Cryptographic protocols

(MTAT.07.014, 4 AP / 6 ECTS)

Lectures and	Mon 12-16	hall 404
Exercises:	Thu 8-12	hall 224

homepage:

http://www.cs.ut.ee/~peeter_l/teaching/krprot10s

(contains lecture materials)

Grading: Home exercises and exam in January.

Overall topic of this course

- Cryptology I was mostly about primitives.
  - (A)symmetric encryption, signatures, MACs, hash functions, etc.
- To achieve the security goals of systems, several of them have to be used together.
- This gives us protocols.
- It's quite easy to use the primitives in the wrong way.
- This makes the protocols insecure, although the primitives themselves might have been secure.
  - Primitive $\equiv$ a lock
  - Protocol $\equiv$ how you use that lock

Example 0

- Alice and Bob want to set up a private channel between themselves.
- They know each other's public keys K_A and K_B.
- Alice generates a new key K_{AB} of some symmetric encryption system.
- Alice sends K_{AB} to B, encrypted with K_B.

$$A \longrightarrow B : \{[K_{AB}]\}_{K_B}$$

- Bob decrypts and learns K_{AB}.
- Alice and Bob use K_{AB} to encrypt messages between each other.
    - Assume it also provides integrity.

Immediate questions

- Who sent the key to Bob?
  - Alice did. . .
- Include Alice's name in the message:

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

- Although that does not prove anything. . . **Why?**

Immediate questions

- When was it sent?
  - consider replay attacks.
  - The adversary may somehow know the old session keys.
- Include a timestamp to the message:

$$A \longrightarrow B : \{[A, T, K_{AB}]\}_{K_B}$$

- B must check that T is not far off.
- How do A and B synchronize their clocks?
- What if the attacker takes over B's NTP server?

Instead of a timestamp

- Better: include a nonce in the message:

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

  - Nonce $\equiv$ random bit-string.

- B must check that it has not received that N before.

Instead of a timestamp

- Better: include a nonce in the message:

$$A \longrightarrow B : \{[A, N, K_{AB}]\}_{K_B}$$

  - Nonce $\equiv$ random bit-string.

- B must check that it has not received that N before.
- B has to store all N-s it receives... What if his hard drive fails?
- The attacker may
  1. not deliver the message $\{[A, N, K_{AB}]\}_{K_B}$;
  2. wait until it learns K_{AB};
  3. deliver $\{[A, N, K_{AB}]\}_{K_B}$.

Liveness of A

- B needs to know that A sent that message recently.
- B must answer to A and then A must answer to B.

$$
\begin{aligned}
A \longrightarrow B &: \{\![A, N, K_{AB}]\!\}_{K_B} \\
B \longrightarrow A &: \{\![???]\!\}_{K_A} \\
A \longrightarrow B &: \{\![???]\!\}_{K_B}
\end{aligned}
$$

Liveness of A

- 2nd and 3rd message have to mention N.

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

- A must verify that it sent N recently.
- B must do the same verification after 3rd message.
- What replay possibilities are there?

Liveness of A

- B needs a nonce, too.

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

Man-in-the-middle attack

Assume now that Alice wants to talk to Charlie

$A \longrightarrow C : \{[A, N_A, K_{AC}]\}_{K_C}$

Man-in-the-middle attack

Assume now that Alice wants to talk to Charlie

But Charlie is bad. . .

$$A \longrightarrow C : \{[A, N_A, K_{AC}]\}_{K_C}$$

$$C(A) \longrightarrow B : \{[A, N_A, K_{AC}]\}_{K_B}$$

Man-in-the-middle attack

Assume now that Alice wants to talk to Charlie

$A \longrightarrow C : \{[A, N_A, K_{AC}]\}_{K_C}$

But Charlie is bad...

$C(A) \longrightarrow B : \{[A, N_A, K_{AC}]\}_{K_B}$

Bob responds, thinking that Alice is talking to him:

$B \longrightarrow C(A) : \{[N_A, N_B]\}_{K_A}$

Man-in-the-middle attack

Assume now that Alice wants to talk to Charlie $A \longrightarrow C : \{[A, N_A, K_{AC}]\}_{K_C}$

But Charlie is bad... $C(A) \longrightarrow B : \{[A, N_A, K_{AC}]\}_{K_B}$

Bob responds, thinking that Alice is talking to him: $B \longrightarrow C(A) : \{[N_A, N_B]\}_{K_A}$

Charlie simply forwards that message: $C \longrightarrow A : \{[N_A, N_B]\}_{K_A}$

Man-in-the-middle attack

Assume now that Alice wants to talk to Charlie $A \longrightarrow C : \{[A, N_A, K_{AC}]\}_{K_C}$

But Charlie is bad... $C(A) \longrightarrow B : \{[A, N_A, K_{AC}]\}_{K_B}$

Bob responds, thinking that Alice is talking to him: $B \longrightarrow C(A) : \{[N_A, N_B]\}_{K_A}$

Charlie simply forwards that message: $C \longrightarrow A : \{[N_A, N_B]\}_{K_A}$

Alice decrypts that pair of nonces for Charlie: $A \longrightarrow C : \{[N_A, N_B]\}_{K_C}$

Man-in-the-middle attack

Assume now that Alice wants to talk to Charlie	$A \longrightarrow C : \{[A, N_A, K_{AC}]\}_{K_C}$
But Charlie is bad...	$C(A) \longrightarrow B : \{[A, N_A, K_{AC}]\}_{K_B}$
Bob responds, thinking that Alice is talking to him:	$B \longrightarrow C(A) : \{[N_A, N_B]\}_{K_A}$
Charlie simply forwards that message:	$C \longrightarrow A : \{[N_A, N_B]\}_{K_A}$
Alice decrypts that pair of nonces for Charlie:	$A \longrightarrow C : \{[N_A, N_B]\}_{K_C}$
and Charlie can respond to Bob:	$C(A) \longrightarrow B : \{[N_A, N_B]\}_{K_B}$

Man-in-the-middle attack

Assume now that Alice wants to talk to Charlie	$A \longrightarrow C : \{[A, N_A, K_{AC}]\}_{K_C}$
But Charlie is bad...	$C(A) \longrightarrow B : \{[A, N_A, K_{AC}]\}_{K_B}$
Bob responds, thinking that Alice is talking to him:	$B \longrightarrow C(A) : \{[N_A, N_B]\}_{K_A}$
Charlie simply forwards that message:	$C \longrightarrow A : \{[N_A, N_B]\}_{K_A}$
Alice decrypts that pair of nonces for Charlie:	$A \longrightarrow C : \{[N_A, N_B]\}_{K_C}$
and Charlie can respond to Bob:	$C(A) \longrightarrow B : \{[N_A, N_B]\}_{K_B}$

Now Bob thinks that he shares the key K_{AC} with Alice, but Charlie also knows that key.

A possible fix

- B's answer must contain his name:

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

- Is this protocol secure? Maybe. . .
- Are all its parts necessary?
  - Do we need all components of all messages?
  - Does everything have to be under encryption?

Probably not.

More fundamental questions

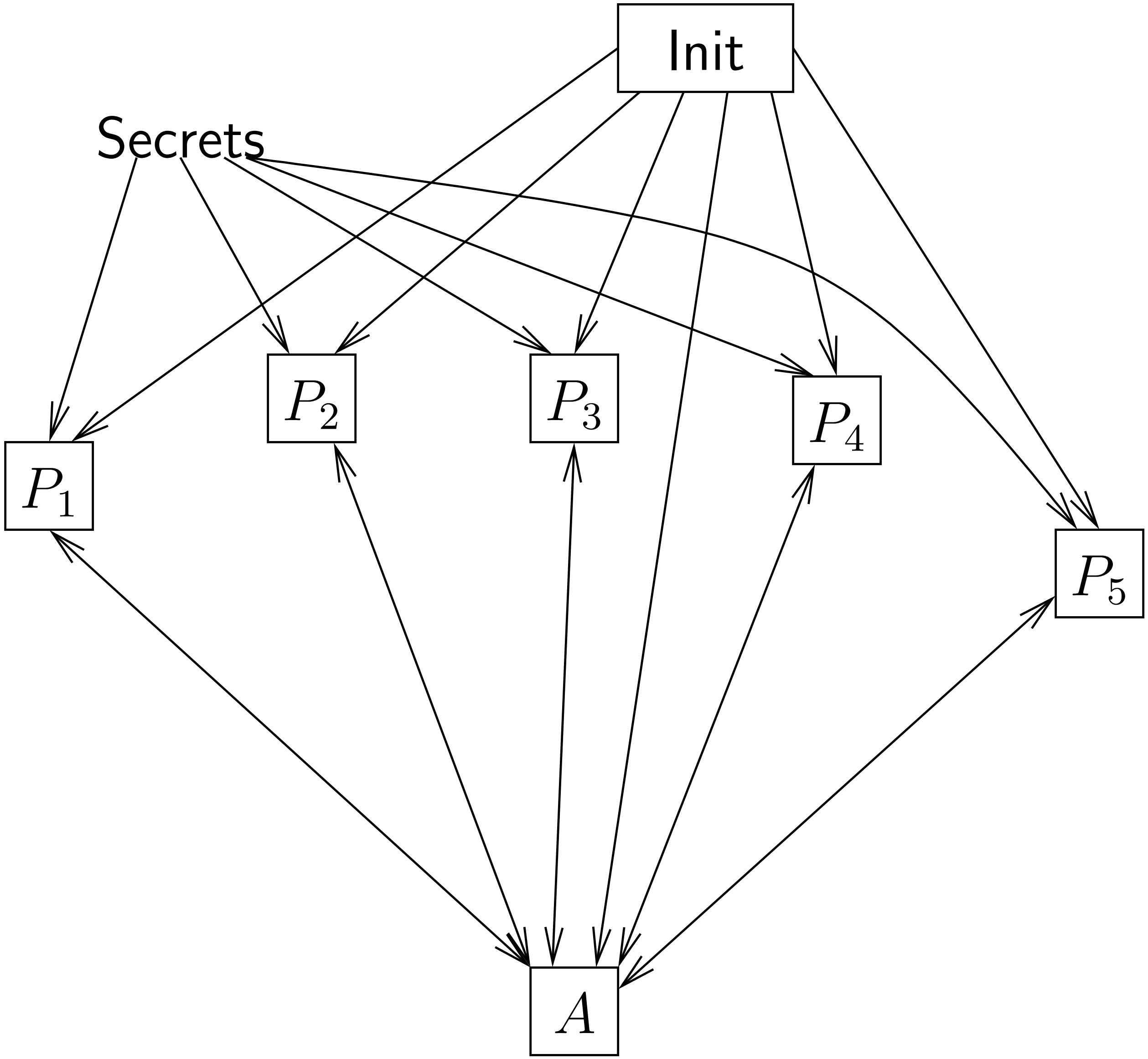
- What is the security property?
- What did this $A \longrightarrow B : M$ actually mean? Or:
- What is the execution model?
  - What data and control structures do the parties use?
  - How are the messages relayed?
  - How are the parties scheduled?
  - Where is the adversary?
    - How are the parties corrupted and the keys leaked?

We do not need answers to all of these questions as long as we are just showing attacks against protocols.

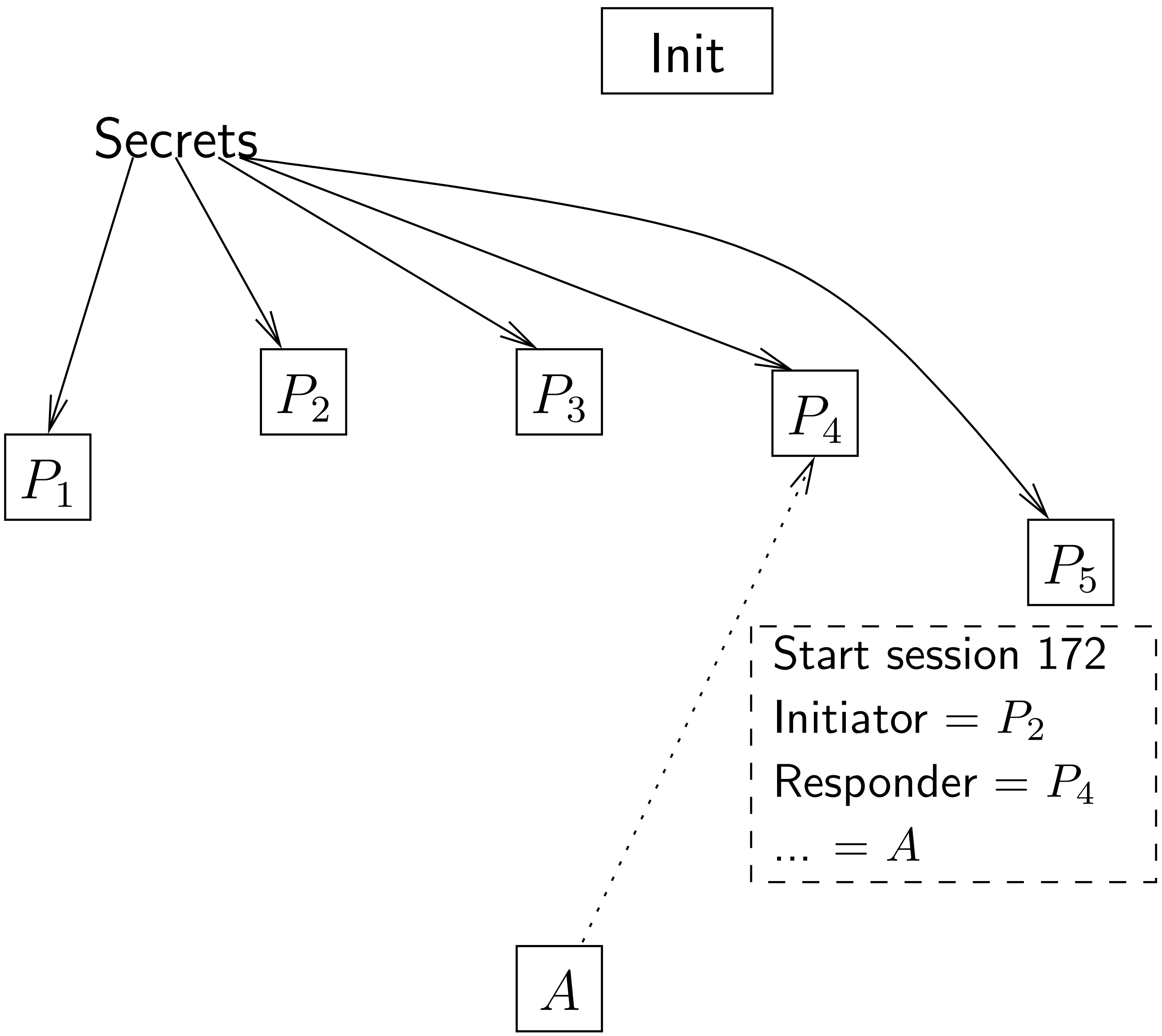
Formally

- Each party is an implementation of some interface. It has methods for
  - starting a session;
  - receiving a message and producing and answer;
  - maybe something more.
- The adversary has a method “run” that takes all participants as its arguments.
  - More generally: there is an environment with a method “run” that takes both the participants and the adversary as arguments.
  - The implementation of this environment is fixed. This defines the scheduling and the relaying of messages.

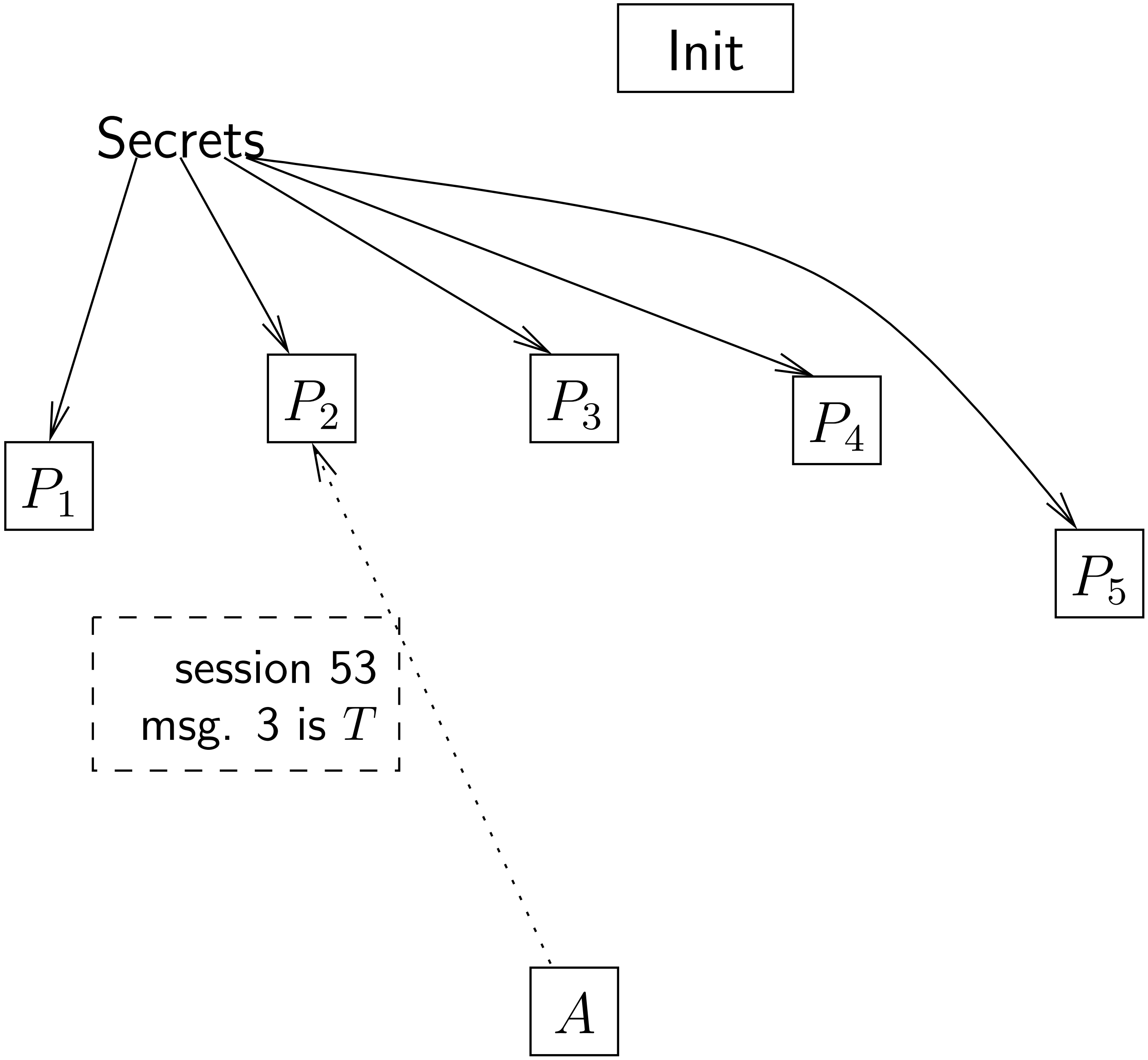
Setup of parties



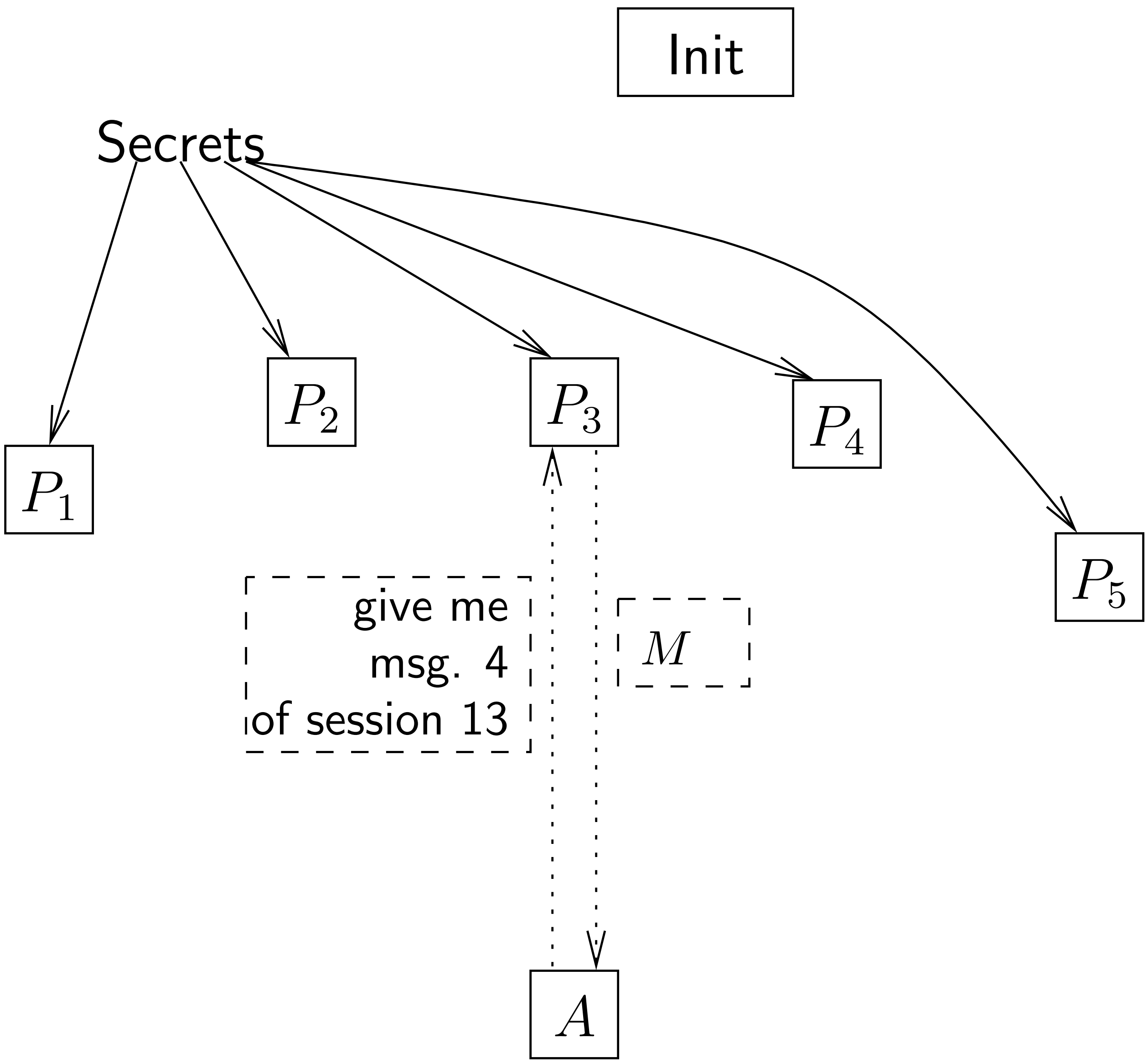
Possible commands to parties



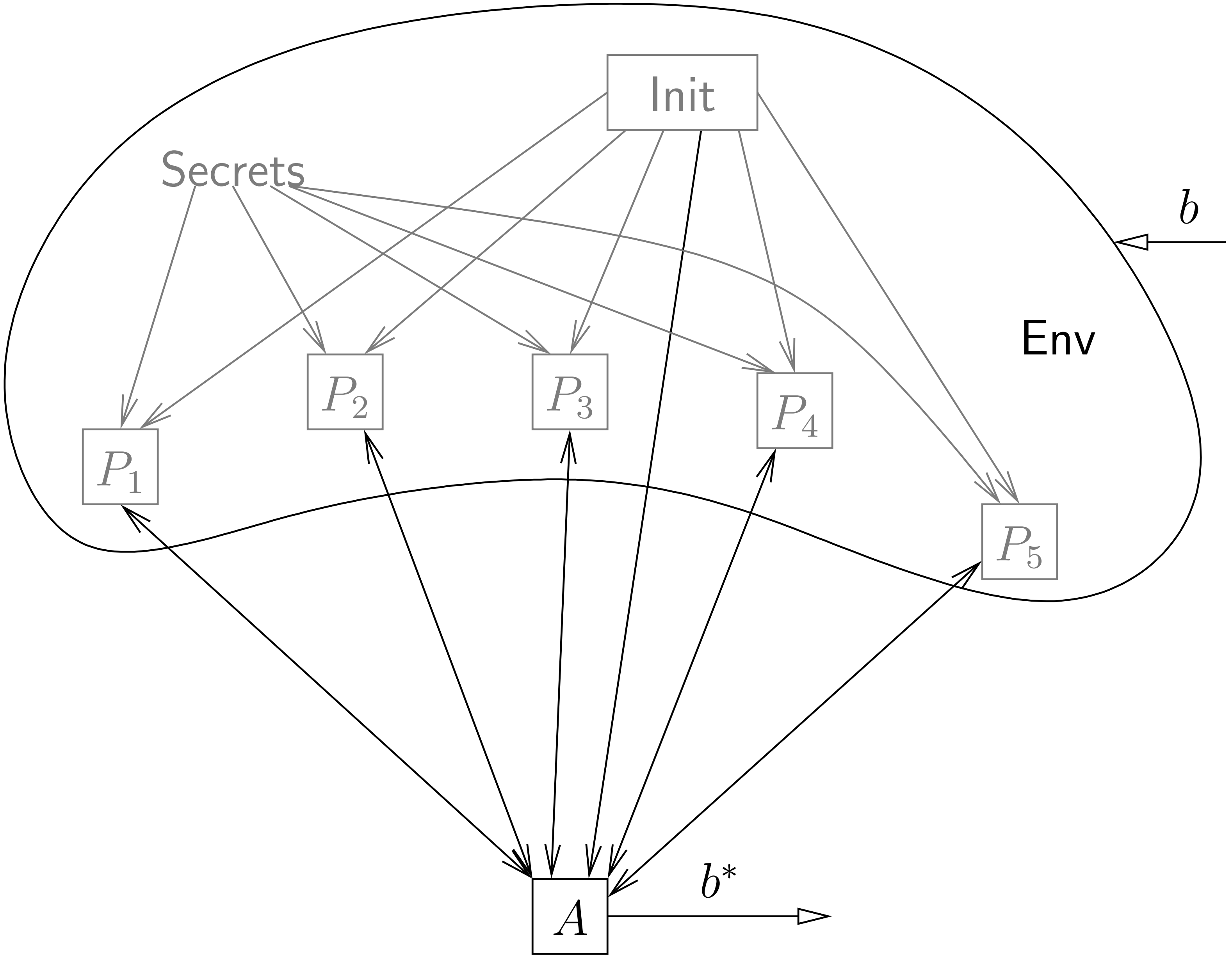
Possible commands to parties



Possible commands to parties



Environment defining the secrecy of something



- Such analysis may be hard. . .
    - but we'll be rewarded with rigorous security proofs.
- But, intuitively, what are the things that an adversary may do?

The adversary can. . .

- Capture messages sent by one party to another.
  - Learn the intended sender and recipient.
- Send a message it has constructed to any party.
  - . . . faking the sender.
- Generate new keys, nonces, . . .
- Construct new messages from the ones its has.
  - Only applying “legitimate” constructors.
  - Because everything else will be weeded out by other parties. . .
- Decompose tuples. Decrypt if it knows the key.

The adversary cannot...

The adversary cannot do things like:

- Learn anything about M from $\{[M]\}_K$.
- Transform $\{M_1\}_K, \ldots, \{M_n\}_K$ to $\{M'\}_K$ for M' related to $M_1, \ldots, M_n$, not knowing the key K.
- ...or construct any $\{M\}_K$ without knowing K at all.

Hence the encryption must provide message integrity, too.

- Such encryption is often called perfect.
- In the next few lectures we make the perfect cryptography assumption (also called the *Dolev-Yao model*).

Contents of this course

- Analysis of protocols in the perfect cryptography model (≈ 3 weeks)
- General secure multiparty computation (≈ 3 weeks)
- Universal composability (≈ 2 weeks)

Modeling computation / communication

- There are many calculi for modeling parallel / distributed processes
  - CCS, CSP, join-calculus,. . .
- π-calculus was preferred by security researchers
  - Because of the **new**-operation in it
    - Used for channel creation
- π-calculus begat spi-calculus and applied pi-calculus
  - **new** used also for generating keys, nonces,. . .

calculus $\equiv$ programming language and its semantics

π-calculus

- Let us have
  - a countable set of names: $m, n, k, l, a, b, c, \ldots$
  - a countable set of variables: $x, y, z, w, \ldots$
- Messages $M, N, K, L, \ldots$ are either names or variables.
- Processes $P, Q, R, \ldots$ are one of

$\mathbf{0}$	(stopped process)
$\overline{N}\langle M\rangle.P$	(send M over channel N, then do P)
$N(x).P$	(receive message from channel N, store in x, do P)
$P \mid Q$	(do P and Q in parallel)
$!P$	(intuitively same as $P \mid P \mid P \mid \cdots$)
$(\nu m)P$	(generate new name m, continue with P)
$[M = N].P$	(if M equals N then do P)

Examples

- $\overline{c}\langle m\rangle.\mathbf{0}$ sends message m on channel c
- $c(x).\overline{d}\langle x\rangle.\mathbf{0}$ receives a message on channel c and forwards it on channel d
- $(\nu m)\overline{c}\langle m\rangle.\mathbf{0}$ generates a new name and sends it on channel c
- $(\nu c)((\nu m)\overline{c}\langle m\rangle \mid c(x).\overline{d}\langle x\rangle)$ causes a newly generated name to be sent on channel d
- $(\nu c)((\nu m)\overline{c}\langle m\rangle \mid c(x).\overline{d_1}\langle x\rangle \mid c(x).\overline{d_2}\langle x\rangle)$ causes a newly generated name to be sent either on channel d_1 or channel d_2

Free and bound (occurrences of) names and variables

- An occurrence can be free, a binder or bound to a previous binder.
- In processes:

$$
\begin{array}{llll}
\mathbf{0} & \overline{N}\langle M\rangle.P & N(x).P_{x\to x} & P \mid Q \\
!P & (\nu m)P_{m\to m} & [M = N].P &
\end{array}
$$

- P and Q are structurally congruent, $P \equiv Q$, if they differ only by renaming of bound variables and names:
  - No captures! $c(x).c(y).\overline{x}\langle m\rangle.\overline{y}\langle n\rangle \not\equiv c(y).c(y).\overline{y}\langle m\rangle.\overline{y}\langle n\rangle$.
    - But $c(x).\overline{x}\langle m\rangle.c(y).\overline{y}\langle n\rangle \equiv c(y).\overline{y}\langle m\rangle.c(y).\overline{y}\langle n\rangle$.
- Let $P\{M_1,\ldots,M_n/u_1,\ldots,u_n\}$ denote the simulataneous substitution of variables/names $u_1,\ldots,u_n$ with messages $M_1,\ldots,M_n$.
  - No captures! Rename bound variables in P as needed.

Structural congruence

- $P \equiv Q$, if they differ only by renaming of bound variables and names
- $P \mid Q \equiv Q \mid P$, $(P \mid Q) \mid R \equiv P \mid (Q \mid R)$, $P \mid 0 \equiv P$
- $!P \equiv P \mid !P$
- $(\nu m)(\nu n)P \equiv (\nu n)(\nu m)P$, $(\nu m)\mathbf{0} \equiv \mathbf{0}$
- $P \mid (\nu m)Q \equiv (\nu m)(P \mid Q)$ if n not free in P
- Congruence! If $P \equiv Q$ then $R[P] \equiv R[Q]$

Operational semantics

- . . . is defined by the step relation $\rightarrow \subseteq Proc \times Proc$.
  - $Proc$ — the set of all processes.
- $\overline{N}\langle M\rangle.P \mid N(x).Q \rightarrow P \mid Q\{M/x\}$
- $[M = M].P \rightarrow P$
- If $P \equiv P' \rightarrow Q' \equiv Q$ then $P \rightarrow Q$
- If $P \rightarrow Q$ then $P \mid R \rightarrow Q \mid R$ and $(\nu m)P \rightarrow (\nu m)Q$
- Not a congruence!

Example

$$
\begin{aligned}
&(\nu c)((\nu m)\overline{c}\langle m\rangle \mid c(x).\overline{d}\langle x\rangle) \\
\equiv&(\nu c)(\nu m)(\overline{c}\langle m\rangle \mid c(x).\overline{d}\langle x\rangle) \\
\rightarrow&(\nu c)(\nu m)(\mathbf{0} \mid \overline{d}\langle m\rangle) \\
\equiv&(\nu m)(\nu c)(\mathbf{0} \mid \overline{d}\langle m\rangle) \\
\equiv&(\nu m)((\nu c)\mathbf{0} \mid \overline{d}\langle m\rangle) \\
\equiv&(\nu m)(\mathbf{0} \mid \overline{d}\langle m\rangle) \\
\equiv&(\nu m)\overline{d}\langle m\rangle
\end{aligned}
$$

spi-calculus

- . . . enriches the structure of messages
- . . . introduces operations to analyze (take apart) messages
- Let Σ be a finite set of term constructors
  - pairing, encryption, signing, hashing, etc.
- Let $\mathrm{ar} : \Sigma \to \mathbb{N}$ give the arity of each constructor.
- A message is one of
  - variable
  - name
  - $f(M_1, \ldots, M_{\mathrm{ar}(f)})$, where $f \in \Sigma$.

For now, let the constructors be

- pk(K) gives the public key corresponding to secret decryption / signing key K.
- $(M_1, \ldots, M_n)$ is the tuple of the messages $M_1, \ldots, M_n$.
- $\{M\}_K$, $\{\![M]\!\}_{K_p}$, $[\![\{M\}]\!]_{K_s}$ are the symmetric, asymmetric encryption and signatures.
  - If we model randomized primitives then there is the third argument, too — the random coins.
- $h(M)$ is the digest of M.

A party can apply a constructor if it knows all of its arguments.

Destructors

- Besides Σ and ar we are given a set of message destructors. They have
  - A name g and arity $ar(g)$, e.g. dec /2
  - Arguments, e.g. x_{key},$\{x_M\}_{x_{\mathrm{key}}}$
  - One or more possible results, e.g. x_M
- Denote $g(M_1, \ldots, M_{\mathrm{ar}(g)}) \rightarrow M$
  - No names in $M_1, \ldots, M_{\mathrm{ar}(g)}, M$.
- More examples:
  - $\pi_i^n((x_1, \ldots, x_n)) \rightarrow x_i$
  - $vfy(\mathsf{pk}(x_{\mathrm{key}}), x_M, [\![x_M]\!]_{x_{\mathrm{key}}}) \rightarrow \mathsf{true}$
    - $\mathsf{true} \in \Sigma$. $\mathrm{ar}(\mathsf{true}) = 0$

Applying destructors

- A process can also be

$$[x := g(M_1, \ldots, M_k)].P \quad \text{(binds } x \text{ in } P\text{)}$$

- The step relation is extended by

$$[x := g(M_1\sigma, \ldots, M_k\sigma)].P \rightarrow P\{M\sigma/x\} \quad \text{where}$$

  - $g(M_1, \ldots, M_k) \rightarrow M$
  - σ is a substitution from variables in $M_1, \ldots, M_k, M$ to messages.

A protocol consists of

- The initialization of common variables;
  - Mainly long-term keys
- The parallel composition of all parties.

The protocol is executed in parallel with the adversary.

- The adversary can be any process

Our example

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

Names $\cong$ public keys

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

Alice's process (single session)

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

$P_{\mathrm{A}}(SK_A, K_B)$ is

$$\begin{array}{l}(\nu n_A)(\nu k_{AB}).\overline{c}\langle\{[\mathsf{pk}(SK_A), n_A, k_{AB}]\}_{K_B}\rangle. \\ c(y_2).[z_2 := dec(SK_A, y_2)]. \\ [x_{NA} := \pi_1^3(z_2)].[x_{NB} := \pi_2^3(z_2)].[x_{KB} := \pi_3^3(z_2)]. \\ [n_A = x_{NA}].[x_{KB} = K_B].\overline{c}\langle\{[n_A, x_{NB}]\}_{K_B}\rangle\end{array}$$

- SK_A is the decryption key of party A. K_B is the public key of B.
- c is the public channel (Internet)

Bob's process (single session)

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

$P_{\mathrm{B}}(SK_B, K_A)$ is

$$\begin{array}{l} c(y_1).[z_1 := dec(SK_B, y_1)]. \\ [x_{KA} := \pi^3_1(z_1)].[x_{NA} := \pi^3_2(z_1)].[x_{KAB} := \pi^3_3(z_1)]. \\ [x_{KA} = K_A].(\nu n_B).\overline{c}\langle\{[x_{NA}, n_B, \mathsf{pk}(SK_B)]\}_{K_A}\rangle. \\ c(y_3)[z_3 := dec(SK_B, y_3)]. \\ [x_{NA2} := \pi^2_1(z_3)].[x_{NB} := \pi^2_2(z_3)].[x_{NA2} = x_{NA}].[x_{NB} = n_B] \end{array}$$

SK_B is the decryption key of party B.

Whole protocol

(Alice as initiator, Bob as responder)

$$
\begin{array}{l}
(\nu sk_A)(\nu sk_B). \\
(\\
\quad !(c(x_K).P_{\mathrm{A}}(sk_A, x_K)) \mid \\
\quad !(c(x_K).P_{\mathrm{B}}(sk_B, x_K)) \mid \\
\quad \overline{c}\langle \mathsf{pk}(sk_A) \rangle \mid \overline{c}\langle \mathsf{pk}(sk_B) \rangle \\
)
\end{array}
$$

. . . and this is executed in parallel with the adversary.

Exercise. How to express that both Alice and Bob can serve as both initiator and responder?

Security properties:

- Secrecy of something — this thing cannot become the value of some variable in the adversarial process.
  - Generally a weaker property than "the adversary cannot distinguish which one of two fixed values this thing has".
  - Justified by the perfection of the cryptographic primitives.
- Authenticity — a certain situation cannot happen...
  - B thinks it shares K_{AB} with A, but A thinks that K_{AB} is for a different purpose...

Alice thinks...

$P_{\mathrm{A}}(SK_A, K_B)$ is

$$(\nu n_A)(\nu k_{AB}).$$
. o O (start session with K_B using (n_A, k_{AB}))
$$\overline{c}\langle\{[\mathsf{pk}(SK_A), n_A, k_{AB}]\}_{K_B}\rangle.$$
$$c(y_2).[z_2 := dec(SK_A, y_2)].$$
$$[x_{NA} := \pi^3_1(z_2)].[x_{NB} := \pi^3_2(z_2)].[x_{KB} := \pi^3_3(z_2)].$$
$$[n_A = x_{NA}].[x_{KB} = K_B].$$
. o O (end session with K_B using (n_A, x_{NB}, k_{AB}))
$$\overline{c}\langle\{[n_A, x_{NB}]\}_{K_B}\rangle$$

Bob thinks. . .

$P_{\mathrm{B}}(SK_B, K_A)$ is

$$
\begin{aligned}
& c(y_1).[z_1 := dec(SK_B, y_1)]. \\
& [x_{KA} := \pi^3_1(z_1)].[x_{NA} := \pi^3_2(z_1)].[x_{KAB} := \pi^3_3(z_1)]. \\
& [x_{KA} = K_A].(\nu n_B).
\end{aligned}
$$

. o O (start session with K_A using (x_{NA}, n_B, x_{KAB}))

$$
\begin{aligned}
& \overline{c}\langle \{[x_{NA}, n_B, \mathsf{pk}(SK_B)]\}_{K_A} \rangle. \\
& c(y_3)[z_3 := dec(SK_B, y_3)]. \\
& [x_{NA2} := \pi^2_1(z_3)].[x_{NB} := \pi^2_2(z_3)].[x_{NA2} = x_{NA}].[x_{NB} = n_B].
\end{aligned}
$$

. o O (end session with K_A using (x_{NA}, n_B, x_{KAB}))

Authentication property

If B ended session with pk(sk_A) using (n_1, n_2, k) then A ended session with pk(sk_B) using (n_1, n_2, k).

If A ended session with pk(sk_B) using (n_1, n_2, k) then B started session with pk(sk_A) using (n_1, n_2, k).

. . . and for different red thoughts correspond different green thoughts.

Scheduling

- Scheduling of protocols — non-deterministic.
- We get a set of protocol traces, not a probability distribution over them.
- Justification — both secrecy and authentication properties are specified by valid protocol traces.
- In our actual arguments we just assume that everything that may go wrong goes wrong.
  - Most secure computer — the one that is switched off
  - Most functional computer — the attacker

Arguing about the protocol

(A1) B ended session i with $K_A[i]$.

(A2) $K_A[i] = \mathsf{pk}(sk_A)$.

(1) $m_3[i]$, which came from outside, contained the value of $N_B[i]$.

(2) $n_B[i]$ left the scope of the current session only inside the second message $M_2[i]$.

(3) $M_2[i]$ was encrypted with $K_A[i] = \mathsf{pk}(sk_A)$. Only someone who knows sk_A is able to decrypt it.

(4) sk_A is used only to get the corresponding public key, and to decrypt. Hence the adversary cannot know sk_A.

Arguing about the protocol

(5) A had a session j where she decrypted $M_2[i] = y_2[j]$. Hence

- $x_{NA}[j] = x_{NA}[i]$, $x_{NB}[j] = n_B[i]$, $x_{KB}[j] = \mathsf{pk}(sk_B)$.
- Maybe there were several such sessions j.

(6) $x_{NB}[j]$ left the scope of the session j only inside the third message $M_3[j]$.

- $K_B[j] = x_{KB}[j] = \mathsf{pk}(sk_B)$, $n_A[j] = x_{NA}[j] = x_{NA}[i]$.
- A ended session j with $K_B[j]$.

- We still have to show that
  - $k_{AB}[j] = x_{KAB}[i]$
  - There is no $i' \neq i$, such that B ended session i' with $\mathsf{pk}(sk_A)$ using $(x_{NA}[i], n_B[i], x_{KAB}[i])$.
    - Easy — $n_B[i'] \neq n_B[i]$.

Arguing about the protocol

(7) $x_{KAB}[i]$ is defined together with $x_{NA}[i]$ which equals $n_A[j]$.

Can the adversary construct a message of the form

$$\{[\mathsf{pk}(sk_A), x_{NA}[i], K']\}_{\mathsf{pk}(sk_B)} \text{ with } K' \neq x_{KAB}[j] \text{ ?}$$

(8) $n_A[j]$ is sent out in messages $M_1[j]$ and $M_3[j]$. They are encrypted with $\mathsf{pk}(sk_B)$.

(9) The adversary does not know sk_B.

(10) B does not accept the message $M_3[j]$ as the first message from A.

(11) If B accepts $M_1[j]$ in some session k, then $K_A[k] = \mathsf{pk}(sk_A)$. Hence the adversary cannot decrypt $M_2[k]$.

The adversary cannot learn $x_{NA}[i]$.

Arguing about the protocol

- The adversary cannot learn $x_{NA}[i] = n_A[j]$ and there is only a single first message containing it constructed by A.
- This message contains the key $k_{AB}[j]$.
- Injective agreement would still have hold if A's belief about ending a session had not contained x_{NB}.
- The other property is proved similarly.
- Secrecy of k_{AB} is shown similarly to the secrecy of n_A.

Correspondence properties

- Authentication properties can be specified using correspondence properties.
- Introduce steps $\mathbf{begin}(M)$ and $\mathbf{end}(M)$ to the calculus.
- These statements do nothing but appear in the trace of the protocol.
  - $\mathbf{begin}(M).P \to P$
  - $\mathbf{end}(M).P \to P$
- A protocol has agreement if every $\mathbf{end}(M)$ in a trace is preceeded by $\mathbf{begin}(M)$.
- A protocol has injective agreement if it satisfies agreement and one can find a different $\mathbf{begin}$ corresponding to each $\mathbf{end}$.

$P_{\mathrm{A}}(SK_A, K_B)$ is

$$(\nu n_A)(\nu k_{AB}).$$
. o O (start session with K_B using (n_A, k_{AB}))
$$\overline{c}\langle\{\![\mathsf{pk}(SK_A), n_A, k_{AB}]\!\}_{K_B}\rangle.$$
$$c(y_2).[z_2 := dec(SK_A, y_2)].$$
$$[x_{NA} := \pi^3_1(z_2)].[x_{NB} := \pi^3_2(z_2)].[x_{KB} := \pi^3_3(z_2)].$$
$$[n_A = x_{NA}].[x_{KB} = K_B].$$
. o O (end session with K_B using (n_A, x_{NB}, k_{AB}))
$$\overline{c}\langle\{\![n_A, x_{NB}]\!\}_{K_B}\rangle$$

$P_{\mathrm{A}}(SK_A, K_B)$ is

$$
\begin{array}{l}
(\nu n_A)(\nu k_{AB}). \\
\overline{c}\langle\{[\mathsf{pk}(SK_A), n_A, k_{AB}]\}_{K_B}\rangle. \\
c(y_2).[z_2 := dec(SK_A, y_2)]. \\
[x_{NA} := \pi^3_1(z_2)].[x_{NB} := \pi^3_2(z_2)].[x_{KB} := \pi^3_3(z_2)]. \\
[n_A = x_{NA}].[x_{KB} = K_B]. \\
\mathbf{end}(\text{“startB”}, n_A, x_{NB}, k_{AB}).\mathbf{begin}(\text{”endB”}, n_A, x_{NB}, k_{AB}). \\
\overline{c}\langle\{[n_A, x_{NB}]\}_{K_B}\rangle
\end{array}
$$

$P_{\mathrm{B}}(SK_B, K_A)$ is

$c(y_1).[z_1 := dec(SK_B, y_1)].$
$[x_{KA} := \pi^3_1(z_1)].[x_{NA} := \pi^3_2(z_1)].[x_{KAB} := \pi^3_3(z_1)].$
$[x_{KA} = K_A].(\nu n_B).$
. o O (start session with K_A using (x_{NA}, n_B, x_{KAB}))
$\overline{c}\langle\{[x_{NA}, n_B, \mathsf{pk}(SK_B)]\}_{K_A}\rangle.$
$c(y_3)[z_3 := dec(SK_B, y_3)].$
$[x_{NA2} := \pi^2_1(z_3)].[x_{NB} := \pi^2_2(z_3)].[x_{NA2} = x_{NA}].[x_{NB} = n_B].$
. o O (end session with K_A using (x_{NA}, n_B, x_{KAB}))

$P_{\mathrm{B}}(SK_B, K_A)$ is

$$
\begin{aligned}
& c(y_1).[z_1 := dec(SK_B, y_1)]. \\
& [x_{KA} := \pi^3_1(z_1)].[x_{NA} := \pi^3_2(z_1)].[x_{KAB} := \pi^3_3(z_1)]. \\
& [x_{KA} = K_A].(\nu n_B). \\
& \mathbf{begin}(\text{“startB”}, x_{NA}, n_B, x_{KAB}). \\
& \overline{c}\langle \{\![x_{NA}, n_B, \mathsf{pk}(SK_B)]\!\}_{K_A} \rangle. \\
& c(y_3)[z_3 := dec(SK_B, y_3)]. \\
& [x_{NA2} := \pi^2_1(z_3)].[x_{NB} := \pi^2_2(z_3)].[x_{NA2} = x_{NA}].[x_{NB} = n_B]. \\
& \mathbf{end}(\text{“endB”}, x_{NA}, n_B, k_{AB})
\end{aligned}
$$

Key-establishment protocols are just one case where authentication is necessary.

In pure authentication protocols (entity authentication) two parties have established a connection. Party A wants to check that the other one is who A thinks it is.

- In a connectionless model of communication, entity authentication is used to check the liveness of the other party.

Mutual authentication — both parties check each other's liveness.

Basic tool for one-way entity authentication: challenge-response mechanism.

- A sends a new nonce to B.
- B transforms that nonce in a way that only B (or A) could do and sends back the result.
- A checks the result.

Let $Cert_X$ be the certificate of the verification key $\mathsf{pk}(K_X)$ of the party X.
Alice checking Bob's liveness:

$$A \longrightarrow B : N_A$$
$$B \longrightarrow A : Cert_B, N_A, N_B, A, [\![\{N_A, N_B, A\}]\!]_{\mathsf{pk}(K_B)}$$

N_B is used to not let Alice completely control what is signed by Bob (otherwise K_B cannot be used for anything else).

(ISO Public Key Two-Pass Unilateral Authentication Protocol)
Exercise. Where do **begin** and **end** go?

Mutual authentication — two unilateral authentications:

$$
\begin{aligned}
&1.\ A \longrightarrow B : N_{A1} \\
&2.\ B \longrightarrow A : Cert_B, N_{A1}, N_B, A, [\![\{N_{A1}, N_B, A\}]\!]_{\mathsf{pk}(K_B)} \\
&3.\ A \longrightarrow B : Cert_A, N_B, N_{A2}, B, [\![\{N_B, N_{A2}, B\}]\!]_{\mathsf{pk}(K_A)}
\end{aligned}
$$

A draft version of ISO Public Key Three-Pass Mutual Authentication Protocol.

- Simply two instances of the protocol on previous slide.
- Insecure.

$$
\begin{array}{rrl}
1. & C(A) \longrightarrow B & : N_{A1} \\
2. & B \longrightarrow C(A) & : Cert_B, N_{A1}, N_B, A, [\![\{N_{A1}, N_B, A\}]\!]_{\mathsf{pk}(K_B)} \\
1'. & C(B) \longrightarrow A & : N_B \\
2'. & A \longrightarrow C(B) & : Cert_A, N_B, N_{A2}, B, [\![\{N_B, N_{A2}, B\}]\!]_{\mathsf{pk}(K_A)} \\
3. & C(A) \longrightarrow B & : Cert_A, N_B, N_{A2}, B, [\![\{N_B, N_{A2}, B\}]\!]_{\mathsf{pk}(K_A)}
\end{array}
$$

B thinks he has been the responder in a protocol session with A. A does not think that she has initiated a session with B.

A variant with no such attacks:

$$
\begin{array}{l}
1.\ A \longrightarrow B : N_A \\
2.\ B \longrightarrow A : Cert_B, N_A, N_B, A, [\![\{N_A, N_B, A\}]\!]_{\mathsf{pk}(K_B)} \\
3.\ A \longrightarrow B : Cert_A, N_B, N_A, B, [\![\{N_B, N_A, B\}]\!]_{\mathsf{pk}(K_A)}
\end{array}
$$

But here B has a lot of control over the message signed by A.

Exercise. What if A and B were not under signature in messages 2 and 3?

$$
\begin{array}{lrl}
1. & A \longrightarrow C & : N_A \\
1'. & C(A) \longrightarrow B & : N_A \\
2'. & B \longrightarrow C(A) & : Cert_B, N_A, N_B, A, \{\!|N_A, N_B|\!\}_{\mathsf{pk}(K_B)} \\
2. & C \longrightarrow A & : Cert_C, N_A, N_B, A, \{\!|N_A, N_B|\!\}_{\mathsf{pk}(K_C)} \\
3. & A \longrightarrow C & : Cert_A, N_B, N_A, C, \{\!|N_B, N_A|\!\}_{\mathsf{pk}(K_A)} \\
3'. & C(A) \longrightarrow B & : Cert_A, N_B, N_A, B, \{\!|N_B, N_A|\!\}_{\mathsf{pk}(K_A)}
\end{array}
$$

B thinks he was the responder in a session initiated by A. A does not think she had initiated a session with B.

Entity authentication can be done using one-time passwords:
A and B have agreed on a code-book $f : \{0,1\}^n \longrightarrow \{0,1\}^*$.

1. A generates $r \in \{0,1\}^n$, sends it to B.
2. B responds with $f(r)$.
3. A checks that it indeed received $f(r)$.

Care has to be taken to not repeat the chellenge r.

Lamport's one-time password scheme:
Initialization: B chooses a password pw and $n \in \mathbb{N}$. Sends $(B, h^n(pw), n)$ to A over an authenticated channel.

- B puts $n_B := n$.
- A puts $pw' := h^n(pw)$.

One round:

1. A sends a notice to B.
2. B computes $r := h^{n_B - 1}(pw)$, decrements n_B and sends r to A.
3. A checks that $h(r) = pw'$ and puts $pw' := r$.

This works as long as A and B are synchronized. Resynchronization again requires authentic channels.

S/KEY one-time password scheme:
Initialization: B chooses a password pw and $n \in \mathbb{N}$. Sends $(B, h^n(pw), n)$ to A over an authenticated channel.

- A puts $n_A := n$.
- A puts $pw' := h^n(pw)$.

One round:

1. A sends the notice $n := n_A$ to B.
2. B computes $r := h^{n-1}(pw)$ and sends r to A.
3. A checks that $h(r) = pw'$, puts $pw' := r$ and $n_A := n - 1$.

Insecure. **Exercise.** Attack it.

We have seen Diffie-Hellman key exchange:
Let G be a group with hard Diffie-Hellman problem. Let g generate G. Let $m = |G|$.

1. A chooses a random $a \in \mathbb{Z}_m$, sends $x = g^a$ to B.
2. B chooses a random $b \in \mathbb{Z}_m$, sends $y = g^b$ to A.
3. A computes $K = y^a$. B computes $K = x^b$.
4. K is used as a common secret. ($h(K)$ may be a symmetric key)

This protocol needs authentication, too.

Station-to-station protocol:

$$
\begin{aligned}
A &\longrightarrow B : g^{N_A} \\
B &\longrightarrow A : g^{N_B}, Cert_B, \{[\![\{g^{N_B}, g^{N_A}\}]\!]_{K_B}\}_{g^{N_A N_B}} \\
A &\longrightarrow B : Cert_A, \{[\![\{g^{N_A}, g^{N_B}\}]\!]_{K_A}\}_{g^{N_A N_B}}
\end{aligned}
$$

Proposed by Diffie et al.

Aimed to have several security properties:

- Mutual entity authentication.
- Key agreement.
  - No third party knows the key.
- Key confirmation.
  - The other party knows the key.
- Perfect forward secrecy.

It does not quite achieve mutual authentication:

$$\begin{array}{rl} 1. & A \longrightarrow C(B) : g^{N_A} \\ 1'. & C \longrightarrow B \quad : g^{N_A} \\ 2'. & B \longrightarrow C \quad : g^{N_B}, Cert_B, \{[\![g^{N_B}, g^{N_A}]\!]_{K_B}\}_{g^{N_A N_B}} \\ 2. & C(B) \longrightarrow A \quad : g^{N_B}, Cert_B, \{[\![g^{N_B}, g^{N_A}]\!]_{K_B}\}_{g^{N_A N_B}} \\ 3. & A \longrightarrow C(B) : Cert_A, \{[\![g^{N_A}, g^{N_B}]\!]_{K_A}\}_{g^{N_A N_B}} \end{array}$$

At this point A thinks she was the initiator in a session with B. But B does not think he was a responder in a session with A.
The secrecy of $g^{N_A N_B}$ is not violated.
Identities of parties inside the signed messages would have helped.

Neumann-Stubblebine key exchange protocol.
A TTP T generates a new key for A and B.
Let K_{XT} be the (long-term) symmetric key shared by X and T.

$$
\begin{aligned}
&1.\ A \longrightarrow B : A, N_A \\
&2.\ B \longrightarrow T : B, N_B, \{A, N_A, T_B\}_{K_{BT}} \\
&3.\ T \longrightarrow A : N_B, \{B, N_A, K_{AB}, T_B\}_{K_{AT}}, \{A, K_{AB}, T_B\}_{K_{BT}} \\
&4.\ A \longrightarrow B : \{A, K_{AB}, T_B\}_{K_{BT}}, \{N_B\}_{K_{AB}}
\end{aligned}
$$

T_B is a timestamp.

A similarity:

1. $A \longrightarrow B : A, N_A$
2. $B \longrightarrow T : B, N_B, \{A, N_A, T_B\}_{K_{BT}}$
3. $T \longrightarrow A : N_B, \{B, N_A, K_{AB}, T_B\}_{K_{AT}}, \{A, K_{AB}, T_B\}_{K_{BT}}$
4. $A \longrightarrow B : \{A, K_{AB}, T_B\}_{K_{BT}}, \{N_B\}_{K_{AB}}$

Attack through a type flaw:

$$\begin{aligned}
&1.\ C(A) \longrightarrow B &&: A, N_A \\
&2.\ \qquad B \longrightarrow C(T) &&: B, N_B, \{A, N_A, T_B\}_{K_{BT}} \\
&4.\ C(A) \longrightarrow B &&: \{A, N_A, T_B\}_{K_{BT}}, \{N_B\}_{N_A}
\end{aligned}$$

where $N_A \in \mathbf{Keys}_{\mathrm{sym}} \cap \mathbf{Nonce}$.

B thinks he has agreed on key K_A with A. A has no idea.

Otway-Rees key exchange protocol:

1. $A \longrightarrow B : N, A, B, \{N_A, N, A, B\}_{K_{AT}}$
2. $B \longrightarrow T : N, A, B, \{N_A, N, A, B\}_{K_{AT}}, \{N_B, N, A, B\}_{K_{BT}}$
3. $T \longrightarrow B : \{N_A, K_{AB}\}_{K_{AT}}, \{N_B, K_{AB}\}_{K_{BT}}$
4. $B \longrightarrow A : \{N_A, K_{AB}\}_{K_{AT}}$

Possible type confusion:

$$
\begin{array}{l}
1.\ A \longrightarrow B : N, A, B, \{N_A, N, A, B\}_{K_{AT}} \\
2.\ B \longrightarrow T : N, A, B, \{N_A, N, A, B\}_{K_{AT}}, \{N_B, N, A, B\}_{K_{BT}} \\
3.\ T \longrightarrow B : \{N_A, K_{AB}\}_{K_{AT}}, \{N_B, K_{AB}\}_{K_{BT}} \\
4.\ B \longrightarrow A : \{N_A, K_{AB}\}_{K_{AT}}
\end{array}
$$

The triple (N, A, B) masquerading as a key may be from some old session.

Further reading:

Chapter 12.1–12.6 and 12.9 of

Menzeses, van Oorschot, Vanstone.
Handbook of Applied Cryptography.

(available on-line)