Protocol calculus:

$$\begin{array}{rcl}
P & ::= & \mathsf{stop} \\
 & | & (x := E); P \\
 & | & (x := \mathsf{new\ nonce/key/}\ldots\mathsf{/channel}); P \\
 & | & (x := \mathsf{receive}(y)); P \\
 & | & (\mathsf{send}\ x_1\ \mathsf{on}\ x_2); P \\
 & | & P_1 \mid P_2 \\
 & | & \mathsf{if}\ E_1 = E_2\ then\ P_1\ else\ P_2 \\
 & | & \mathsf{let}\ x = \mathbf{D}(E_1, \ldots, E_k)\ \mathsf{in}\ P_1\ \mathsf{else}\ P_2 \\
 & | & !P \\
E & ::= & x \mid \mathbf{C}(E_1, \ldots, E_k)
\end{array}$$

Each variable is defined only once.

- The communication happens on channels.
- To read a message, one has to know the corresponding channel.
- To write a message, one also has to know the channel where it is to be written.
  - Communication on a secret channel is invisible to the adversary.
- Non-public channels should not normally be used in protocols.
- They may be used for ideal protocols.

Shorter (and more traditional) write-up:

$$
\begin{array}{rcl}
P & ::= & \mathbf{0} \\
& | & (x := E).P \\
& | & (\nu x)P \\
& | & y(x).P \\
& | & \overline{x_2}\langle x_1\rangle; P \\
& | & P_1 \mid P_2 \\
& | & [E_1 = E_2]P \qquad | \qquad [E_1 \neq E_2]P \\
& | & \mathsf{let}\ x = \mathbf{D}(E_1, \ldots, E_k)\ \mathsf{in}\ P_1\ \mathsf{else}\ P_2 \\
& | & !P \\
E & ::= & x \mid \mathbf{C}(E_1, \ldots, E_k)
\end{array}
$$

To simplify notation (and be more general):

- Consider both constructors and destructors as constructors.
- Consider terms *modulo* an equational theory.

Example: pairing:

- Constructors: $(\cdot, \cdot)$, π_1, π_2.
- Equations: $\pi_1((x, y)) = x$, $\pi_2((x, y)) = y$.

Example: symmetric encryption:

- Constructors: E_s, D_s.
- Equation: $D_s(k, E_s(k, x)) = x$.

Example: asymmetric encryption:

- Constructors: pk, E_a, D_a.
- Equation: $D_a(k, E(\mathsf{pk}(k), x)) = x$.

Exercise: Diffie-Hellman? XOR?

A notion of types would be nice, too...

Our protocol from the first lecture:

$$
\begin{aligned}
A &\longrightarrow_{\text{pub}} B : \{\!|K_A, N_A, K_{AB}|\!\}_{K_B} \\
B &\longrightarrow_{\text{pub}} A : \{\!|N_A, N_B, K_B|\!\}_{K_A} \\
A &\longrightarrow_{\text{pub}} B : \{\!|N_A, N_B|\!\}_{K_B}
\end{aligned}
$$

$$(\nu sk_A)(\nu sk_B)(pi := (\mathsf{pk}(sk_A), \mathsf{pk}(sk_B))).\overline{c}\langle pi\rangle.(!P_A|!P_B)$$

$$\begin{gathered}P_A \equiv \mathsf{c}(pk_B).(\nu n_A)(\nu k_{AB})(m_1 := E(pk_B, (\mathsf{pk}(sk_A), n_A, k_{AB}))).\overline{c}\langle m_1\rangle. \\ \mathsf{c}(m_2).\mathsf{let}\ (n'_A, n_B, pk'_B) = D(sk_A, m_2)\ \mathsf{in} \\ [n'_A = n_A][pk'_B = pk_B](m_3 := E(pk_B, (n_A, n_B))).\overline{c}\langle m_3\rangle.\underline{\mathrm{OK}}\end{gathered}$$

$$\begin{aligned}P_B \equiv\ & \mathsf{c}(w_1).\mathsf{let}\ (pk_A, v_A, l_{AB}) = D(sk_B, w_1)\ \mathsf{in} \\ & (\nu v_B)(w_2 := E(pk_A, (v_A, v_B, \mathsf{pk}(sk_B)))).\overline{c}\langle w_2\rangle.\mathsf{c}(w_3). \\ & \quad \mathsf{let}\ (v'_A, v'_B) = D(sk_B, w_3)\ \mathsf{in}\ [v'_A = v_A][v'_B = v_B]\underline{\mathrm{OK}}\end{aligned}$$

Idealized version (s is a secret channel):

$$\begin{aligned}
A &\longrightarrow_{\mathrm{pub}} B : \{\![K_A, N_A, N_1]\!\}_{K_B} \\
B &\longrightarrow_{\mathrm{pub}} A : \{\![N_A, N_B, K_B]\!\}_{K_A} \\
A &\longrightarrow_{\mathrm{pub}} B : \{\![N_A, N_B]\!\}_{K_B} \\
A &\longrightarrow_{s} B : (N_A, N_B, K_{AB}) \\
B &\longrightarrow_{s} A : (N_A, N_B)
\end{aligned}$$

Exercise. Discuss the security of this protocol. Confidentiality? Authentication?

$$(\nu sk_A)(\nu sk_B)(\nu s)(pi := (\mathsf{pk}(sk_A), \mathsf{pk}(sk_B))).\overline{c}\langle pi\rangle.(!P_A|!P_B)$$

$$\begin{gathered}P_A \equiv c(pk_B).(\nu n_A)(\nu n_1)(m_1 := E(pk_B, (\mathsf{pk}(sk_A), n_A, n_1))).\overline{c}\langle m_1\rangle. \\ c(m_2).\mathsf{let}\ (n'_A, n_B, pk'_B) = D(sk_A, m_2)\ \mathsf{in} \\ [n'_A = n_A][pk'_B = pk_B](m_3 := E(pk_B, (n_A, n_B))).\overline{c}\langle m_3\rangle. \\ (\nu k_{AB})(m_4 := (n_a, n_b, k_{AB}).\overline{s}\langle m_4\rangle.s(n''_A, n''_B).[n_A = n''_A][n_B = n''_B]\underline{\mathrm{OK}}\end{gathered}$$

$$\begin{aligned}P_B \equiv c(w_1).&\mathsf{let}\ (pk_A, v_A, \_) = D(sk_B, w_1)\ \mathsf{in} \\ &(\nu v_B)(w_2 := E(pk_A, (v_A, v_B, \mathsf{pk}(sk_B)))).\overline{c}\langle w_2\rangle.c(w_3). \\ \mathsf{let}\ &(v'_A, v'_B) = D(sk_B, w_3)\ \mathsf{in}\ [v'_A = v_A][v'_B = v_B]s(v''_A, v''_B, l_{AB}). \\ &\quad [v_A = v''_A][v_B = v''_B](m_5 := (v_A, v_B)).\overline{s}\langle m_5\rangle.\underline{\mathrm{OK}}\end{aligned}$$

- We want real $\simeq$ ideal (equivalence of processes).
- We use testing equivalence.
- A test is a pair (R, β) of a process and a public channel.
- P passes the test (R, β), if $P \mid R$ sends something on channel β.
- $P \simeq Q$ if they pass the same tests.
- Testing equivalence is a congruence.
  - If $P \approx Q$ then $C[P] \approx C[Q]$ for all processes $C[]$ with a hole (contexts).
  - C does not see the variables defined by P and Q. P and Q see the variables defined by C.

In general, testing equivalence is hard to verify directly, because we have to quantify over all possible (adversarial) processes R.

Example of processes that are not testing equivalent. . .

Testing equivalence is hard to verify because

- We are concerned only about the "ends" of the runs of P and Q.
- Before reaching the end, the processes may do whatever they like.
- We run P and Q in parallel with an arbitrary R.

A bisimulation relation relates the states of currently running processes.

If two processes are bisimilar, then they

- React the same way to the same inputs;
- are afterwards again bisimilar.

$$\begin{tikzcd}
 & s_1 \arrow[dl, "a"'] \arrow[dr, "a"] & \\
s_2 \arrow[rr, bend left, "b"] & & s_3 \arrow[ll, bend left, "b"] \arrow[loop right, "b"]
\end{tikzcd}
\qquad\qquad
\begin{tikzcd}
t_1 \arrow[d, "a"] \\
t_2 \arrow[loop right, "b"]
\end{tikzcd}$$

$s_1 \sim t_1$, $s_2 \sim t_2$, $s_3 \sim t_2$.

A process in its initial state is just a process.

A running process consists of

- a set of *threads*, each of them
  - a process that remains to be executed;
  - a memory that assigns values to all already defined variables in this thread.
- The sequence of pairs (c, M), such that M has been sent out on a public channel c.
- The new values created by this process.

Static equivalence:

- Given two memories $\mathcal{M}_1$ and $\mathcal{M}_2$ with the same variables.
  - . . . or sequences of values.
- Try to tell them apart.
- Devise a computation that returns YES for one of them and NO for the other.
- If the sent-out variables of two currently running processes cannot be told apart then they are statically equivalent.

Bisimilarity, as known for process calculi, is too strong to be meaningful for cryptographic processes.

It does not take into account the properties of encryption:

- The environment distinguishes different ciphertexts.
- The environment can give any message as an input to the process.

A relation $\mathcal{R}$ between running processes is a cryptographic simulation if $(P, \mathcal{M}_P, \mathcal{N}_P) \mathrel{\mathcal{R}} (Q, \mathcal{M}_Q, \mathcal{N}_Q)$ implies

- The memories $\mathcal{M}_P$ and $\mathcal{M}_Q$ cannot be told apart.
- If $P \longrightarrow P'$, creating the values in $\mathcal{N}'_P$, then $Q \longrightarrow^* Q'$, creating the values in $\mathcal{N}'_Q$ and $(P', \mathcal{M}_P, \mathcal{N}_P \cup \mathcal{N}'_P) \mathrel{\mathcal{R}} (Q', \mathcal{M}_Q, \mathcal{N}_Q \cup \mathcal{N}'_Q)$.
  - Assume $\mathcal{N}'_P$ and $\mathcal{N}'_Q$ are different from everything else.
- If $P \xrightarrow{\overline{c}\langle M_P \rangle} P'$ then $Q \longrightarrow^* \xrightarrow{\overline{c}\langle M_Q \rangle} \longrightarrow^* Q'$ and $(P', \mathcal{M}_P : (c, M_P), \mathcal{N}_P) \mathrel{\mathcal{R}} (Q', \mathcal{M}_Q : (c, M_Q), \mathcal{N}_Q)$.
  - In particular, $\mathcal{M}_P : (c, M_P)$ and $\mathcal{M}_Q : (c, M_Q)$ cannot be told apart.

- If $P \xrightarrow{c(M_P)} P'$, such that $M_P = f(\mathcal{M}_P, n_1, n_2, \ldots, n_k)$ for
  - an expression f that does not create new values;
  - values $n_1, \ldots, n_k \notin \mathcal{N}_P \cup \mathcal{N}_Q$

  then $Q \longrightarrow^* \xrightarrow{c(M_Q)} \longrightarrow^* Q'$ and

$$(P', \mathcal{M}_P : (*, n_1) \cdots (*, n_k), \mathcal{N}_P) \ \mathcal{R} \ (Q', \mathcal{M}_Q : (*, n_1) \cdots (*, n_k), \mathcal{N}_Q)$$

  where $M_Q = f(\mathcal{M}_Q, n_1, n_2, \ldots, n_k)$.

A relation $\mathcal{R}$ between running processes is a cryptographic bisimulation if $\mathcal{R}$ and $\mathcal{R}^{-1}$ are cryptographic simulations.

Our real example is bisimilar to the ideal example.

Indeed, the first three messages are exchanged in the same way.

The last two messages of the ideal protocol will always succeed. . . or do they?

Protocol composition logic is a framework for systematizing the security proofs for cryptographic protocols.

- Language for protocols (roles, principals, runs).
- Language for formulas.
- Semantics (truth) of formulas.
- Axioms and inference rules.

- There are a number of principals, $Q_1, Q_2, \ldots$.
- There may be fixed long-term keys:
  - asymmetric encryption and signing keys of principals;
  - shared keys for symmetric encryption between pairs of principals.
  - We'll invent notation for them when needed. . .

A role is a single-threaded process.

- initiator, responder, server. . .

A protocol $\mathcal{Q}$ — principals and roles.

A session is a mapping of principals (including the attacker) to roles. Each point in this mapping is a thread.

A run is a sequence of actions.

Actions are

- $\text{Send}(X,t)$, $\text{Recv}(X,t)$
- $\text{New}(X,t)$
- $\text{Decr}(X,t)$, $\text{Verify}(X,t) \longleftarrow$ successful actions only

X may be one of the threads of $\mathcal{Q}$ or the adversary. t is a term.

A received term must be sent. A sent term must be constructable by the sender.

Formulas:

- each action is a formula.
- $\text{Has}(X, t)$ and $\text{Fresh}(X, t)$
- $\text{Start}(X)$
- $\text{Honest}(Q) \longleftarrow Q$ is a principal
- $\text{Exec}(Q^{\rho}, X)$
- $t_1 \sqsubseteq t_2$
- $\varphi \wedge \psi$, $\neg\varphi$, other propositional connectives
- $\forall x.\varphi$ and $\exists x.\varphi$
- $\Diamond\varphi$ and $\bigcirc\varphi$
- If P is a single-threaded process, then $[P]_X\varphi$

Let R be a run of protocol $\mathcal{Q}$ and E a mapping from variables to terms.

We define the relation $R, E \models_{\mathcal{Q}} \varphi$.

For a closed formula φ, we abbreviate it to $R \models_{\mathcal{Q}} \varphi$ (φ is correct in the run R of protocol $\mathcal{Q}$).

We say $\models_{\mathcal{Q}} \varphi$ if $R \models_{\mathcal{Q}} \varphi$ for all runs R.

$R \vDash_Q \langle\text{action}\rangle$ if $\langle\text{action}\rangle$ is the last action of the run R.

$Ra, E \vDash_Q \bigcirc\varphi$ if $R, E \vDash_Q \varphi$.

$R, E \vDash_Q \Diamond\varphi$ if $R', E \vDash_Q \varphi$ for some prefix R' of R.

$R, E \vDash_Q t_1 \sqsubseteq t_2$ if t_1 is a subterm of t_2.

$R \vDash_Q \text{Start}(X)$ if R contains no actions of the thread X.

$R, E \vDash_Q \exists x.\varphi$ if $R, E[x \mapsto t] \vDash_Q \varphi$ for some term t.

$R, E \vDash_Q \varphi \wedge \psi$ if principal Q executes the thread X.

$R, E \vDash_Q \varphi$ and $R, E \vDash_Q \psi$. Similar for other propositional connectives.

$R \vDash_Q \text{Exec}(Q^\rho, X)$ if principal Q is executing the thread X in the role ρ.

$R, E \vDash_Q \text{Has}(X, t)$ if t can be constructed from the terms that the thread X has received or generated.

$R, E \vDash_{\mathcal{Q}}$ Fresh(X, t) if

- thread X generated some new value m;
- t "significantly" depends on m;
- X has not sent out any term that contains m.

$R \vDash_{\mathcal{Q}}$ Honest(Q) if the threads of the principal Q in the run R follow the protocol $\mathcal{Q}$.

$R, E \vDash_{\mathcal{Q}} [P]_X \varphi$, if

- for all runs R', where $R'|_X$ is the sequence of actions performed by P;
- for the environment E' that maps the variables of P to values that P assigns to them in R',

we have $R||R', E \cup E' \vDash_{\mathcal{Q}} \varphi$.

Some axioms:

- All true statements of predicate calculus.
- $[\nu x]_X(\mathrm{Has}(X, x) \wedge \mathrm{Fresh}(X, x))$
- $[\nu x]_X(\mathrm{Has}(Y, x) \Rightarrow Y = X)$
- $[\mathsf{action}]_X\mathrm{action}$
- $[\mathsf{receive}\ x]_X\mathrm{Has}(X, x)$
- $[\mathsf{verify}(\{\!|m|\!\}_{K_Q})]_X\mathrm{Verify}(X, \{\!|m|\!\}_{K_Q})$
- $\mathrm{Honest}(Q) \wedge \mathrm{Decr}(X, \{\![m]\!\}_{K_Q}) \Rightarrow \mathrm{Exec}(Q, X)$
- $\mathrm{Has}(X, \{\![m]\!\}_{K_Q}) \wedge \mathrm{Has}(X, K_Q^{-1}) \Rightarrow \mathrm{Has}(X, m)$
- $\mathrm{Exec}(Q, X) \Rightarrow \mathrm{Has}(X, K_Q^{-1})$
- $[P]_X\varphi \wedge [P]_X\psi \Rightarrow [P]_X(\varphi \wedge \psi)$. Same for $\Diamond$ and $\bigcirc$.

Inference rules:

$$\frac{\varphi \quad \varphi \Rightarrow \psi}{\psi} \qquad \frac{\varphi}{\Diamond\varphi} \qquad \frac{\varphi}{\bigcirc\varphi}$$

$$\frac{\mathsf{Start}(X) \Rightarrow \varphi \qquad \forall \rho \in \mathcal{Q}\ \forall A \in \rho : \varphi \Rightarrow [A]_X \varphi}{\vdash_{\mathcal{Q}} \mathsf{Exec}(Q, X) \wedge \mathsf{Honest}(Q) \Rightarrow \varphi}$$

where ρ ranges over all roles in $\mathcal{Q}$ and A over all protocol steps in ρ.

The last rule allows us to establish protocol invariants.

$$A \longrightarrow B : \{\![K_A, N_A, K_{AB}]\!\}_{K_B}$$

$$B \longrightarrow A : \{\![N_A, N_B, K_B]\!\}_{K_A}$$

$$A \longrightarrow B : \{\![N_A, N_B]\!\}_{K_B}$$

$$B \longrightarrow A : \{M\}_{K_{AB}} \text{ (if the owner of } K_A \text{ is honest)}$$

Two roles (initiator and responder), three principals (A and B and the adversary).

Let K_I be the public key of the adversary.

We want to show $\mathrm{Has}(X, M) \Rightarrow (\mathrm{Exec}(A, X) \vee \mathrm{Exec}(B, X))$ if A and B are both honest.

We also want to show that some authentication property holds.

$$\text{Honest}(Q) \wedge \text{Exec}(Q^I, X) \wedge \text{Send}(X, \{\![K_Q, N_A, K_{AB}]\!\}_{K_{Q'}}) \Rightarrow$$
$$\bigcirc\,(\text{Fresh}(X, N_A) \wedge \text{Fresh}(X, K_{AB}))$$

$$\text{Honest}(Q) \wedge \text{Exec}(Q^R, X) \wedge \text{Send}(X, \{\![N_A, N_B, K_Q]\!\}_{K_{Q'}}) \Rightarrow$$
$$\Diamond \text{Recv}(X, \{\![K_Q, N_A, K_{AB}]\!\}_{K_Q}) \wedge \bigcirc \text{Fresh}(X, N_B)$$

$$\text{Honest}(Q) \wedge \text{Exec}(Q^I, X) \wedge \text{Send}(X, \{\![N_A, N_B]\!\}_{K_{Q'}}) \Rightarrow$$
$$\Diamond(\text{Recv}(X, \{\![N_A, N_B, K_{Q'}]\!\}_{K_Q}) \wedge \Diamond \text{Send}(X, \{\![K_Q, N_A, K_{AB}]\!\}_{K_{Q'}}))$$

$$\text{Honest}(Q) \wedge \text{Exec}(Q^R, X) \wedge \text{Send}(X, \{M\}_{K_{AB}}) \Rightarrow$$
$$(Q' = A \vee Q' = B) \wedge \Diamond(\text{Recv}(X, \{\![N_A, N_B]\!\}_{K_{Q'}}) \wedge$$
$$\Diamond(\text{Send}(X, \{\![N_A, N_B, K_Q]\!\}_{K_{Q'}}) \wedge \Diamond \text{Recv}(X, \{\![K_{Q'}, N_A, K_{AB}]\!\}_{K_Q})))$$

$$\mathrm{Honest}(Q) \wedge \mathrm{Exec}(Q^I, X) \wedge \mathrm{Send}(X, m_1) \wedge \\ \neg \bigcirc \Diamond \mathrm{Send}(X, m) \Rightarrow m = \{\![K_Q, N_A, K_{AB}]\!\}_{K_{Q'}}$$

$$\mathrm{Honest}(Q) \wedge \mathrm{Exec}(Q^I, X) \wedge \mathrm{Send}(X, m_1) \wedge \\ \bigcirc \Diamond \mathrm{Send}(X, m) \Rightarrow m = \{\![N_A, N_B]\!\}_{K_{Q'}}$$

$$\neg\Big(\mathrm{Honest}(Q) \wedge \mathrm{Exec}(Q^I, X) \wedge \mathrm{Send}(X, m_1) \wedge \\ \bigcirc \Diamond (\mathrm{Send}(X, m_2) \wedge \bigcirc\Diamond \mathrm{Send}(X, m_3))\Big)$$

Same ordering axioms also for the responder.

Authentication as matching conversation.

$$\mathsf{Auth}^{I}_{Q,X} \equiv \mathsf{Start}(X) \Rightarrow [\rho_I]_X \Big(\exists Q', Y : \mathsf{Honest}(Q') \wedge \mathsf{Exec}({Q'}^{R}, Y) \wedge$$

$$\mathsf{Send}(X, \{\!|K_Q, N_A, K_{AB}|\!\}_{K_{Q'}}) <$$

$$\mathsf{Recv}(Y, \{\!|K_Q, N_A, K_{AB}|\!\}_{K_{Q'}}) <$$

$$\mathsf{Send}(Y, \{\!|N_A, N_B, K_{Q'}|\!\}_{K_Q}) <$$

$$\mathsf{Recv}(X, \{\!|N_A, N_B, K_{Q'}|\!\}_{K_Q}) \Big)$$

Here

$$A < B \equiv \Diamond(B \wedge \bigcirc\Diamond A)$$

$$A_1 < A_2 < \cdots < A_n \equiv A_1 < A_2 \wedge A_2 < A_3 \wedge \cdots \wedge A_{n-1} < A_n$$

We want to show

$$\mathsf{Honest}(Q) \wedge \mathsf{Exec}(Q^I, X) \Rightarrow \mathsf{Auth}^{I}_{Q,X}$$