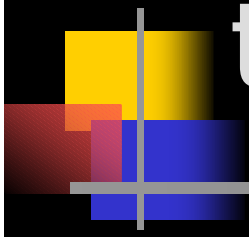
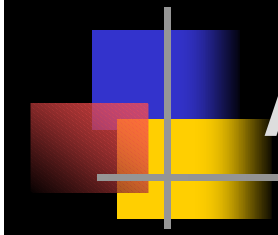


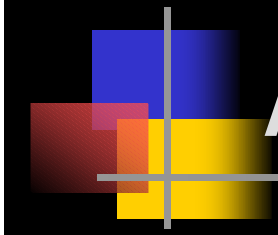
Encoding security protocols in the cryptographic λ-calculus

Eijiro Sumii
Joint work with Benjamin Pierce
University of Pennsylvania



An obvious fact

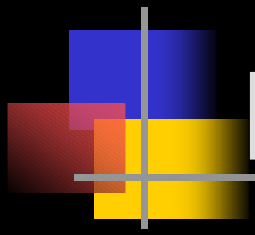
- Security is important
- Cryptography is a major way to achieve security
- Therefore, cryptography is important



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]



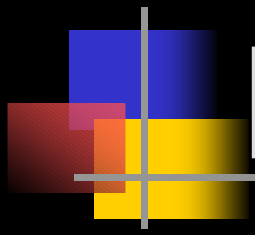
- Assumption: all encryption keys and the network are public
- Purpose: principals A and B authenticate each other, and exchange two secret nonces

$$A \rightarrow B: \{ A, Na \}_{Kb}$$

$$B \rightarrow A: \{ Na, Nb \}_{Ka}$$

$$A \rightarrow B: \{ Nb \}_{Kb}$$

An attack on the protocol [Lowe 95]



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

$A \rightarrow E: \{ A, Na \}_{Ke}$

$E(A) \rightarrow B: \{ A, Na \}_{Kb}$

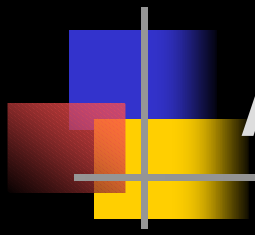
$B \rightarrow E(A): \{ Na, Nb \}_{Ka}$

$E \rightarrow A: \{ Na, Nb \}_{Ka}$

$A \rightarrow E: \{ Nb \}_{Ke}$

$E(A) \rightarrow B: \{ Nb \}_{Kb}$

N.B.
( ) means forgery or interception of a message

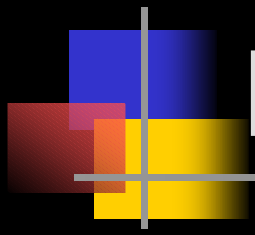


A fix [Lowe 95]

$A \rightarrow B$: $\{ A, Na \}_{Kb}$

$B \rightarrow A$: $\{ Na, Nb, B \}_{Ka}$

$A \rightarrow B$: $\{ Nb \}_{Kb}$



How does it prevent the attack?

$A \rightarrow E: \{ A, Na \}_{Ke}$

$E(A) \rightarrow B: \{ A, Na \}_{Kb}$

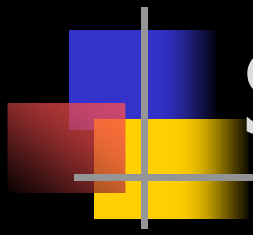
$B \rightarrow E(A): \{ Na, Nb, B \}_{Ka}$

$E \rightarrow A: \{ Na, Nb, B \}_{Ka}$

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

$A \rightarrow E: \{ Nb \}_{Ke}$

$E(A) \rightarrow B: \{ Nb \}_{Kb}$



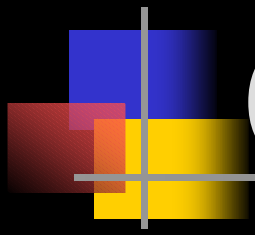
So what?

- 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

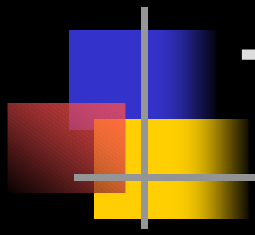
$\Downarrow$

λ-calculus + cryptographic primitives



Why λ-calculus? (not π-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 λ-calculus

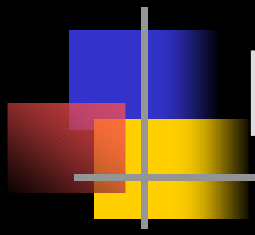
Simply-typed call-by-value λ-calculus + shared-key cryptographic primitives

- e ::= … | k | new x in e | $\{e1\}_{e2}$ | $\lambda\{x\}_{e1}$. e2
- τ ::= … | key | bits(τ)

$$(\lambda\{x\}_k.\ e)\ \{v\}_k \rightarrow e[v/x]$$

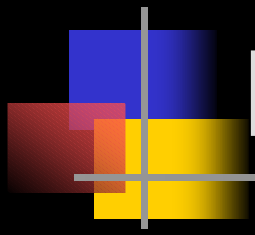
Subsumes public-key cryptography

$$k^+ \equiv \lambda z.\{z\}_k \qquad k^- \equiv \lambda\{z\}_k.z$$



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



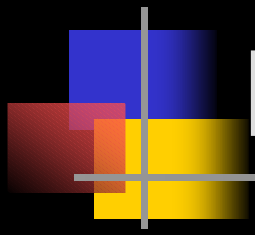
Encoding Needham-Schroeder

new Ka in new Kb in new Ke in

{ A = …,

B = …,

$Ka^{+} = \lambda z.\{z\}_{ka}$, $Kb^{+} = \lambda z.\{z\}_{kb}$, Ke = Ke }



Encoding Needham-Schroeder

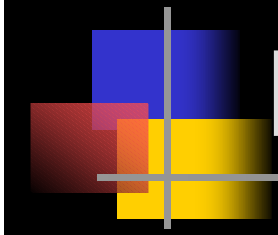
new Ka in new Kb in new Ke in

{ A = new Na in

send { "A", Na }$_{Kb}$ to B in …,

B = …,

Ka^{+} = λz.{z}$_{ka}$, Kb^{+} = λz.{z}$_{kb}$, Ke = Ke }



Encoding Needham-Schroeder

new Ka in new Kb in new Ke in

{ A = new Na in

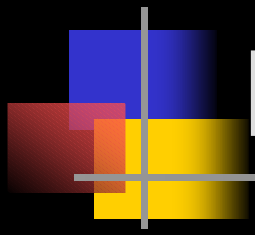
send { "A", Na }$_{Kb}$ to B in …,

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 }



Encoding Needham-Schroeder

new Ka in new Kb in new Ke in

{ A = new Na in

send { "A", Na }$_{Kb}$ to B in

receive { Na', Nb }$_{Ka}$ in

assert Na = Na' in

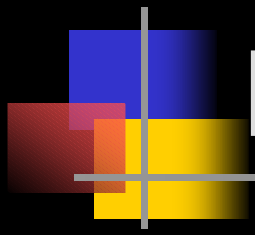
send { Nb }$_{Kb}$ to B in …,

B = receive { "A", Na }$_{Kb}$ in

new Nb in

send { Na, Nb }$_{Ka}$ to A in …,

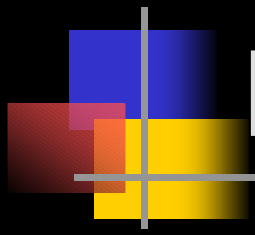
$Ka^{+} = \lambda z.\{z\}_{ka}$, $Kb^{+} = \lambda z.\{z\}_{kb}$, Ke = Ke }



Encoding Needham-Schroeder

send m to X in c
$\Rightarrow$ ("X", m, c)
receive m in c
$\Rightarrow$ λm. c

new Ka in new Kb in new Ke in
{ A = new Na in
("B", { "A", Na }$_{Kb}$,
λ{ Na', Nb }$_{Ka}$.
if Na' $\neq$ Na then $\perp$ else
("B", { Nb }$_{Kb}$, …)),
B = λ{ "A", Na }$_{Kb}$.
new Nb in
("A", { Na, Nb }$_{Ka}$, …),
$Ka^+ = \lambda z.\{z\}_{ka}$, $Kb^+ = \lambda z.\{z\}_{kb}$, Ke = Ke }

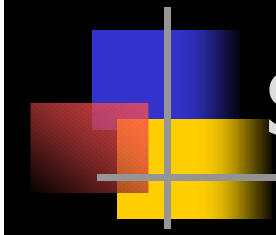


Encoding Needham-Schroeder

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' $\neq$ Na then $\perp$ 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



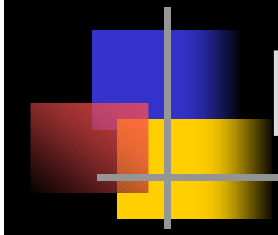
"A context applying right principals to right messages in a right order"

Net[r] =

let $(_, m_1, c_A) = \#_A(r)$ "B" in

let $(_, m_2, c_B) = \#_B(r)\ m_1$ in

let $(_, m_3, c_A') = c_A\ m_2$ in …



Encoding the attacker

Attack[r] =

let Ke = $\#_{Ke}(r)$ in

let Kb^{+} = $\#_{Kb+}(r)$ in

let (_, { _, Na }$_{Ke}$, c_A) = $\#_A(r)$ "E" in

let (_, m, c_B) = $\#_B(r)$ Kb^{+}(A, Na) in

(* m becomes { Na, Nb }$_{Ka}$ *)

let (_, { Nb }$_{Ke}$, c_A') = c_A m in …

(* use Nb to trick B *)

Another example: ffgg protocol

- An artificial protocol with a "necessarily parallel" attack

$A \rightarrow B : A$

$B \rightarrow A : N_1, N_2$

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

$B \rightarrow A : N_1, X, \{ X, Y, N_1 \}_{Kb}$

A "parallel" attack to the protocol

$A \rightarrow B : A$

$(A) \rightarrow B' : A$

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

$B' \rightarrow (A) : N_1', N_2'$

$(B) \rightarrow A : N_1, N_1'$

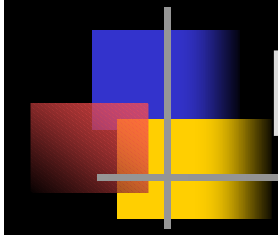
$A \rightarrow B : \{ N_1, N_1', M \}_{Kb}$

$B \rightarrow (A) : N_1, N_1', \{ N_1', M, N_1 \}_{Kb}$

$(A) \rightarrow B' : \{ N_1', M, N_1 \}_{Kb}$

$B' \rightarrow (A) : N_1', M, \{ M, N_1, N_1' \}_{Kb}$

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

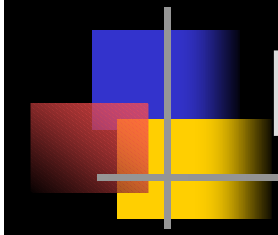


Encoding ffgg

new Kb in

{ A = ("B", "A",
$\lambda(N_1, N_2)$.
("B", { N_1, N_2, M }$_{Kb}$, …)),

B = λn. new N_1 in new N_2 in
(n, (N_1, N_2),
λ{ N_1', X, Y }$_{Kb}$.
if $N_1' \neq N_1$ then $\perp$ else
(n, (N_1, X, { X, Y, N_1 }$_{Kb}$), …)) }



Encoding the attacker

Attack[r] =

let (_, (N_1, N_2), c_B) = $\#_B$(r) "A" in

let (_, (N_1', N_2'), c_B') = $\#_B$(r) "A" in

let (_, m_A, _) = $\#_A$(r) (N_1, N_1') in

(* m_A becomes { N_1, N_1', M }$_{Kb}$ *)

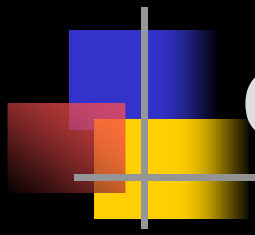
let (_, (_, _, m_B), _) = c_B m_A in

(* m_B becomes { N_1', M, N_1 }$_{Kb}$ *)

let (_, (_, M, _), _) = c_B' m_B in …

(* use M for whatever *)

Secrecy ≈ non-interference ≈ contextual equivalence



Let NS[i] be:

new … in

{ A = …

receive $\{ x \}_{Nn}$ in

x mod 2,

B = …

send $\{ i \}_{Nb}$ to A in

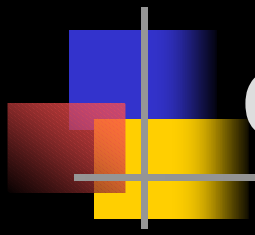
(),

… }

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

NS[1] ≈ NS[3]

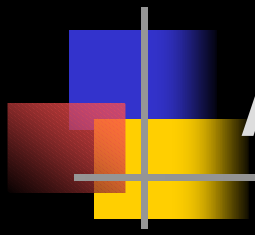
Using logical relation to prove contextual equivalence



$$\vdash e \sim e' : \tau \;\Rightarrow\; e \approx e' : \tau$$

"Logical relation implies contextual equivalence"

- Defined by induction on τ, 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)

  $\Rightarrow$ Consider linear λ-calculus?