma) & \xrightarrow{\tau} & (\mathcal{P}, & \tilde{n}, & \sigma) & \\
(\mathcal{P} \cup \{P \mid Q\}, & \tilde{n}, & \sigma) & \xrightarrow{\tau} & (\mathcal{P} \cup \{P, Q\}, & \tilde{n}, & \sigma) & \\
(\mathcal{P} \cup \{!\, P\}, & \tilde{n}, & \sigma) & \xrightarrow{\tau} & (\mathcal{P} \cup \{!\, P, P\}, & \tilde{n}, & \sigma) & \\
(\mathcal{P} \cup \{\mathbf{if}\ t = u\ \mathbf{then}\ P\ \mathbf{else}\ Q\}, & \tilde{n}, & \sigma) & \xrightarrow{\tau} & (\mathcal{P} \cup \{P\}, & \tilde{n}, & \sigma) & \text{if } t =_R u \\
(\mathcal{P} \cup \{\mathbf{if}\ t = u\ \mathbf{then}\ P\ \mathbf{else}\ Q\}, & \tilde{n}, & \sigma) & \xrightarrow{\tau} & (\mathcal{P} \cup \{Q\}, & \tilde{n}, & \sigma) & \text{if } t \neq_R u \\
 & & & & & & & t, u \text{ ground}
\end{array}
$$

REDUCTION RULES 2

$$
\begin{array}{llllllll}
(\mathscr{P} \cup \{\mathbf{let}\ x = t\ \mathbf{in}\ P\}, & \tilde{n}, & \sigma) & \xrightarrow{\tau} & (\mathscr{P} \cup \{P[x \mapsto t]\}, & \tilde{n}, & \sigma) & \\
(\mathscr{P} \cup \{\nu n.P\}, & \tilde{n}, & \sigma) & \xrightarrow{\tau} & (\mathscr{P} \cup \{P[n \mapsto m]\}, & \tilde{n} \cup \{m\}, & \sigma) & m \text{ fresh} \\
(\mathscr{P} \cup \{\mathbf{out}(c, t).P\}, & \tilde{n}, & \sigma) & \xrightarrow{+x} & (\mathscr{P} \cup \{P\}, & \tilde{n}, & \sigma') & x \notin dom(\sigma) \\
 & & & & & & & \sigma' = \sigma \cup [x \mapsto t\downarrow] \\
(\mathscr{P} \cup \{\mathbf{in}(c, x).P\}, & \tilde{n}, & \sigma) & \xrightarrow{-r} & (\mathscr{P} \cup \{P[x \mapsto t]\}, & \tilde{n}, & \sigma) & t \text{ constructed using} \\
 & & & & & & & \text{the recipe } r
\end{array}
$$

- $P[t \mapsto u]$ denotes P where each free t is replaced by u
- m is fresh iff $m \notin \tilde{n} \cup \text{names}(\mathscr{P} \cup \{P\}) \cup \text{names}(rng(\sigma))$

FRAMES

- Attacker knowledge captured via a *frame* $\varphi = \nu\tilde{n} \cdot \sigma$
  - A substitution with some bound names
- The frame of a configuration $\mathcal{C} = (\mathcal{P}, \tilde{n}, \sigma)$ is $\varphi(\mathcal{C}) := \nu\tilde{n} \cdot \sigma$
- For $\varphi = \nu\tilde{n} \cdot \sigma$, we say $\varphi \vdash t$ if $rng(\sigma) \cup (\mathcal{N} \setminus \tilde{n}) \vdash t$
- Can also be expressed in terms of *recipes*

RECIPES

- r is a φ-recipe for a term t if
  - $\mathrm{vars}(r) \subseteq dom(\sigma)$
  - $\mathrm{names}(r) \cap \tilde{n} = \emptyset$
  - $t =_{\mathrm{R}} r\sigma$ (where $=_{\mathrm{R}}$ is the equational theory under consideration)
- Note that any name not bound in $\mathscr{C}$ can be used by the attacker

NOW WHAT?

- We now have an abstract formal model in which to formalize protocols
- Now we need to specify properties as checks over this model
- Interested in various properties
  - Secrecy ("nobody but <some parties> should know t")
  - Authentication ("If A thinks she's talking to B, B should have spoken to A")
  - Agreement ("If A and B think they share value v with each other, that is the case")
  - Privacy ("Nobody should know that agent A holds value a, even if A and a are themselves publicly known values)…

PROPERTIES

- Two main classes of properties: *trace* and *equivalence*
- *Trace*: verified by examining one run of the protocol at a time
  - Secrecy: There is no run of the protocol where I knows *m*
  - Agreement: In every run of the protocol where A and B participate, if A thinks they share some freshly-generated value *v* with B, then B does share v with A.

SECRECY IN APPLIED-PI

- m is secret in a protocol iff there is no run where the configuration yields a frame which can derive m
- m is bound under a ν operator in our example protocol
- How do we even specify that m is intended to be secret?

SECRECY: FORMALIZED

- Rename bound variables to avoid name clashes
- Use a *monitor* process annotated with *events*
- A reduction sequence $P_0 \xrightarrow{\gamma_1} P_1 \cdots \xrightarrow{\gamma_n} P_n$ satisfies an event *e(t)* iff there is an *i* such that *e(t)* appears in P_i
- Let $P = \nu s \cdot P'$, and *leak* be an event that does not occur in P
- Define $P^s := \nu s\ .\ (P' \mid (\text{ in}(c, x).\ \text{if } x = s \text{ then event } leak(s) \text{ else } 0\ )\ )$
- *s* is secret in P iff there is no reduction sequence starting from P^s which satisfies *leak(s)*

MORE TRACE PROPERTIES

- Correspondence properties: “If an event e happened, then an event e’ must have happened before”
- Examples: Authentication, agreement etc
  - Authentication: “If B finished an execution of the protocol with A, then A must have started an execution with B earlier”
  - Agreement: “If B thinks they share a value v with A, then A must have generated v for use with B”
  - Various flavours: aliveness, weak agreement, injective agreement &c.

CORRESPONDENCE: FORMALIZED

- $e_0(\overrightarrow{t_0}) \triangleright e_1(\overrightarrow{t_1})$ denotes the following correspondence: "if $e_1(\overrightarrow{t_1})$ occurred in a run, then $e_0(\overrightarrow{t_0})$ occurred earlier"

- A reduction sequence $P_0 \xrightarrow{\gamma_1} P_1 \cdots \xrightarrow{\gamma_n} P_n$ satisfies a correspondence $e_0(\overrightarrow{t_0}) \triangleright e_1(\overrightarrow{t_1})$ iff for any σ,

  whenever $e_1(\overrightarrow{t_1}\sigma)$ occurs in some P_i, there is a $j \leq i$ such that $e_0(\overrightarrow{t_0}\sigma)$ occurs in P_j

- A process P satisfies a correspondence property iff all reduction sequences starting from P satisfy it.

EQUIVALENCE PROPERTIES

- *Equivalence*: require simultaneous examination of multiple protocol runs, often to ensure link between two values is secret
  - Strong Secrecy: The attacker should not be able to link an input of their choice to the value of some observable variable.
  - Voter anonymity: The attacker should not be able to link a voter's identity to their vote.
- Need to identify what differences the attacker can observe between multiple runs
- Simplest possible observation: does variable x map to the same term in all runs?

STATIC EQUIVALENCE

- Frames φ_1 & φ_2 with $\sigma_1 = [x \mapsto 0, y \mapsto 1]$ & $\sigma_2 = [x \mapsto 1, y \mapsto 0]$
- Can learn the same terms from both frames
    - But need different recipes for the same term!
- Capture ability to compare messages via *static equivalence*
- Formalize what equalities the attacker can learn from a frame

STATIC EQUIVALENCE

- Consider a frame and terms t and u
- We say $\varphi \vDash t =_R u$ iff there are $\tilde{n}$ and σ such that:
  - $\varphi = \nu\tilde{n}\,.\,\sigma$ (after appropriate variable renaming)
  - $(\text{names}(t) \cup \text{names}(u)) \cap \tilde{n} = \varnothing$
  - $\text{vars}(t) \cup \text{vars}(u) \subseteq dom(\sigma)$
  - $\mathrm{t}\sigma =_{\mathrm{R}} u\sigma$
- Two frames $\varphi_1 = \nu\widetilde{n_1}\,.\,\sigma_1$ and $\varphi_2 = \nu\widetilde{n_2}\,.\,\sigma_2$ are statically equivalent (denoted $\varphi_1 \sim \varphi_2$) iff
  - $\text{dom}(\sigma_1) = \text{dom}(\sigma_2)$, and
  - for any terms t and u, $\varphi_1 \vDash t =_R u$ iff $\varphi_2 \vDash t =_R u$

OBSERVATIONAL EQUIVALENCE

- But what about a property like voter anonymity?
- "The attacker should not be able to link a voter's identity to their vote"
  - Left implicit: "No matter what the attacker does"!
- How do we formalize this bit?

OBSERVATIONAL EQUIVALENCE

- Use *contexts*
    - A context is a process capturing intruder behaviour with a hole, where we can plug in the process under examination
    - Quantifying over contexts captures all possible intruder behaviours
- Two processes are *observationally equivalent* if
    - any sequence of reduction rules results in observationally equivalent processes, and
    - if they remain observationally equivalent under any context