Topics in Concurrency

Lecture 10

Jonathan Hayman

6 March 2015

Public-key cryptography

Public key cryptography:

- for each entity/participant/agent A, there is a key $Pub(\mathrm{A})$ and a key $Priv(\mathrm{A})$.
- $Pub(\mathrm{A})$ is intended to be known by everybody: it is public
- $Priv(\mathrm{A})$ is intended to be known only by A: it is private
- Any agent can encrypt using a key that it knows
- To decrypt a message encrypted under $Pub(\mathrm{A})$ it is necessary to know $Priv(\mathrm{A})$
- To decrypt a message encrypted under $Priv(\mathrm{A})$ it is necessary to know $Pub(\mathrm{A})$

Will also allow symmetric keys e.g. $Key(\mathrm{A}, B)$.

The Needham-Schröder-Lowe Protocol

The goal of the NSL protocol: two agents use public-key cryptography to ensure

- **authentication**: For A as the initiator: upon completion of the protocol, A can be be sure that B generated the messages that A received following the protocol in response to A's request
- **shared secret**: if two entities complete the protocol with each other, at the end they both know a value not known to any potential attacker (e.g. to be used in more efficient symmetric-key cryptographic operations)

Formally, the correctness properties are subtle (e.g. what if B chose to release its private key?)

The original protocol

$$
\begin{array}{lll}
(1) & A \longrightarrow B: & \{m, A\}_{Pub(B)} \\
(2) & B \longrightarrow A: & \{m, n\}_{Pub(A)} \\
(3) & A \longrightarrow B: & \{n\}_{Pub(B)}
\end{array}
$$

m and n are nonces: fresh randomly-generated (very) long integers

The original protocol

$$
\begin{array}{lll}
(1) & A \longrightarrow B: & \{m, A\}_{Pub(B)} \\
(2) & B \longrightarrow A: & \{m, n\}_{Pub(A)} \\
(3) & A \longrightarrow B: & \{n\}_{Pub(B)}
\end{array}
$$

m and n are nonces: fresh randomly-generated (very) long integers

Original protocol introduced by Needham and Schröder in 1978 contained a flaw revealed (and fixed) by Lowe in 1995 [using CSP].

An attack against the original protocol

Man-in-the-middle attacker E convinces A to start communication with E and uses the messages generated by A to follow the protocol with B, posing as A.

A E B

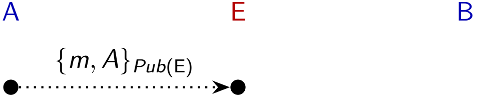
$\mathsf{A} \longrightarrow \mathsf{B}: \; \{m, \mathsf{A}\}_{Pub(B)}$

$\mathsf{B} \longrightarrow \mathsf{A}: \; \{m, n\}_{Pub(A)}$

$\mathsf{A} \longrightarrow \mathsf{B}: \; \{n\}_{Pub(B)}$

An attack against the original protocol

Man-in-the-middle attacker E convinces A to start communication with E and uses the messages generated by A to follow the protocol with B, posing as A.



$A \longrightarrow B : \{m, A\}_{Pub(B)}$

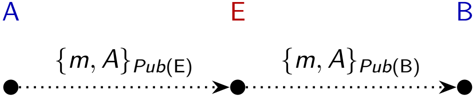
$B \longrightarrow A : \{m, n\}_{Pub(A)}$

$A \longrightarrow B : \{n\}_{Pub(B)}$

An attack against the original protocol

Man-in-the-middle attacker E convinces A to start communication with E and uses the messages generated by A to follow the protocol with B, posing as A.

$$\mathsf{A} \longrightarrow \mathsf{B} : \ \{m, \mathsf{A}\}_{Pub(B)}$$



$$\mathsf{B} \longrightarrow \mathsf{A} : \ \{m, n\}_{Pub(A)}$$

$$\mathsf{A} \longrightarrow \mathsf{B} : \ \{n\}_{Pub(B)}$$

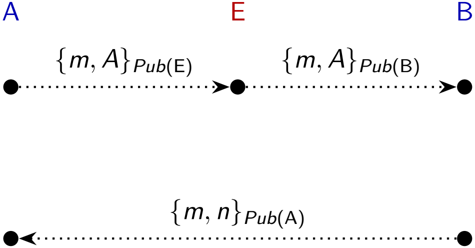
An attack against the original protocol

Man-in-the-middle attacker E convinces A to start communication with E and uses the messages generated by A to follow the protocol with B, posing as A.

$\mathsf{A} \longrightarrow \mathsf{B} : \; \{m, \mathsf{A}\}_{Pub(B)}$

$\mathsf{B} \longrightarrow \mathsf{A} : \; \{m, n\}_{Pub(A)}$

$\mathsf{A} \longrightarrow \mathsf{B} : \; \{n\}_{Pub(B)}$



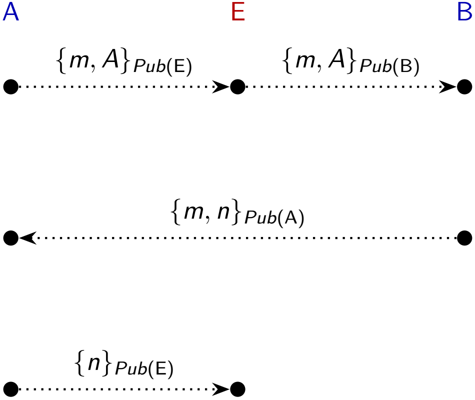
An attack against the original protocol

Man-in-the-middle attacker E convinces A to start communication with E and uses the messages generated by A to follow the protocol with B, posing as A.

$\mathsf{A} \longrightarrow \mathsf{B} : \ \{m, \mathsf{A}\}_{Pub(B)}$

$\mathsf{B} \longrightarrow \mathsf{A} : \ \{m, n\}_{Pub(A)}$

$\mathsf{A} \longrightarrow \mathsf{B} : \ \{n\}_{Pub(B)}$



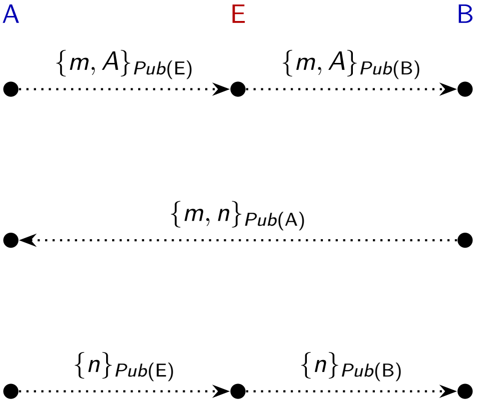
An attack against the original protocol

Man-in-the-middle attacker E convinces A to start communication with E and uses the messages generated by A to follow the protocol with B, posing as A.

$\mathsf{A} \longrightarrow \mathsf{B} : \{m, \mathsf{A}\}_{Pub(B)}$

$\mathsf{B} \longrightarrow \mathsf{A} : \{m, n\}_{Pub(A)}$

$\mathsf{A} \longrightarrow \mathsf{B} : \{n\}_{Pub(B)}$



The fixed protocol

$$
\begin{array}{lll}
(1) & \mathsf{A} \longrightarrow \mathsf{B}: & \{m, \mathsf{A}\}_{Pub(B)} \\
(2) & \mathsf{B} \longrightarrow \mathsf{A}: & \{m, n, \mathsf{B}\}_{Pub(A)} \\
(3) & \mathsf{A} \longrightarrow \mathsf{B}: & \{n\}_{Pub(B)}
\end{array}
$$

- Only B can decrypt the message sent in (1)
- A knows that only B can have sent the message in (2)
- B knows that only A can have sent the message in (1)
- the nonces m and n are shared secrets

But these properties are informal and approximate, and we've only described what's *supposed* to happen ...

SPL

Security Protocol Language

- One of a range of languages and models for analyzing crypto-protocols
- Others include Spi calculus, strand spaces
- Supports reasoning based on events (vs transitions)

- Asynchronous communication
- Messages persist on network
- New-name generation on output
- Input pattern-matches messages on network

Syntax

- We take an infinite set of names

$$\mathbf{Names} = \{m, n, \ldots, \mathrm{A}, \mathrm{B}, \ldots\}$$

$m, n, \ldots$ stand for for nonces, A, B agent identifiers

- with name variables inside the language

$$x, y, \ldots, X, Y$$

X, Y tend to range over agent identifiers, x, y over nonces

- The language also contains message variables

$$\psi, \psi', \psi_1, \ldots$$

- Indices shall be used to identify components of parallel compositions

$$i \in \mathbf{Indices}$$

SPL syntax

Name expressions	$v ::= n \mid A \mid \ldots \mid x \mid X$
Key expressions	$K ::= Pub(v) \mid Priv(v) \mid Key(v, v')$
Messages	$M ::= \psi \mid v \mid k \mid M_1, M_2 \mid \{M\}_k$
Processes	$p ::= \quad \text{out new } \vec{x}\, M.p \;\mid\; \text{in pat } \vec{x}, \vec{\psi}\, M.p \;\mid\; \|_{i \in I}\, p_i$

- out new $\vec{x}\, M.p$ generates a new vector of nonce values $\vec{n}$, using them for $\vec{x}$ to output $M[\vec{n}/\vec{x}]$ before resuming as $p[\vec{n}/\vec{x}]$
- input uses pattern matching...

Messages as patterns

Messages can contain variables.

$$\psi \qquad n, x \qquad \{m, y, \mathsf{B}\}_{Pub(\mathsf{A})}$$

These are used to perform matching.

Examples:

- match $\{\mathsf{A}, \mathsf{B}\}_{Pub(A)}$ against the pattern ψ
- match $\{m, n, \mathsf{B}\}_{Pub(\mathsf{A})}$ against the pattern $\{m, x, Y\}_{Pub(\mathsf{A})}$
- match $m, (n, \mathsf{A})$ against the pattern n, x where $m \neq n$

Messages as patterns

Messages can contain variables.

$$\psi \qquad n, x \qquad \{m, y, \mathsf{B}\}_{Pub(\mathsf{A})}$$

These are used to perform matching.

Examples:

- match $\{\mathsf{A}, \mathsf{B}\}_{Pub(A)}$ against the pattern ψ $\qquad \psi \mapsto \{\mathsf{A}, \mathsf{B}\}_{Pub(A)}$
- match $\{m, n, \mathsf{B}\}_{Pub(\mathsf{A})}$ against the pattern $\{m, x, Y\}_{Pub(\mathsf{A})}$
- match $m, (n, \mathsf{A})$ against the pattern n, x where $m \neq n$

Messages as patterns

Messages can contain variables.

$$\psi \qquad n, x \qquad \{m, y, \mathsf{B}\}_{Pub(\mathsf{A})}$$

These are used to perform matching.

Examples:

- match $\{\mathsf{A}, \mathsf{B}\}_{Pub(A)}$ against the pattern ψ $\qquad \psi \mapsto \{\mathsf{A}, \mathsf{B}\}_{Pub(A)}$
- match $\{m, n, \mathsf{B}\}_{Pub(\mathsf{A})}$ against the pattern $\{m, x, Y\}_{Pub(\mathsf{A})}$
  $x \mapsto n, Y \mapsto \mathsf{B}$
- match $m, (n, \mathsf{A})$ against the pattern n, x where $m \neq n$

Messages as patterns

Messages can contain variables.

$$\psi \qquad n, x \qquad \{m, y, \mathsf{B}\}_{Pub(\mathsf{A})}$$

These are used to perform matching.

Examples:

- match $\{\mathsf{A}, \mathsf{B}\}_{Pub(A)}$ against the pattern ψ $\qquad$ $\psi \mapsto \{\mathsf{A}, \mathsf{B}\}_{Pub(A)}$
- match $\{m, n, \mathsf{B}\}_{Pub(\mathsf{A})}$ against the pattern $\{m, x, Y\}_{Pub(\mathsf{A})}$
  $x \mapsto n, Y \mapsto \mathsf{B}$
- match $m, (n, \mathsf{A})$ against the pattern n, x where $m \neq n$ $\qquad$ no match

Conventions

- Messages with no free variables are closed.
- The input in pat $\vec{x}, \vec{\psi}\ M.p$ binds the variables $\vec{x}, \vec{\psi}$ in M and p, attempting to match the pattern M against any closed message that has been output to the network
- The output out new $\vec{x}\ M.p$ binds the variables $\vec{x}$ in M and p
- A process with no unbound (free) variables is closed.

We write:

- out $M.p$ where the list of new variables is empty
- in $M.p$ where the lists of name and message variables are precisely the free name and message variables in M
- *nil* is the empty parallel composition, which may be freely omitted
- use infix notation for finite parallel composition: $p_1 \parallel p_2$ is $\|_{i \in \{1,2\}}\ p_i$
- replication of a process $!p$ is $\|_{i \in \omega}\ p$

Names and variables

- A closed process can contain names: nonce values n or real agent identifiers A
- Variables are not names
- The set of all names in a process term is $\mathrm{names}(p)$ and in a message is $\mathrm{names}(M)$
- Example:

$$\mathrm{names}(\mathsf{out}\,\mathsf{new}\ x\{n, x\}_{Pub(\mathsf{A})}) = \{n, \mathsf{A}\}$$

The NSL protocol in SPL

The initiator initiator of the protocol is parameterized by the identity of the initiator and their intended participant:

$$
\begin{aligned}
\mathit{Init}(\mathrm{A}, \mathrm{B}) \quad \equiv \quad & \mathsf{out}\ \mathsf{new}\ x\ \{x, \mathrm{A}\}_{Pub(\mathrm{B})}. \\
& \mathsf{in}\ \{x, y, B\}_{Pub(A)}. \\
& \mathsf{out}\ \{y\}_{Pub(B)}
\end{aligned}
$$

The responder:

$$
\begin{aligned}
\mathit{Resp}(\mathrm{B}) \quad \equiv \quad & \mathsf{in}\ \{x, Z\}_{Pub(\mathrm{B})}. \\
& \mathsf{out}\ \mathsf{new}\ y\ \{x, y, \mathrm{B}\}_{Pub(Z)}. \\
& \mathsf{in}\ \{y\}_{Pub(\mathrm{B})}
\end{aligned}
$$

Dolev-Yao assumptions

We can program various forms of attacker process. Viewing messages as persisting once output to the network, they output new messages built from existing ones.

$$
\begin{aligned}
\mathit{Spy}_1 &\equiv \mathsf{in}\ \psi_1.\mathsf{in}\ \psi_2.\mathsf{out}\,(\psi_1, \psi_2) \\
\mathit{Spy}_2 &\equiv \mathsf{in}\ (\psi_1, \psi_2).\mathsf{out}\,\psi_1.\mathsf{out}\,\psi_2 \\
\mathit{Spy}_3 &\equiv \mathsf{in}\ X.\mathsf{in}\ \psi.\mathsf{out}\,\{\psi\}_{\mathit{Pub}(X)} \\
\mathit{Spy}_4 &\equiv \mathsf{in}\ \mathit{Priv}(X).\mathsf{in}\ \{\psi\}_{\mathit{Pub}(X)}.\mathsf{out}\,\psi
\end{aligned}
$$

$$
\mathit{Spy} \equiv \|_{i\in\{1,2,3,4\}}\ \mathit{Spy}_i
$$

The NSL system

We reason about concurrent runs of the protocol in parallel with ω-copies of the attacker.

$$\begin{aligned} P_{spy} &\equiv\ !Spy \\ P_{init} &\equiv \mathop{\Big|\Big|}_{\mathsf{A},\mathsf{B}\in\textbf{Agents}} !Init(\mathsf{A},\mathsf{B}) \\ P_{resp} &\equiv \mathop{\Big|\Big|}_{\mathsf{A}\in\textbf{Agents}} !Resp(\mathsf{A}) \end{aligned}$$

Messages from one run of the protocol can be used by the attacker against another run of the protocol.

$$NSL \equiv \mathop{\Big|\Big|}_{i\in\{resp,init,spy\}} P_i$$

Operational semantics

- A configuration is a tuple

$$\langle p, s, t \rangle$$

  - p is a closed process term
  - s is a finite subset of names: the names already in use
  - t is a subset of closed messages: the messages that have been output to the network
- Proper configurations:
  1. $\mathrm{names}(p) \subseteq s$
  2. $\mathsf{A} \in s$ for every agent identifier A
  3. $\bigcup\{\mathrm{names}(M) \mid M \in t\} \subseteq s$
- Transitions are labelled with actions

$$\alpha ::= \mathsf{out\,new}\ \vec{n}\, M \mid \mathsf{in}\ M \mid i : \alpha$$

Operational semantics

- **Output:** if $\vec{n}$ all distinct and not in s

$$\langle \mathrm{out\,new}\ \vec{x}\, M.p, s, t\rangle \xrightarrow{\mathrm{out\,new}\ \vec{n}\, M[\vec{n}/\vec{x}]} \langle p[\vec{n}/\vec{x}], s \cup \{\vec{n}\}, t \cup \{M[\vec{n}/\vec{x}]\}\rangle$$

- **Input:** if $M[\vec{n}/\vec{x}][\vec{N}/\vec{\psi}] \in t$

$$\langle \mathrm{in\ pat}\ \vec{x}, \vec{\psi}\ M.p, s, t\rangle \xrightarrow{\mathrm{in}\ M[\vec{n}/\vec{x}][\vec{N}/\vec{\psi}]} \langle p[\vec{n}/\vec{x}][\vec{N}/\vec{\psi}], s, t\rangle$$

- **Parallel:**

$$\frac{\langle p_j, s, t\rangle \xrightarrow{\alpha} \langle p'_j, s', t'\rangle \quad j \in I}{\langle \|_{i\in I}\ p_i, s, t\rangle \xrightarrow{j:\alpha} \langle \|_{i\in I}\ p'_i, s', t'\rangle}$$

where $p'_i = p_i$ for $j \neq i$

Reasoning from the transition semantics

Secrecy of the responder's nonce:
Suppose *Priv*(A) and *Priv*(B) do not occur as the contents of any message in t_0. For all runs

$$\langle NSL, s_0, t_0\rangle \xrightarrow{\alpha_1} \ldots \langle p_{r-1}, s_{r-1}, t_{r-1}\rangle \xrightarrow{\alpha_r} \ldots$$

where $\langle NSL, s_0, t_0\rangle$ is proper, if α_r has the form $resp : B : j : \mathrm{out}\,\mathrm{new}\; n\{m, n, \mathrm{B}\}_{Pub(A)}$, then $n \not\in t_l$ for any $l \in \omega$.

Proof idea: strengthen hypothesis, prove by induction / assume earliest violation.

The model obscures the key reasoning technique: that a violation must be by an event that causally depends (either through input/output or control) on an earlier event that violates the invariant.

$\rightsquigarrow$ a Petri net semantics for SPL

Petri net semantics of SPL

A net with persistent conditions representing all of SPL (not just particular processes at first).

Conditions viewed as being: control, network and name

- Control conditions form a set **C** of capacity-1 conditions

$$b ::= \mathrm{out\,new}\ \vec{x}\ M.p \mid \mathrm{in\ pat}\ \vec{x}, \vec{\psi}\ M.p \mid i : b$$

  the control state of each thread

- Network conditions: form a set **O** of persistent conditions

$$\mathbf{O} = \{\text{closed messages}\}$$

  the messages already output

- Name conditions: form a set **S** of capacity-1 conditions

$$\mathbf{S} = \mathbf{Names}$$

  the names in use

Control conditions

For a process p, the subset of control conditions

$$Ic(p)$$

is called its initial conditions.

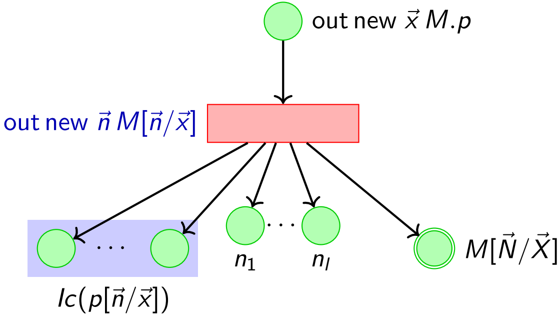
$$\begin{aligned} Ic(\text{out new } \vec{x}\, M.p) &= \text{out new } \vec{x}\, M.p \\ Ic(\text{in pat } \vec{x}, \vec{\psi}\ M.p) &= \text{in pat } \vec{x}, \vec{\psi}\ M.p \\ Ic(\|_{i \in I}\ p_i) &= \bigcup_{i \in I} i : Ic(p) \end{aligned}$$

where $i : C = \{i : b \mid b \in C\}$ for $C \subseteq \mathbf{C}$.

The events of SPL: output

The set **Events** includes:

if $\text{out}\,\text{new}\ \vec{x}\ M.p$ is a closed term and $\vec{n} = n_1, \ldots, n_l$ are distinct names to match $\vec{x} = x_1, \ldots, x_l$

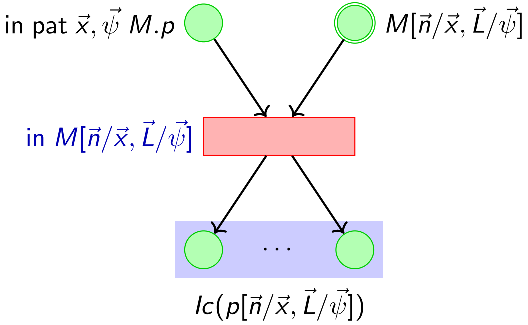


Events are labelled with an action.

The events of SPL: input

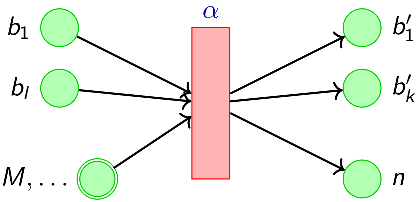
The set **Events** includes:

if in pat $\vec{x}, \vec{\psi}\ M.p$ is a closed term and $\vec{n} = n_1, \ldots, n_l$ are distinct names to match $\vec{x} = x_1, \ldots, x_l$ and $\vec{L} = L_1, \ldots, L_k$ are messages to match $\vec{\psi} = \psi_1, \ldots, \psi_k$



The events of SPL: tags

If e.g. there is an event



then there is an event