VERIFICATION OF SECURITY PROTOCOLS: THE APPLIED PI CALCULUS
Security Protocols

- A security protocol is a distributed procedure with the need to achieve a security goal.

- Examples of “security goal”:
  - Authentication
  - Privacy
  - Key agreement
  - Non-repudiation
  - ...
Example: Handshake Protocol

- C knows S's public key. S is willing to talk to any C (does not know their public keys in advance). They want to agree a session key; they communicate on a channel that is controlled by the attacker.
Example: Handshake Protocol

Intended properties:
- Secrecy: The value s is known only to C and S.
- Authentication of S: if C reaches the end of the protocol with session key k, then S proposed k for use by C.
- Authentication of C: if S reaches the end of the protocol and she believes she has session the key k with C, then C was indeed her interlocutor and she has session k.
Handshake Protocol Attack

Intended properties:

- Secrecy: The value $s$ is known only to $C$ and $S$.
- Authentication of $S$: if $C$ reaches the end of the protocol with session key $k$, then $S$ proposed $k$ for use by $C$.
- Authentication of $C$: if $S$ reaches the end of the protocol and she believes she has session the key $k$ with $C$, then $C$ was indeed her interlocutor and she has session $k$. 

\[ S \quad \text{new } k \quad M \quad C \]

\[
\text{enc}_{pkM}(\text{sign}_{skS}(k)) \quad \text{enc}_{pkC}(\text{sign}_{skS}(k)) \\
\text{sen}c_k(s) \quad \text{sen}c_k(s)
\]
The attack is avoided by making the package the initiator sends include the identity of the respondent.

The three properties hold of the revised protocol.

Our aim is to be able to automatically establish these facts.
Verifying Security Protocols

- **Provable/computational security**
  - Computationally bounded (polynomial) attacker
  - Exact cryptographic operations on bitstrings
  - Bitstring (more concrete) model
  - Prove difficulty of violating security property is equivalent to solving a hard problem

- **Formal/symbolic methods**
  - Idealized (worst case) attacker
  - Idealized (best case) perfect cryptography
  - Symbolic (more abstract) model of protocol
  - Prove impossibility of violating security property within the model
Two Views of Verification

- Provable security vs. Formal methods
  - Provable security provides stronger promises
  - But, “proofs are so turgid that other specialists don't even read them” [KoblitzMenezes'04]
  - Furthermore, they fail to detect certain kinds of attack [Meadows'03, KoblitzMenezes'04, SmythRyanChen'07]
  - Formal methods are simpler, specifications are nicer and automated support is available
  - Caveat: gulf between abstract formal model and real world specification (and the actual implementation)
The applied pi calculus is a language for describing concurrent processes and their interactions
- Developed explicitly for modeling security protocols
- Similar to pi calculus; with more general cryptography

ProVerif is a leading software tool for automated reasoning
- Takes applied pi processes and reasons about observational equivalence, correspondence assertions and secrecy
Terms

\[ L, M, N, T, U, V ::= \]

\[ a, b, c, k, m, n, s, t, r, \ldots \quad \text{name} \]

\[ x, y, z \quad \text{variable} \]

\[ g(M_1, \ldots, M_l) \quad \text{function} \]

Equational theory

Suppose we have defined nullary function \( ok \), unary function \( pk \), binary functions \( enc, dec, senc, sdec, sign \), and ternary function \( checksign \).

\[
\begin{align*}
\text{sdec}(x, \text{senc}(x, y)) &= y \\
\text{dec}(x, \text{enc}(\text{pk}(x), y)) &= y \\
\text{checksign}(\text{pk}(x), y, \text{sign}(x, y)) &= \text{ok}
\end{align*}
\]
**Applied Pi Calculus: Grammar**

**Processes (Plain Processes)**

<table>
<thead>
<tr>
<th>Grammar</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$0$</td>
<td>null process</td>
</tr>
<tr>
<td>$P</td>
<td>Q$</td>
</tr>
<tr>
<td>$!P$</td>
<td>replication</td>
</tr>
<tr>
<td>$\nu \ n. P$</td>
<td>name restriction</td>
</tr>
<tr>
<td>$u(x). P$</td>
<td>message input</td>
</tr>
<tr>
<td>$\bar{u} \langle M \rangle. P$</td>
<td>message output</td>
</tr>
<tr>
<td>if $M = N$ then $P$ else $Q$</td>
<td>cond’nl</td>
</tr>
</tbody>
</table>

- **syntactic substitution (usually just called substitution)**
  - replaces the variable $x$ with the term $M$

- We sometimes write $let \ x = M \ in \ P$ in place of $P\{M/x\}$
Applied Pi Calculus: Grammar

- Extended Processes

  \[ A, B, C ::= \text{extended processes} \]

  \[ P \quad \text{plain process} \]

  \[ A \mid B \quad \text{parallel comp.} \]

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

  \[ \nu x.A \quad \text{variable restriction} \]

  \[ \{M/x\} \quad \text{active substitution} \]

- active substitution vs. syntactic substitutions
Applied Pi Calculus: Grammar

- **Scope of names and variables**
  - Set of bound names $bn(A)$ contains every name $n$ which is under restriction $\nu n$ inside $A$.
  - Set of bound variables $bv(A)$ consists of all those variables $x$ occurring in $A$ that are bound by restriction $\nu x$ or input $u(x)$.
  - Also, free names $fn(A)$ and free variables $fv(A)$
Applied Pi Calculus: Grammar

- **Contexts**
  - may be used to represent the adversarial environment in which a process is run; that environment provides the data that the process inputs, and consumes the data that it outputs.
  - context C[-] : an extended process with a hole
  - C[A] : the result of filling C[-]’s hole with the extended process A
Handshake Protocol with Applied Pi Calculus

\[ P \triangleq \nu sk_S. \nu sk_C. \nu s. \]
\[
\text{let } pk_S = \text{pk}(sk_S) \text{ in let } pk_C = \text{pk}(sk_C) \text{ in }
\]
\[
(\overline{c}\langle pk_S \rangle \mid \overline{c}\langle pk_C \rangle \mid !P_S \mid !P_C)
\]

\[ P_S \triangleq c(x_p k). \nu k. \overline{c}\langle \text{aenc}(x_p k, \text{sign}(sk_S, k)) \rangle.
\]
\[
c(z). \text{if } \text{fst}(\text{sdec}(k, z)) = \text{tag} \text{ then } Q
\]

\[ P_C \triangleq c(y). \text{let } y' = \text{adem}(sk_C, y) \text{ in let } y_k = \text{getmsg}(y') \text{ in }
\]
\[
\text{if } \text{checksign}(pk_S, y') = \text{true} \text{ then }
\]
\[
\overline{c}\langle \text{senc}(y_k, \text{pair(tag, s))} \rangle
\]
Operational Semantics: Structural Equivalence

Informally, two processes are structurally equivalent if they model the same thing.

<table>
<thead>
<tr>
<th>Rule</th>
<th>Equation</th>
</tr>
</thead>
<tbody>
<tr>
<td>PAR-0</td>
<td>$A \equiv A \mid 0$</td>
</tr>
<tr>
<td>PAR-A</td>
<td>$(A \mid (B \mid C)) \equiv (A \mid B) \mid C$</td>
</tr>
<tr>
<td>PAR-C</td>
<td>$A \mid B \equiv B \mid A$</td>
</tr>
<tr>
<td>REPL</td>
<td>$!P \equiv P \mid !P$</td>
</tr>
<tr>
<td>NEW-0</td>
<td>$\nu n.0 \equiv 0$</td>
</tr>
<tr>
<td>NEW-C</td>
<td>$\nu u.v. w.A \equiv \nu w. v.u.A$</td>
</tr>
<tr>
<td>NEW-PAR</td>
<td>$A \mid \nu u.B \equiv \nu u.(A \mid B)$ where $u \notin \text{fv}(A) \cup \text{fn}(A)$</td>
</tr>
<tr>
<td>ALIAS</td>
<td>$\nu x.{M/x} \equiv 0$</td>
</tr>
<tr>
<td>SUBST</td>
<td>${M/x} \mid A \equiv {M/x} \mid A{M/x}$</td>
</tr>
<tr>
<td>REWRITE</td>
<td>${M/x} \equiv {N/x}$ where $M =_E N$</td>
</tr>
</tbody>
</table>
Operational Semantics: Internal Reduction

\begin{align*}
\text{COMM} & \quad \overline{c}(x).P \mid c(x).Q \rightarrow P \mid Q \\
\text{THEN} & \quad \text{if } N = N \text{ then } P \text{ else } Q \rightarrow P \\
\text{ELSE} & \quad \text{if } L = M \text{ then } P \text{ else } Q \rightarrow Q \\
& \quad \text{for ground terms } L, M \text{ where } L \not\equiv E M
\end{align*}
Operational Semantics: Labeled Reductions

- Labeled semantics: \( A \xrightarrow{\alpha} B \)
- \( A \xrightarrow{c(M)} B \) means that the process A performs an input of the term M from the environment on the channel c, and the resulting process is B.
- \( A \xrightarrow{c(u)} B \) means that the process A outputs the free u (which may be a variable, or a channel name).
- \( A \xrightarrow{\nu u.\overline{c}(u)} B \) means A outputs u that is restricted in A, and becomes free in B. Again, u is a channel name or a variable representing a term.
Operational Semantics: Labeled Reductions

\[ c(x).P \xrightarrow{c(M)} P\{M/x\} \]

**Out-Atom**

\[ \overline{c(u)}.P \xrightarrow{c(u)} P \]

**Open-Atom**

\[
\frac{A \xrightarrow{c(u)} A' \quad u \neq c}{\nu u.A \xrightarrow{\nu u.\overline{c(u)}} A'}
\]

**Scope**

\[
\frac{A \xrightarrow{\alpha} A' \quad \text{u does not occur in } \alpha}{\nu u.A \xrightarrow{\alpha} \nu u.A'}
\]

**Par**

\[
\frac{A \xrightarrow{\alpha} A' \quad \text{bv}(\alpha) \cap \text{fv}(B) = \text{bn}(\alpha) \cap \text{fn}(B) = \emptyset}{A \mid B \xrightarrow{\alpha} A' \mid B}
\]

**Struct**

\[
\frac{A \equiv B \quad B \xrightarrow{\alpha} B' \quad B' \equiv A'}{A \xrightarrow{\alpha} A'}
\]
Operational Semantics: Example

The process

\[ A \xrightarrow{\nu} s.(c(x).if \ x = s \ then \ \overline{c}(i\_got\_s)) \]

can never output \textit{i\_got\_s}, because no term input as \textit{x} can be equal to the `new' \textit{s} created by the process.

More precisely, there is no sequence of reductions

\[ A \xrightarrow{\alpha} \cdots \xrightarrow{\alpha} B \mid \{i\_got\_s/y\} \]

for some process \textit{B} and variable \textit{y}. 
Operational Semantics: Example

Consider $A''$:

$$A'' \trianglerighteq \nu \ s. (\overline{c} \langle \text{senc}(k, s) \rangle \cdot c(x). \text{if } x = s \text{ then } \overline{c} \langle i\_got\_s \rangle)$$

This test can succeed; the process can output $i\_got\_s$, as shown by the following execution:

$$A'' \quad\frac{\nu\ y.\overline{c}(y)}{c(sdec(k,y))} \quad \nu\ s.(c(x).\text{if } x = s \text{ then } \overline{c} \langle i\_got\_s \rangle | \{\text{senc}(k, s)/y\})$$

$$\equiv \quad \nu\ s.(\text{if } sdec(k, y) = s \text{ then } \overline{c} \langle i\_got\_s \rangle | \{\text{senc}(k, s)/y\})$$

$$\equiv \quad \nu\ s.(\text{if } sdec(k, \text{senc}(k, s)) = s \text{ then } \overline{c} \langle i\_got\_s \rangle | \{\text{senc}(k, s)/y\})$$

$$\equiv \quad \nu\ s.(\text{if } s = s \text{ then } \overline{c} \langle i\_got\_s \rangle | \{\text{senc}(k, s)/y\})$$

$$\rightarrow \quad \nu\ s.(\overline{c} \langle i\_got\_s \rangle | \{\text{senc}(k, s)/y\})$$

$$\quad\frac{\nu\ z.\overline{c}(z)}{\nu\ s.(\{\text{senc}(k, s)/y\} | \{i\_got\_s/z\})}$$

$$\equiv \quad \nu\ s.(\{\text{senc}(k, s)/y\}) | \{i\_got\_s/z\}$$
Secrecy Property

- Secrecy of $M$ is preserved if an adversary cannot construct $M$ from the outputs of the protocol.
- Formalize the adversary as a process $I$ running in parallel. If $I$ cannot output $M$, then secrecy is preserved.
- Syntactic secrecy

A closed plain process $P$ preserves the syntactic secrecy of $M$, if for all plain processes $I$ where $\text{fn}(I) \cap \text{bn}(P) = \emptyset$, there is no evaluation context $C[\_]$ with channel $c \notin \text{bn}(C)$ and process $R$ such that $P \parallel I \rightarrow^* C[\overline{c}\langle M\rangle.R]$. 
Example: Handshake Protocol

\[ S \xrightarrow{\text{new } k} I \xrightarrow{\text{new } s} C \]

- P does not preserve the secrecy of s

\[ C[\_] \triangleq \nu sk_S.\nu sk_C.\nu s.(_\mid !P_S \mid !P_c) \]

\[ P \equiv C[\overline{c}\langle pk(\text{sk}_S)\rangle \mid \overline{c}\langle pk(\text{sk}_C)\rangle \mid P_S \mid P_c] \]
Example: Handshake Protocol

\[ I \triangleq c(y_{pk}).c(\text{pk}(sk_M)).c(x). \]
\[ \quad \overline{c}\langle\text{aenc}(y_{pk}, \text{adec}(sk_M, x))\rangle.c(z). \]
\[ \quad \overline{c}\langle\text{snd}(\text{sdec}(\text{getmsg}(\text{adec}(sk_M, x), z)))\rangle \]

- P | I can evolve to a process that can output s on a public channel (lead to \( \overline{c}\langle s \rangle \))

\[ P \mid I \equiv C[\overline{c}\langle\text{pk}(sk_S)\rangle \mid\overline{c}\langle\text{pk}(sk_C)\rangle \mid c(x_{pk}) \nu k.\overline{c}\langle\text{aenc}(x_{pk}, \text{sign}(sk_S, k))\rangle. \]
\[ \quad c(z).\text{if} \text{fst}(\text{sdec}(k, z)) = \text{tag} \text{ then } Q \]
\[ \quad | c(y).\text{if checksign}(\text{pk}(sk_S), \text{adec}(sk_C, y)) = \text{true} \text{ then } \overline{c}\langle\text{senc}(\text{getmsg}(\text{adec}(sk_C, y)), \text{pair}(\text{tag}, s))\rangle \]
\[ \quad | I ] \]
Correspondence Properties

- A correspondence property asserts if event $f$ has been executed then the event $g$ must have been previously executed and any relationship between the event parameters must be satisfied.

A *correspondence property* is a formula of the form:

$$\bar{f}(M) \leadsto \bar{g}(N).$$
Example: Annotated Handshake Protocol

\[
P \triangleq \nu sk_S.\nu sk_C.\nu s. \\
\text{let } pk_S = pk(sk_S) \text{ in let } pk_C = pk(sk_C) \text{ in} \\
(\overline{c}(pk_S) \mid \overline{c}(pk_C) \mid !P_S \mid !P_C)
\]

\[
P_S \triangleq c(x\_pk).\nu k.\overline{\text{started}}S(\text{pair}(x\_pk, k)) \\
\overline{c}(\text{aenc}(x\_pk, \text{sign}(sk_S, k))). \\
c(z).\text{if } \text{fst}(\text{sdec}(k, z)) = \text{tag} \text{ then} \\
\overline{\text{completed}}S(\text{pair}(k, \text{eq}(x\_pk, pk_C))).Q
\]

\[
P_C \triangleq c(y).\text{let } y' = \text{adec}(sk_C, y) \text{ in let } y\_k = \text{getmsg}(y') \text{ in} \\
\overline{\text{started}}C(y\_k) \\
\text{if } \text{checksign}(pk_S, y') = \text{true} \text{ then} \\
\overline{c}(\text{senc}(y\_k, \text{pair}(\text{tag}, s)))). \\
\overline{\text{completed}}C(\text{pair}(pk_C, y\_k))
\]
Example: Annotated Handshake Protocol

- Authentication of $S$
  \[
  \overline{\text{completed}}^S\langle \text{pair}(y, \text{true}) \rangle \leadsto \overline{\text{started}}^C\langle y \rangle
  \]

- Authentication of $C$
  \[
  \overline{\text{completed}}^C\langle \text{pair}(x, y) \rangle \leadsto \overline{\text{started}}^S\langle \text{pair}(x, y) \rangle
  \]

- This correspondence property is not valid
Equivalence Properties

- Equivalence defines indistinguishability between two processes and allows us to consider properties that cannot be expressed as secrecy or correspondence properties.
- Example: Privacy in E-Voting

A voting protocol respects **privacy** if

\[ S[V_A^{a/v} | V_B^{b/v}] \approx S[V_A^{b/v} | V_B^{a/v}] \]
Five Steps to Verification

1 - Write equations to capture cryptographic primitives
2 - Decide which participants are honest/dishonest
3 - Model the honest parties as processes
4 - Model the intended security property
   - as a reachability property
   - as a correspondence property
   - as an equivalence property
5 - Evaluate the complete model using ProVerif and/or hand reasoning
References

- Mark D. Ryan and Ben Smyth, Applied pi calculus
- Martin Abadi, Security Protocols: Principles and Calculi