Encoding security protocols in the cryptographic $\lambda$-calculus

Eijiro Sumii
Joint work with Benjamin Pierce
University of Pennsylvania
An obvious fact

- Security is important
- Cryptography is a major way to achieve security
- Therefore, cryptography is important
A less obvious fact

- There are nice cryptosystems like RSA, 3DES, etc.
- ...but how to use them is often non-trivial
Example: Needham-Schroeder public-key protocol [NS78]

- Assumption: all encryption keys and the network are public
- Purpose: principals A and B authenticate each other, and exchange two secret nonces

A → B: \{ A, Na \}_{K_b}
B → A: \{ Na, Nb \}_{K_a}
A → B: \{ Nb \}_{K_b}
An attack on the protocol [Lowe 95]

- If some B is malicious (say, E), it can impersonate A and fool another B

\[
\begin{align*}
A &\rightarrow E: \{ A, Na \}_{Ke} \\
E(A) &\rightarrow B: \{ A, Na \}_{Kb} \\
B &\rightarrow E(A): \{ Na, Nb \}_{Ka} \\
E &\rightarrow A: \{ Na, Nb \}_{Ka} \\
A &\rightarrow E: \{ Nb \}_{Ke} \\
E(A) &\rightarrow B: \{ Nb \}_{Kb}
\end{align*}
\]

N.B. ( ) means forgery or interception of a message.
A fix [Lowe 95]

\[ A \rightarrow B: \{ A, Na \} \_{K_b} \]

\[ B \rightarrow A: \{ Na, Nb, B \} \_{K_a} \]

\[ A \rightarrow B: \{ Nb \} \_{K_b} \]
How does it prevent the attack?

A → E: \{ A, Na \}_Ke

E(A) → B: \{ A, Na \}_Kb

B → E(A): \{ Na, Nb, B \}_Ka

E → A: \{ Na, Nb, B \}_Ka

(* Here, A asserts E = B, which is false *)

A → E: \{ Nb \}_Ke

E(A) → B: \{ Nb \}_Kb
So what?

- We want a way to specify and verify security protocols
- But informal notation is too ambiguous
  (It is often unclear how each principal reacts to various messages)
- So we need a formal model

\[\lambda\text{-calculus} \, + \, \text{cryptographic primitives}\]
Why $\lambda$-calculus?
(not $\pi$-calculus, for example)

- It's simple and high-level
- It's standard and well-studied
  - For instance, logical relations help to prove various properties, such as contextual equivalence (cf. [Mitchell 96, Chapter 8])
    - Equivalences in process calculi are hard to prove! (e.g. [Abadi & Gordon 96])
- It's actually (almost) expressive enough to model various protocols and attacks
The cryptographic $\lambda$-calculus

Simply-typed call-by-value $\lambda$-calculus +
shared-key cryptographic primitives

$e ::= \ldots \mid k \mid \text{new } x \text{ in } e \mid \{e_1\}_{e_2}$
$\mid \lambda \{x\}_{e_1}. e_2$

$\tau ::= \ldots \mid \text{key} \mid \text{bits}(\tau)$
$(\lambda \{x\}_k. e) \{v\}_k \rightarrow e[v/x]$

Subsumes public-key cryptography

$k^+ \equiv \lambda z. \{z\}_k \quad k^- \equiv \lambda \{z\}_k. z$
Encoding protocols

- configuration = record (or tuple) of principals and public keys
- principal = function from messages to messages with a continuation (of the principal itself)
- sound network and scheduler = context applying "right" principals to right messages in a right order
- malicious attacker = arbitrary context
new Ka in new Kb in new Ke in
{
  A = ..., \\
  B = ..., \\
  Ka^+ = \lambda z. \{z\}_ka, Kb^+ = \lambda z. \{z\}_kb, Ke = Ke
}
new Ka in new Kb in new Ke in
{  A = new Na in
    send { "A", Na }_{Kb} to B in ...

B = ...
Ka^+ = \lambda z.\{z\}_ka, Kb^+ = \lambda z.\{z\}_kb, Ke = Ke }

Encoding Needham-Schroeder
Encoding Needham-Schroeder

new Ka in new Kb in new Ke in
{  A = new Na in
    send { "A", Na }_{Kb} to B in ..., 
B = receive { "A", Na }_{Kb} in 
    new Nb in
    send { Na, Nb }_{Ka} to A in ..., 
Ka^{+} = \lambda z.\{z\}_{ka}, \ Kb^{+} = \lambda z.\{z\}_{kb}, \ Ke = Ke \}
new Ka in new Kb in new Ke in 
{   A = new Na in 
    send { "A", Na }\textsubscript{Kb} to B in 
    receive { Na', Nb }\textsubscript{Ka} in 
    assert Na = Na' in 
    send { Nb }\textsubscript{Kb} to B in ...,
    B = receive { "A", Na }\textsubscript{Kb} in 
    new Nb in 
    send { Na, Nb }\textsubscript{Ka} to A in ...,
    Ka^+ = \lambda z.\{z\}\textsubscript{ka}, Kb^+ = \lambda z.\{z\}\textsubscript{kb}, Ke = Ke }
Encoding Needham-Schroeder

\[
\begin{align*}
\text{new } Ka & \text{ in new } Kb \text{ in new } Ke \text{ in } \\
\{ & \quad \text{new } Na \text{ in } \\
& \quad ("B", \{ "A", Na \} Kb, \\
& \quad \lambda \{ Na', Nb \} Ka. \\
& \quad \text{if } Na' \neq Na \text{ then } \bot \text{ else } \\
& \quad ("B", \{ Nb \} Kb, \ldots)), \\
B & = \lambda \{ "A", Na \} Kb. \\
& \quad \text{new } Nb \text{ in } \\
& \quad ("A", \{ Na, Nb \} Ka, \ldots), \\
Ka^+ & = \lambda z. \{z\}_k a, \\
Kb^+ & = \lambda z. \{z\}_k b, \\
& \quad \text{Ke } = \text{ Ke } \\
\end{align*}
\]

send \( m \) to \( X \) in \( c \) \\
\( \Rightarrow ("X", m, c) \)

receive \( m \) in \( c \) \\
\( \Rightarrow \lambda m. c \)
Encoding Needham-Schroeder

new Ka in new Kb in new Ke in
{   A = λn. let Kn = lookup n in
    new Na in
    (n, { "A", Na }_{Kn},
    λ{ Na', Nn }_{Ka}.
    if Na' ≠ Na then ⊥ else
    (n, { Nn }_{Kn}, ...)),
B = λ{ "A", Na }_{Kb}.
    new Nb in
    ("A", { Na, Nb }_{Ka}, ...),
Ka⁺ = λz.{z}_{ka}, Kb⁺ = λz.{z}_{kb}, Ke = Ke }
Encoding the network and scheduler

"A context applying right principals to right messages in a right order"

\[
\text{Net}[r] = \\
\begin{align*}
\text{let } (_, m_1, c_A) &= \#_A(r) \text{ "B" in } \\
\text{let } (_, m_2, c_B) &= \#_B(r) m_1 \text{ in } \\
\text{let } (_, m_3, c_A') &= c_A m_2 \text{ in } ...
\end{align*}
\]
Encoding the attacker

\[
\text{Attack}[r] = \\
\text{let } K_e = \#_{K_e}(r) \text{ in} \\
\text{let } K_b^+ = \#_{K_b^+}(r) \text{ in} \\
\text{let } (_, \{ _, Na \}^-_{K_e}, c_A) = \#_A(r) \text{ "E" in} \\
\text{let } (_, m, c_B) = \#_B(r) K_b^+(A, Na) \text{ in} \\
\text{(* m becomes } \{ \text{Na, Nb } \}^-_{K_a} \text{ *)} \\
\text{let } (_, \{ Nb \}^-_{K_e}, c_A') = c_A m \text{ in ...} \\
\text{(* use Nb to trick B *)}
\]
Another example: ffgg protocol

An artificial protocol with a "necessarily parallel" attack

$$A \rightarrow B : A$$

$$B \rightarrow A : N_1, N_2$$

$$A \rightarrow B : A, \{ N_1, N_2, M \}_{Kb} \text{ as } \{N_1, X, Y\}_{Kb}$$

$$B \rightarrow A : N_1, X, \{ X, Y, N_1 \}_{Kb}$$
A "parallel" attack to the protocol

- B and B' are two concurrent processes for the same principal.
- ( ) means forgery or interception of a message by the attacker.

A → B : A
(A) → B' : A
B → (A) : N₁, N₂
B' → (A) : N₁', N₂'
(B) → A : N₁, N₁'
A → B : { N₁, N₁', M }ₖₐ
B → (A) : N₁, N₁', { N₁', M, N₁ }ₖₐ
(A) → B' : { N₁', M, N₁ }ₖₐ
B' → (A) : N₁', M, { M, N₁, N₁' }ₖₐ
Encoding ffgg

new Kb in
{   A = ("B", "A",
    λ(N₁, N₂).
    ("B", { N₁, N₂, M }ₖₒ₃, ...)),
    B = λn. new N₁ in new N₂ in
    (n, (N₁, N₂),
    λ{ N₁', X, Y }ₖₒ₃.
    if N₁' ≠ N₁ then ⊥ else
    (n, (N₁, X, { X, Y, N₁ }ₖₒ₃), ...)) }
Encoding the attacker

\[ \text{Attack}[r] = \]
\[
\begin{align*}
&\text{let } (_, (N_1, N_2), c_B) = \#_B(r) \text{ "A" in } \\
&\text{let } (_, (N_1', N_2'), c'_B) = \#_B(r) \text{ "A" in } \\
&\text{let } (_, m_A, _) = \#_A(r) (N_1, N_1') \text{ in } \\
&\hspace{1cm} (*) \text{ } m_A \text{ becomes } \{ N_1, N_1', M \}_{K_B} * \\
&\text{let } (_, (_, _, m_B), _) = c_B m_A \text{ in } \\
&\hspace{1cm} (*) \text{ } m_B \text{ becomes } \{ N_1', M, N_1 \}_{K_B} * \\
&\text{let } (_, (_, M, _), _) = c'_B m_B \text{ in } ... \\
&\hspace{1cm} (*) \text{ use } M \text{ for whatever *}
\end{align*}
\]
Secrecy $\approx$ non-interference $\approx$ contextual equivalence

Let $\text{NS}[i]$ be:

```
new ... in
{  A = ...
    receive \{ x \}_{Nn} in
    x \mod 2,
    B = ...
    send \{ i \}_{Nb} to A in
    ()
  ... }
```

Then, the secrecy of $i$ can be expressed as, say,

$\text{NS}[1] \approx \text{NS}[3]$
Using logical relation to prove contextual equivalence

\[ e \sim e' : \tau \implies e \simeq e' : \tau \]

"Logical relation implies contextual equivalence"

- Defined by induction on \( \tau \), and (hopefully) easier to prove
- Whole topic of another talk!
A drawback

- There is no "state" of principals
  - Some attacks might be bogus
    (i.e., impossible in reality)
  ⇒ Consider linear $\lambda$-calculus?