Formal verification of security protocols
RECAP

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

\[
\begin{align*}
t &:= m \mid \text{pk}(k) \mid (t_1, t_2) \mid \text{aenc}(t, \text{pk}(k)) \\
\text{fst}((t_1, t_2)) &= t_1 \\
\text{snd}((t_1, t_2)) &= t_2 \\
\text{adec}(&\text{aenc}(t, \text{pk}(k)), k) = t
\end{align*}
\]
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

\[ P, Q := \text{plain process} \]

\[ \circ \quad \text{[null process]} \]

\[ P \mid Q \quad \text{[parallel composition]} \]

\[ !P \quad \text{[replication]} \]

\[ \nu n.P \quad \text{[name restriction]} \]

\[ \text{if } t_1 = t_2 \text{ then } P \text{ else } Q \quad \text{[conditional branching]} \]

\[ \text{in}(c, x).P \quad \text{[receive action]} \]

\[ \text{out}(c, t).P \quad \text{[send action]} \]

\[ \text{let } x = t \text{ in } P \quad \text{[let binding]} \]
# ALICE-BOB VS APPLIED-PI

<table>
<thead>
<tr>
<th>Alice-Bob Notation</th>
<th>Applied-Pi Calculus</th>
</tr>
</thead>
<tbody>
<tr>
<td>Known set of agents</td>
<td>Generate agents dynamically</td>
</tr>
<tr>
<td>Agents identified by name</td>
<td>Agents identified by key</td>
</tr>
<tr>
<td>Only constructors for terms</td>
<td>Both constructors &amp; destructors</td>
</tr>
<tr>
<td>New terms: Inference system</td>
<td>New terms: recipes</td>
</tr>
<tr>
<td>Received message is a pattern</td>
<td>Received message is a variable</td>
</tr>
</tbody>
</table>
FORMALIZING EXECUTIONS

\[ A \rightarrow B : A, \text{enc}(m, \text{pk}(B)) \]
\[ B \rightarrow A : \text{enc}(m, \text{pk}(A)) \]

**init**(ski: skey, pkr: pkey) {
  new n: bytes;
  send(pk(ski), aenc(n, pkr));
  recv(x: bytes);
  if (adec(x, ski) \neq n)
    error;
}

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

\[ P_i(ski, pkr) \triangleq \begin{aligned} &\nu n. \text{ out}(c, \text{ aenc}(n, pkr)). \\
&\text{in}(c, x). \\
&\text{if}(\text{adec}(x, ski) = n) \text{ then SUCCESS} \end{aligned} \]

\text{init}(ski: \text{skey}, pkr: \text{pkey}) \{ \\
\text{new } n: \text{bytes}; \\
\text{send(pk}(ski), \text{aenc}(n,pkr)); \\
\text{recv}(x: \text{bytes}); \\
\text{if (adec}(x,\text{ski}) \neq n) \\
\text{error;} \\
\}
FORMALIZING EXECUTIONS

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

\[ \text{Pr}(\text{skr}) \triangleq \]
\[ \text{in}(c, y). \]
\[ \text{let} \text{pka} = \text{fst}(y) \text{ in} \]
\[ \text{let} z = \text{adec}(y, \text{skr}) \text{ in} \]
\[ \text{out}(c, \text{aenc}(z, \text{pka})) \]
A FIRST ATTEMPT

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

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

- Have to put these two roles together to get an execution of the overall protocol?

- Agent with key pk($sk_a$) executes an instance of $P_i$, while the agent with key pk($sk_b$) executes an instance of $P_r$

- We also output the agents’ public keys to make them available to the intruder

- $Pr^i \triangleq \nu sk_a. \nu sk_b. (\ P_i(sk_a, pk(sk_b)) \mid P_r(sk_b) \mid \text{out}(c, pk(sk_a)) \mid \text{out}(c, pk(sk_b)) )$
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 Pr²!
- 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

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

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

- 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

- \( P_r^2 \triangleq \nu sk_a. \nu sk_b. (P_i(sk_a, \text{pk}(sk_b)) \mid P_i(sk_a, \text{pk}(sk_c)) \mid P_r(sk_b) \mid \text{out}(c, \text{pk}(sk_a)) \mid \text{out}(c, \text{pk}(sk_b))) \)
A THIRD ATTEMPT

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

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

- 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 \( \text{pk}(sk_a) \) or \( \text{pk}(sk_b) \), or even the intruder’s own public key \( \text{pk}(sk_c) \)

- \( \text{Pr}^3 \triangleq \nu sk_a. \nu sk_b. (\text{in}(c, x_{pk}). P_i(sk_a, x_{pk}) | P_r(sk_b) | \text{out}(c, \text{pk}(sk_a)) | \text{out}(c, \text{pk}(sk_b)) ) \)
MORE MISSING ELEMENTS

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

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

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

- \( Pr^4 \triangleq \nu sk_a. \nu sk_b. (\cdots) \)

- 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

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

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

- \( \Pr^5 \triangleq \nu sk_a. !\nu sk_b. ( !\text{in}(c, x_{pk}). P_i(sk_a, x_{pk}) | !P_r(sk_b) | \\
  !\text{in}(c, x_{pk}). P_i(sk_b, x_{pk}) | !P_r(sk_a) | \\
  \text{out}(c, pk(sk_a)) | \text{out}(c, pk(sk_b)) ) \)

- Allow the same agent to play either role; allow unboundedly many honest agents

- Can write this out more succinctly as follows:

  \( \Pr \triangleq !\nu sk. ( !\text{in}(c, x_{pk}). P_i(sk, x_{pk}) | !P_r(sk) | \text{out}(c, pk(sk)) ) \)
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 $\sigma$
- $t$ is derivable from $\sigma$ iff $rng(\sigma) \vdash t^*$
OTHER BOOKKEEPING

- $\nu n.P$ 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 $\nu$ 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
A configuration of a process is a triple $\mathcal{C} := (\mathcal{P}, \tilde{n}, \sigma)$ where

- $\mathcal{P}$ is a finite multiset of processes
- $\tilde{n}$ is a finite set of freshly generated names
- $\sigma$ 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 $\nu\tilde{n}. (P_1 | \ldots | P_n | \sigma)$

Process evolution: transition (reduction) rules on configurations
REDUCTION RULES 1

\[(\mathcal{P} \cup \{\sigma\}, \tilde{n}, \sigma) \rightarrow (\mathcal{P}, \tilde{n}, \sigma)\]

\[(\mathcal{P} \cup \{P \mid Q\}, \tilde{n}, \sigma) \rightarrow (\mathcal{P} \cup \{P, Q\}, \tilde{n}, \sigma)\]

\[(\mathcal{P} \cup \{! P\}, \tilde{n}, \sigma) \rightarrow (\mathcal{P} \cup \{! P, P\}, \tilde{n}, \sigma)\]

\[(\mathcal{P} \cup \{\text{if } t = u \text{ then } P \text{ else } Q\}, \tilde{n}, \sigma) \rightarrow (\mathcal{P} \cup \{P\}, \tilde{n}, \sigma) \quad \text{if } t =_{R} u\]

\[(\mathcal{P} \cup \{\text{if } t = u \text{ then } P \text{ else } Q\}, \tilde{n}, \sigma) \rightarrow (\mathcal{P} \cup \{Q\}, \tilde{n}, \sigma) \quad \text{if } t \neq_{R} u\]

\[t, u \text{ ground}\]
REDUCTION RULES 2

(P \cup \{let \ x = t \ in \ P\}, \ \tilde{n}, \ \sigma) \xrightarrow{\cdot} (P \cup \{P[x \rightarrow t]\}, \ \tilde{n}, \ \sigma)

(P \cup \{v n. P\}, \ \tilde{n}, \ \sigma) \xrightarrow{\cdot} (P \cup \{P[n \rightarrow m]\}, \ \tilde{n} \cup \{m\}, \ \sigma) \quad m \ fresh

(P \cup \{out(c, t).P\}, \ \tilde{n}, \ \sigma) \xrightarrow{+x} (P \cup \{P\}, \ \tilde{n}, \ \sigma') \quad x \notin \text{dom}(\sigma)
\quad \sigma' = \sigma \cup [x \rightarrow t \downarrow]

(P \cup \{in(c, x).P\}, \ \tilde{n}, \ \sigma) \xrightarrow{-r} (P \cup \{P[x \rightarrow t]\}, \ \tilde{n}, \ \sigma) \quad t \ constructed \ using \ the \ recipe \ r

- P[t \rightarrow u] \ denotes \ P \ where \ each \ free \ t \ is \ replaced \ by \ u

- m \ is \ fresh \ iff \ m \notin \tilde{n} \cup \text{names}(P \cup \{P\}) \cup \text{names}(\text{rng}(\sigma))
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 $\text{rng}(\sigma) \cup (\mathcal{N} \setminus \tilde{n}) \vdash t$

Can also be expressed in terms of recipes
RECIPES

- $r$ is a $\varphi$-recipe for a term $t$ if
  - $\text{vars}(r) \subseteq \text{dom}(\sigma)$
  - $\text{names}(r) \cap \tilde{n} = \emptyset$
  - $t =_R r\sigma$ (where $=_R$ is the equational theory under consideration)
- Note that any name not bound in $\mathcal{C}$ 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")...
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.
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 \( \nu \) 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 $P^s := \nu s \cdot ( P' | (\text{ in(c, x). if } x = s \text{ then event } \text{leak}(s) \text{ else } \circ ) )$
- $s$ is secret in $P$ iff there is no reduction sequence starting from $P^s$ which satisfies $\text{leak}(s)$
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**

- \( e_0(\vec{t}_0) \triangleright e_1(\vec{t}_1) \) denotes the following correspondence: “if \( e_1(\vec{t}_1) \) occurred in a run, then \( e_0(\vec{t}_0) \) occurred earlier”

- A reduction sequence \( P_0 \xrightarrow{\gamma_1} P_1 \cdots \xrightarrow{\gamma_n} P_n \) satisfies a correspondence \( e_0(\vec{t}_0) \triangleright e_1(\vec{t}_1) \) iff for any \( \sigma \),

\[
\text{whenever } e_1(\vec{t}_1 \sigma) \text{ occurs in some } P_i, \text{ there is a } j \leq i \text{ such that } e_0(\vec{t}_0 \sigma) \text{ occurs in } P_j
\]

- 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 $\varphi_1$ & $\varphi_2$ 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 $\sigma$ such that:
  - $\varphi = \nu\tilde{n} . \sigma$ (after appropriate variable renaming)
  - $(\text{names}(t) \cup \text{names}(u)) \cap \tilde{n} = \emptyset$
  - $\text{vars}(t) \cup \text{vars}(u) \subseteq \text{dom}(\sigma)$
  - $t\sigma =_R u\sigma$

- Two frames $\varphi_1 = \nu\tilde{n}_1 . \sigma_1$ and $\varphi_2 = \nu\tilde{n}_2 . \sigma_2$ 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