Security II - Cryptographic Protocols

Stefano Calzavara

Università Ca' Foscari Venezia

April 23, 2020

Università
Ca'Foscari
Venezia

Introduction

Cryptographic protocols are the foundations of many distributed systems

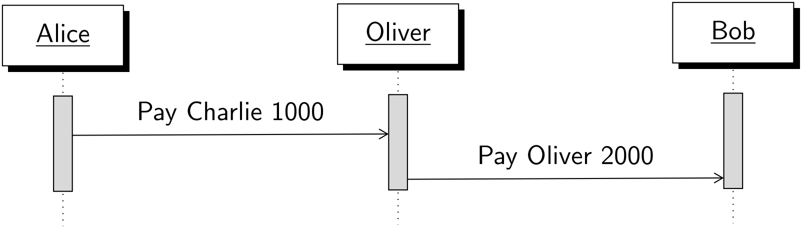
- SSL / TLS to establish secure channels on the Web
- Kerberos to authenticate network services
- WPA2 to securely connect to Wifi networks

Complicated to prove correct:

- conceptual flaws in the protocol design
- implementation mistakes, which make a correct protocol insecure
- (cryptographic breaches)

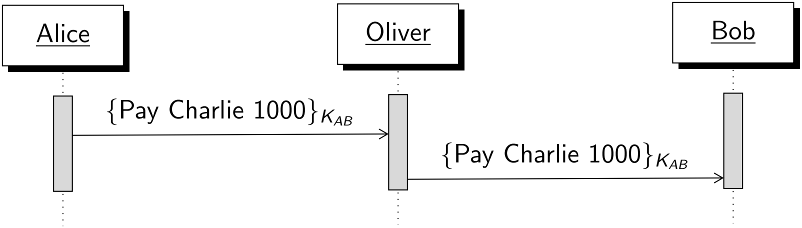
Threat Model

Protocol participants communicate on an untrusted network: everything sent on the network can be read and modified by the attacker



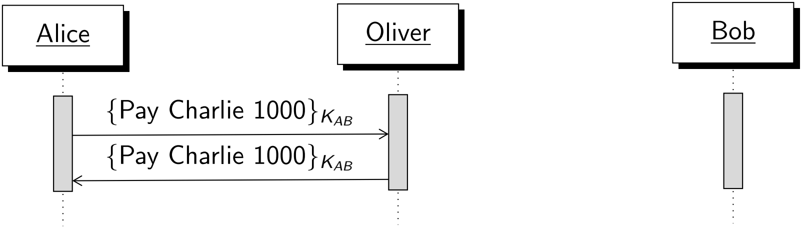
Cryptography

We assume the use of perfect cryptography, that the attacker cannot breach. Using symmetric crypto we can protect the exchange



Reflection Attack

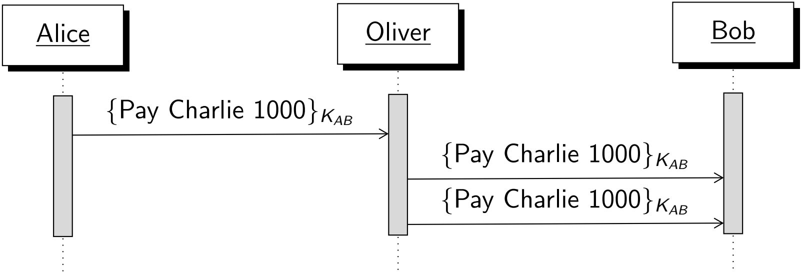
Unfortunately, perfect cryptography is not enough for security!



Solution: break symmetry by including the sender's name in the message

Replay Attack

Another example where perfect cryptography does not help...



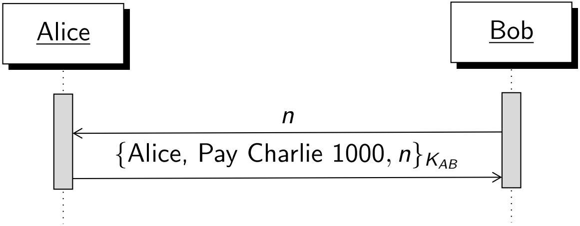
Solution: ensure freshness by including a timestamp / sequence number

Challenge - Response

Timestamps and sequence numbers are not great for freshness

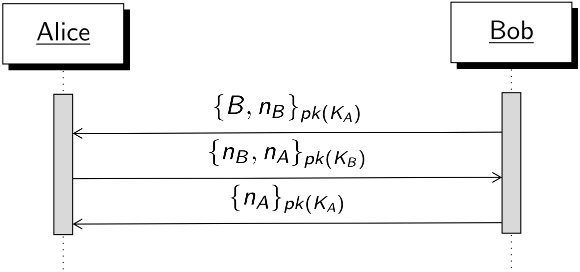
- timestamps require the use of a global clock (synchronization?)
- sequence numbers require the use of state information

Better solution: challenge-response protocols

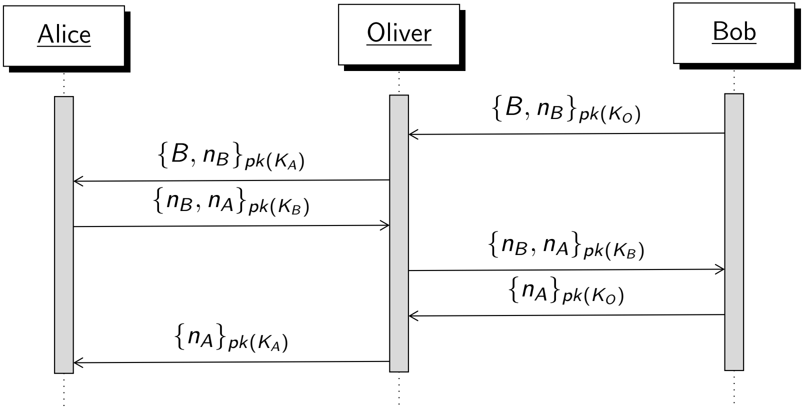


Example: Needham - Schroeder Protocol

Goal: exchange nonces n_A, n_B to generate a symmetric key

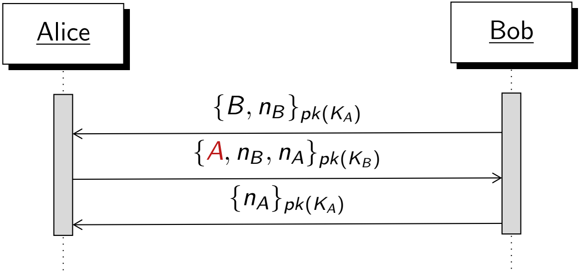


Breaking Needham - Schroeder



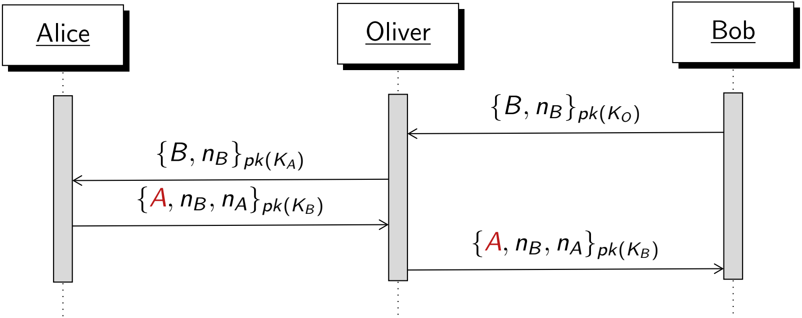
Fixing Needham - Schroeder

Fix (Lowe): extend the second message with Alice's identity



Fixing Needham - Schroeder

Now Bob can spot that something went wrong...



Protocol Verification

Manual analysis is long, tedious and very error-prone

- protocols run on distributed, concurrent systems
- ... which are supposed to satisfy complex security properties
- ... and are assumed to be under attack from the network

Luckily, there's great support for automated verification nowadays

1. encode the protocol in an appropriate formalism, e.g., process calculi
2. express the intended security properties in the chosen formalism
3. push the button and get the results of the security analysis

Process Calculi

Process calculus = tiny formalism to express distributed systems

Extensive literature in the area since 1980:

- 1980, CCS: focus on synchronization over channels
- 1989, pi-calculus: CCS + channel mobility
- 1997, spi-calculus: pi-calculus + simple cryptography
- 2001, applied pi-calculus: pi-calculus + arbitrary cryptography

CCS

Ordering a pizza in CCS:

$$
\begin{array}{lcl}
C & \triangleq & \overline{askpizza}.\overline{pay}.pizza \\
P & \triangleq & askpizza.pay.\overline{pizza} \\
S & \triangleq & C \mid P
\end{array}
$$

Small-step semantics:

$$
\begin{array}{lcl}
S & \rightarrow & \overline{pay}.pizza \mid pay.\overline{pizza} \\
 & \rightarrow & pizza \mid \overline{pizza} \\
 & \rightarrow & 0 \mid 0
\end{array}
$$

Value-Passing CCS

Hey, let me choose my pizza!

$$
\begin{array}{lcl}
C & \triangleq & \overline{askpizza}\langle margherita\rangle.\overline{pay}\langle 5\rangle.pizza(x) \\
P & \triangleq & askpizza(x).pay(y).\overline{pizza}\langle x\rangle \\
S & \triangleq & C \mid P
\end{array}
$$

Small-step semantics:

$$
\begin{array}{lcl}
S & \rightarrow & \overline{pay}\langle 5\rangle.pizza(x) \mid pay(y).\overline{pizza}\langle margherita\rangle \\
 & \rightarrow & pizza(x) \mid \overline{pizza}\langle margherita\rangle \\
 & \rightarrow & 0 \mid 0
\end{array}
$$

Non-Determinism

Multiple clients might induce confusion on pizza delivery...

$$
\begin{array}{lcl}
C_1 & \triangleq & \overline{askpizza}\langle margherita\rangle.pizza(x).\overline{eat_1}\langle x\rangle \\
C_2 & \triangleq & \overline{askpizza}\langle pepperoni\rangle.pizza(x).\overline{eat_2}\langle x\rangle \\
P & \triangleq & !askpizza(x).\overline{pizza}\langle x\rangle \\
S & \triangleq & C_1 \mid C_2 \mid P
\end{array}
$$

Small step semantics:

$$
\begin{array}{lll}
S & \rightarrow & pizza(x).\overline{eat_1}\langle x\rangle \mid \overline{pizza}\langle margherita\rangle \mid C_2 \mid P \\
 & \rightarrow & pizza(x).\overline{eat_1}\langle x\rangle \mid \overline{pizza}\langle margherita\rangle \mid \\
 & & pizza(x).\overline{eat_2}\langle x\rangle \mid \overline{pizza}\langle pepperoni\rangle \mid P \\
 & \rightarrow & \overline{eat_1}\langle pepperoni\rangle \mid \overline{pizza}\langle margherita\rangle \mid pizza(x).\overline{eat_2}\langle x\rangle \mid P \\
 & \rightarrow & \overline{eat_1}\langle pepperoni\rangle \mid \overline{eat_2}\langle margherita\rangle \mid P
\end{array}
$$

Pi-Calculus

Reliable home delivery of pizza!!!

$$
\begin{array}{lcl}
C_1 & \triangleq & (\nu h)\,(\overline{askpizza}\langle margherita, h\rangle.h(x).\overline{eat_1}\langle x\rangle) \\
C_2 & \triangleq & (\nu h)\,(\overline{askpizza}\langle pepperoni, h\rangle.h(x).\overline{eat_2}\langle x\rangle) \\
P & \triangleq & !askpizza(x, y).\overline{y}\langle x\rangle \\
S & \triangleq & C_1 \mid C_2 \mid P
\end{array}
$$

Small-step semantics:

$$
\begin{array}{lll}
S & \rightarrow & (\nu h)\,(h(x).\overline{eat_1}\langle x\rangle \mid \overline{h}\langle margherita\rangle) \mid C_2 \mid P \\
 & \rightarrow & (\nu h)\,(h(x).\overline{eat_1}\langle x\rangle \mid \overline{h}\langle margherita\rangle) \mid \\
 & & (\nu h)\,(h(x).\overline{eat_2}\langle x\rangle \mid \overline{h}\langle pepperoni\rangle) \mid P \\
 & \rightarrow & \overline{eat_1}\langle margherita\rangle \mid (\nu h)\,(h(x).\overline{eat_2}\langle x\rangle \mid \overline{h}\langle pepperoni\rangle) \mid P \\
 & \rightarrow & \overline{eat_1}\langle margherita\rangle \mid \overline{eat_2}\langle pepperoni\rangle \mid P
\end{array}
$$

Scope Extrusion

The restriction operator $(\nu a)\,P$ creates a fresh name a which is local to the scope of P

- scope extrusion extends the scope of a to other processes
- useful to model a selective release of secrets
- formalized via structural equivalence $\equiv$

$$
\begin{array}{rcl}
(\nu a)\,(\overline{c}\langle a\rangle.a(x).0) \mid c(x).\overline{x}\langle k\rangle.0 & \equiv & (\nu a)\,(\overline{c}\langle a\rangle.a(x).0 \mid c(x).\overline{x}\langle k\rangle.0) \\
& \rightarrow & (\nu a)\,(a(x).0 \mid \overline{a}\langle k\rangle.0) \\
& \rightarrow & (\nu a)\,(0 \mid 0) \\
& \equiv & 0
\end{array}
$$

Applied Pi-Calculus

The applied pi-calculus exchanges constructed terms on channels

$$
\begin{array}{llll}
\text{Terms} & M, N & ::= & x \mid c \mid f(M_1, \ldots, M_n) \\
\text{Processes} & P, Q & ::= & \overline{M}\langle N \rangle.P \\
 & & \mid & M(x).P \\
 & & \mid & 0 \\
 & & \mid & P \mid Q \\
 & & \mid & !P \\
 & & \mid & (\nu a)\, P \\
 & & \mid & \text{let } x = g(M_1, \ldots, M_n) \text{ in } P \text{ else } Q
\end{array}
$$

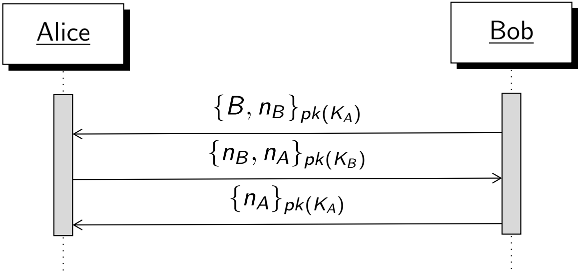
Equational Theory

Terms are subject to an equational theory which defines their semantics

$$
\begin{aligned}
\mathsf{fst}(\mathsf{pair}(M, N)) &= M \\
\mathsf{snd}(\mathsf{pair}(M, N)) &= N \\
\\
\mathsf{sdec}(\mathsf{senc}(M, N), N) &= M \\
\\
\mathsf{dec}(\mathsf{enc}(M, \mathsf{pk}(N)), N) &= M \\
\mathsf{ver}(\mathsf{sign}(M, N), \mathsf{pk}(N)) &= M
\end{aligned}
$$

Equations are used to define the semantics of destructors (let)

Example: Needham - Schroeder Protocol



$$A \triangleq a(x).\text{let } y = \text{dec}(x, K_A) \text{ in } (\nu n_A)\, \overline{b}\langle \text{enc}(\text{pair}(\text{snd}(y), n_A), \text{pk}(K_B))\rangle. \\ a(z).\text{let } w = \text{dec}(z, K_A) \text{ in if } w = n_A \text{ then } 0$$

$$B \triangleq (\nu n_B)\, \overline{a}\langle \text{enc}(\text{pair}(b, n_B), \text{pk}(K_A))\rangle.b(x).\text{let } y = \text{dec}(x, K_B) \text{ in} \\ \text{if fst}(y) = n_B \text{ then } \overline{a}\langle \text{enc}(\text{snd}(y), \text{pk}(K_A))\rangle$$

$$P \triangleq (\nu K_A)\,(\nu K_B)\,(A \mid B)$$

Modeling the Attacker

The attacker is implicitly modeled as an arbitrary process, which is run in parallel with the protocol

- the attacker knows all the public names, i.e., those names which are not bound by a restriction operator
- restricted names are revealed to the attacker once they are sent on public channels
- the attacker can exploit his knowledge to read/write on public channels and tamper with known cryptographic material

Previous case: $P \triangleq (\nu K_A)\,(\nu K_B)\,(A \mid B \mid \overline{net}\langle \mathrm{pk}(K_A)\rangle \mid \overline{net}\langle \mathrm{pk}(K_B)\rangle)$

Example

Consider the following process:

$$(\nu s)\,(\nu b)\,(\overline{a}\langle \mathsf{pair}(M, s)\rangle \mid a(x).\mathsf{if}\ \mathsf{snd}(x) = s\ \mathsf{then}\ \overline{b}\langle \mathsf{fst}(x)\rangle)$$

Can this process ever output something different from M on b?

Example

Consider the following process:

$$(\nu s)\,(\nu b)\,(\overline{a}\langle \mathsf{pair}(M, s)\rangle \mid a(x).\mathsf{if}\ \mathsf{snd}(x) = s\ \mathsf{then}\ \overline{b}\langle \mathsf{fst}(x)\rangle)$$

Can this process ever output something different from M on b?

Yes, pick the attacker: $a(y).\overline{a}\langle \mathsf{pair}(N, \mathsf{snd}(y))\rangle$