technische universiteit eindhoven

A Syntactic Criterion for Injectivity of Authentication Protocols

Sjouke Mauw

joint work with Cas Cremers and Erik de Vink

sjouke@win.tue.nl

Overview

- Motivation and problem statement.
- Formal model.
  - Security protocol.
  - Semantics.
  - Injectivity, authentication.
- Main theorem.
- Conclusions.

Two authentication properties

Agreement

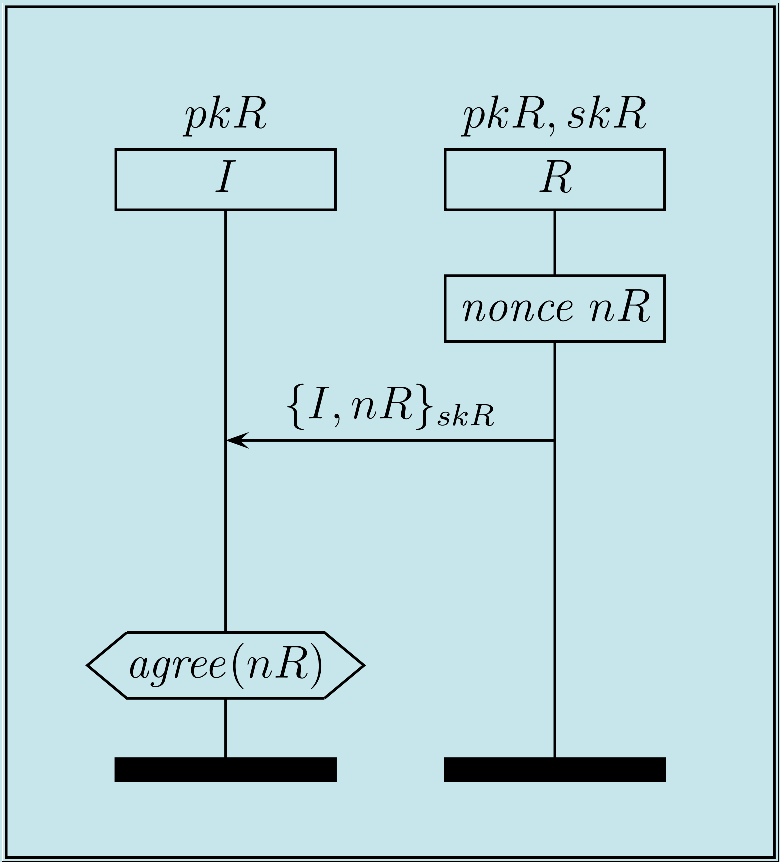
Upon successfully finishing a protocol session, parties agree on the values of (common) variables.

Synchronization

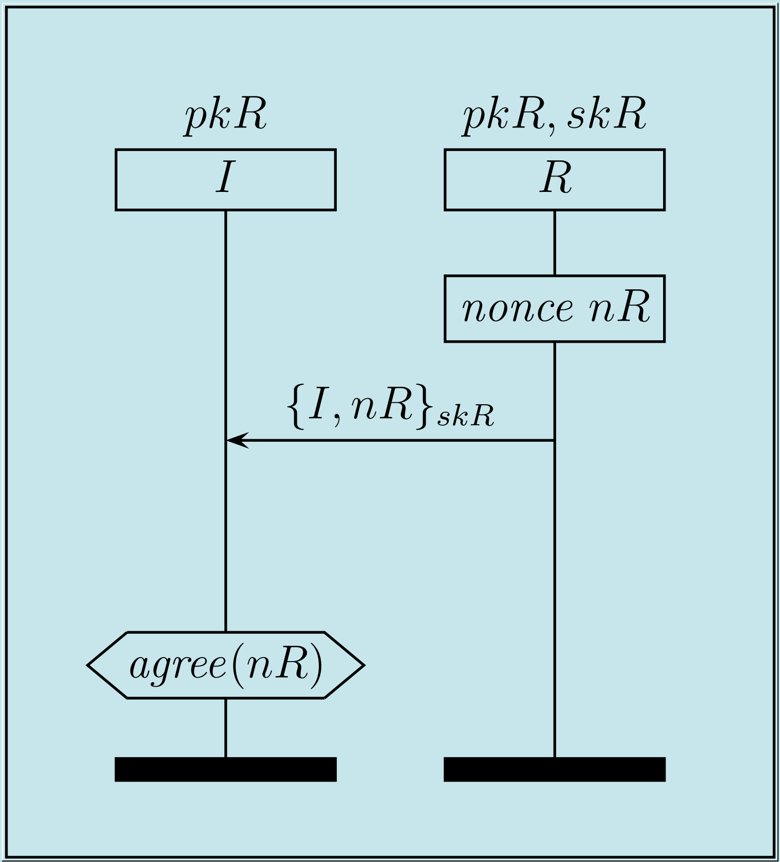
Upon successfully finishing a protocol session, all messages have been executed in intended order, with intended contents.

Synchronization is strictly stronger than agreement, but the differences are subtle.

Example: unilateral authentication protocol

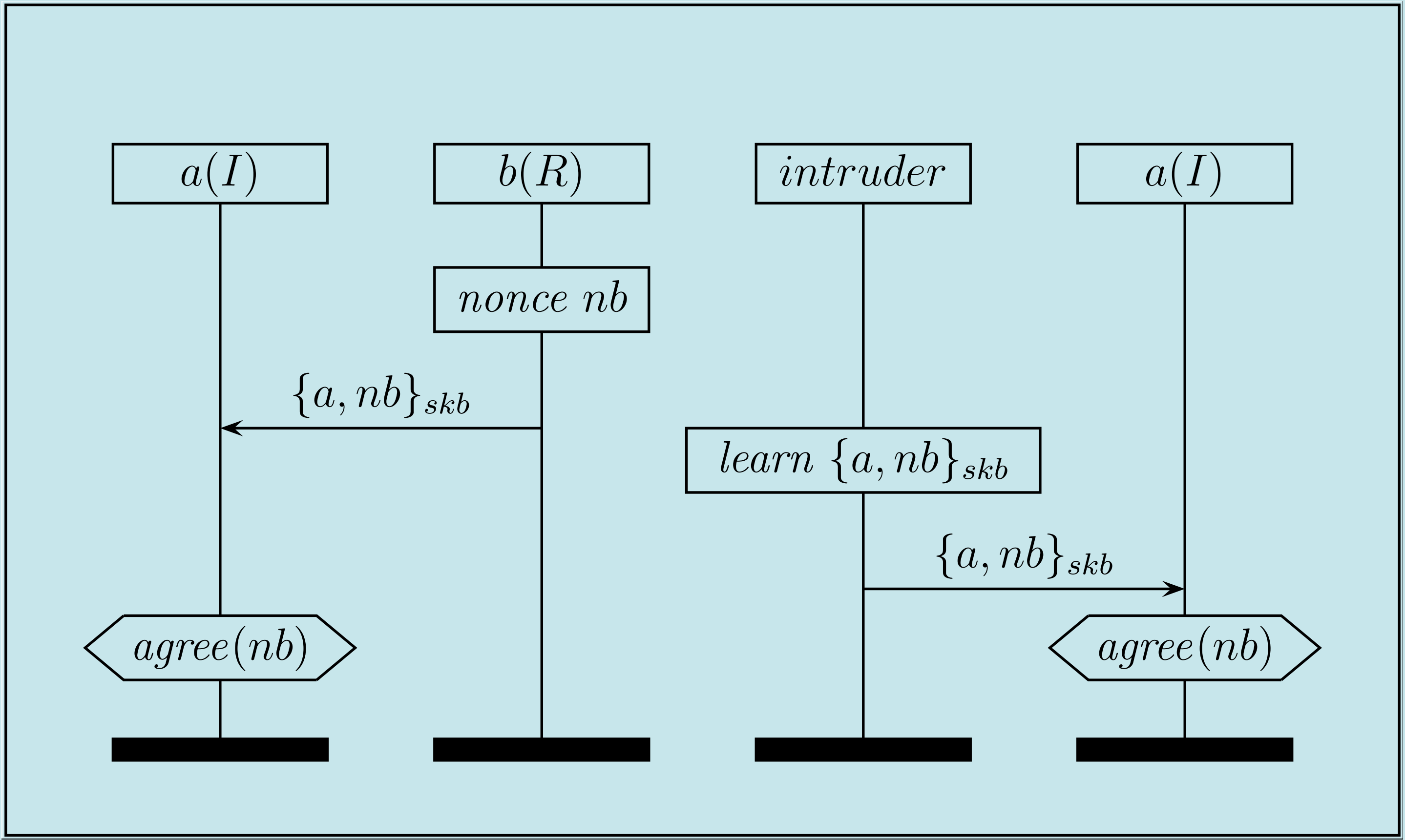


Example: unilateral authentication protocol

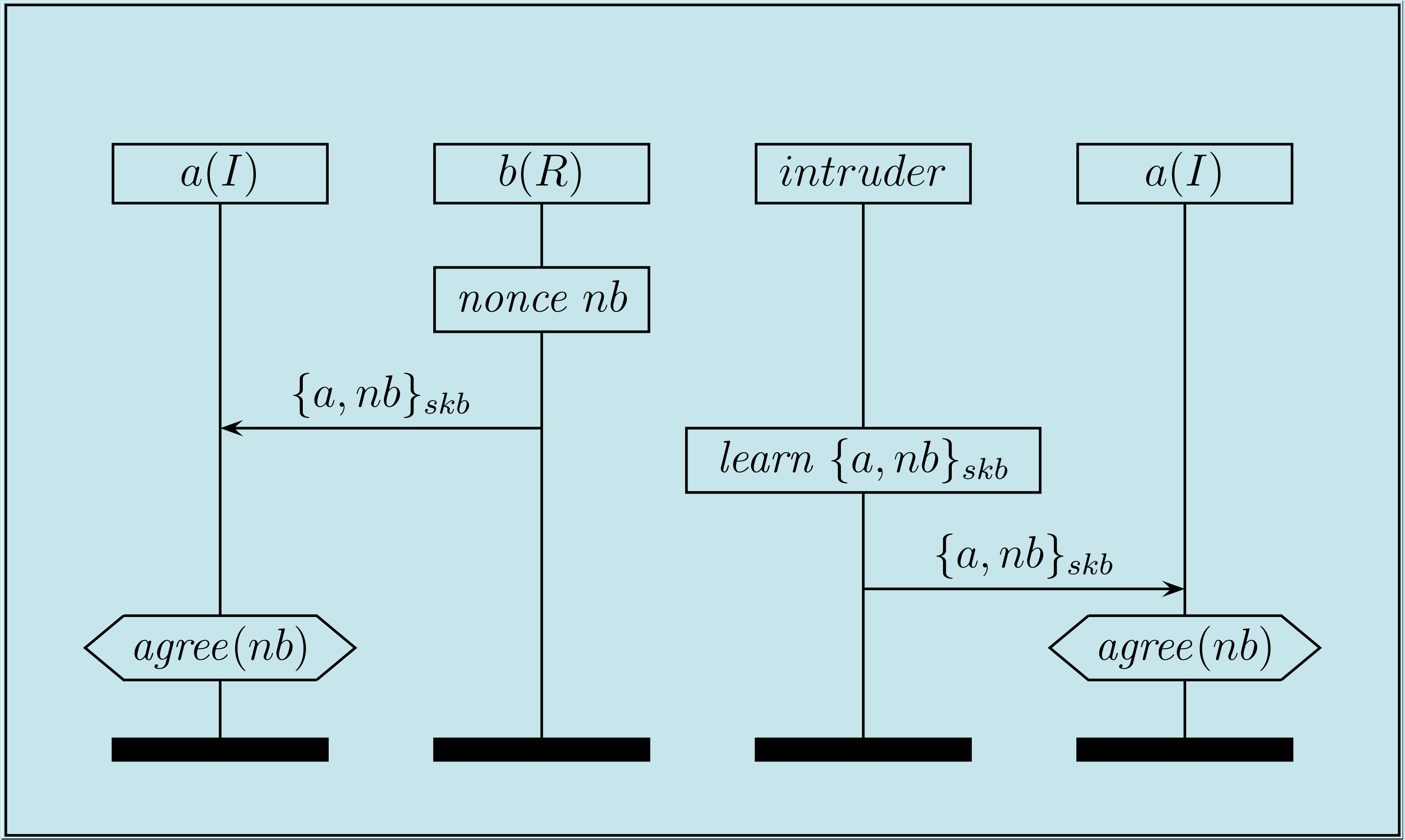


Question: Does this protocol satisfy agreement and/or synchronization?

A replay attack



A replay attack

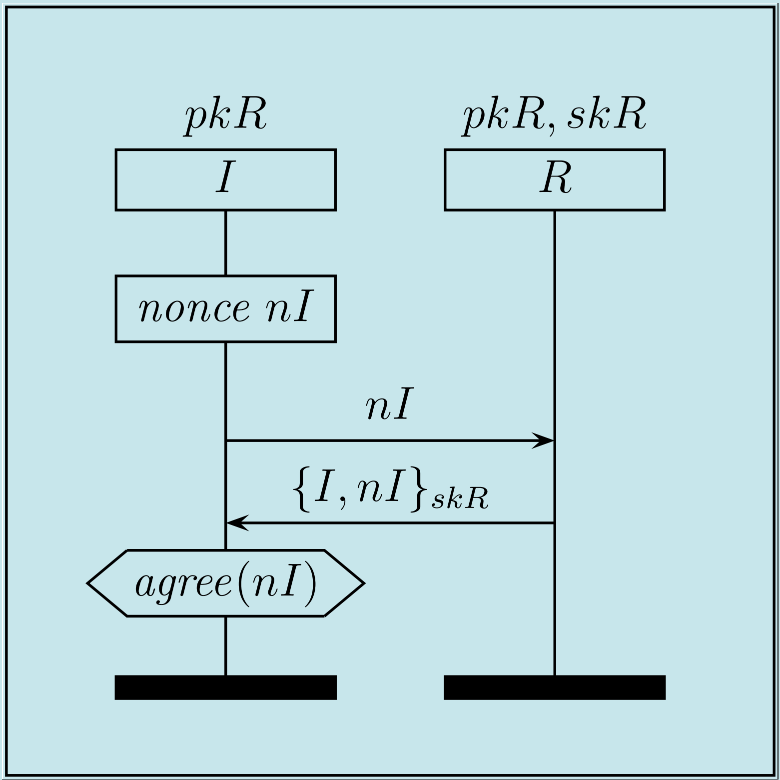


Question: How to fix this protocol?

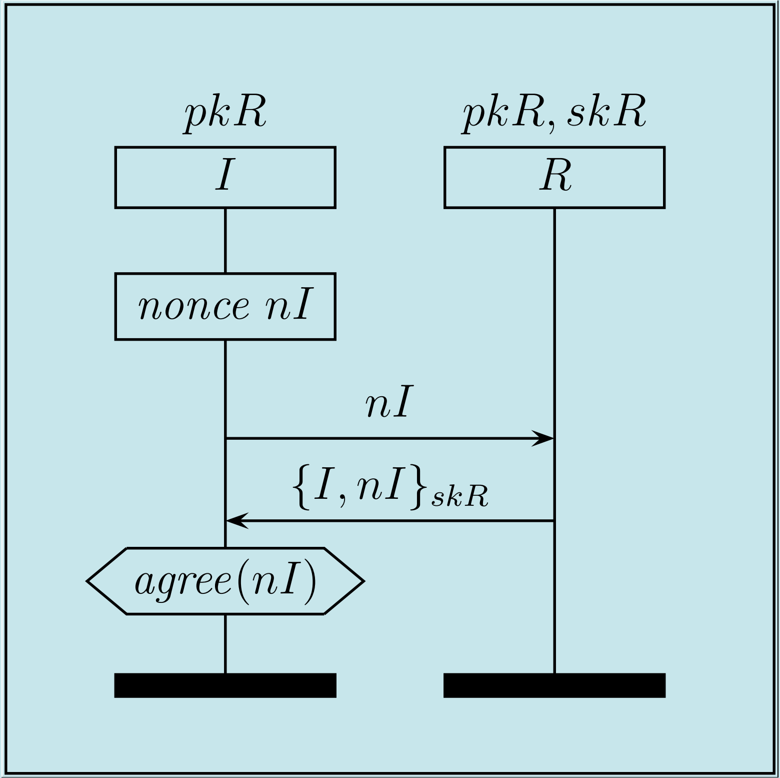
Fixed protocol should satisfy *injectivity*

Each run of an agent executing the initiator role corresponds to a *unique* run of its communication partner running the responder role.

Fixing the injectivity problem

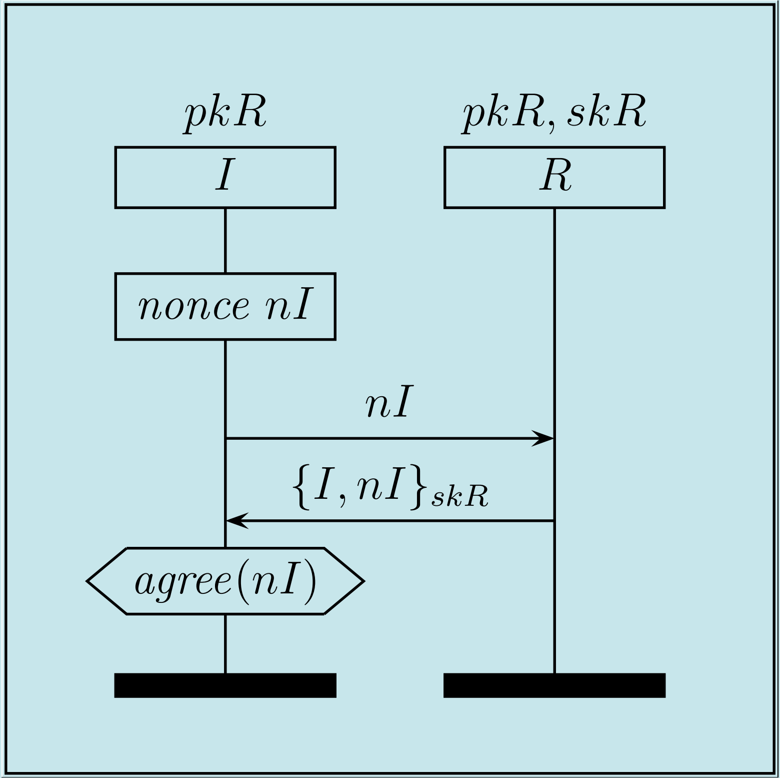


Fixing the injectivity problem



Question: What's the general idea behind this fix?

Fixing the injectivity problem



Question: What's the general idea behind this fix?

Answer 1: By letting I control the nonce.

Answer 2: By introducing a loop from I via R back to I.

How to verify injectivity?

model-checking approach

Counting: $\sharp(\text{I-runs}) \leq \sharp(\text{corresponding R-runs})$

other approaches (logics, term rewriting)

- Strand spaces: solicited authentication tests (Guttman, Theyer 2002)
- π-calculus: injective correspondence (Gordon, Jeffrey 2002)
- Logic: e-commerce protocol logic (Adi, Debbabi, Mejri 2003)
- Further: Ad-hoc reasoning, informal reasoning, or simply not.

Problem statement

Find a *generic* and *easy* way to validate injectivity for synchronizing protocols.

Generic:
As few assumptions on the security model as possible.

Easy:
Statically decidable.

The model (1): Protocol specification

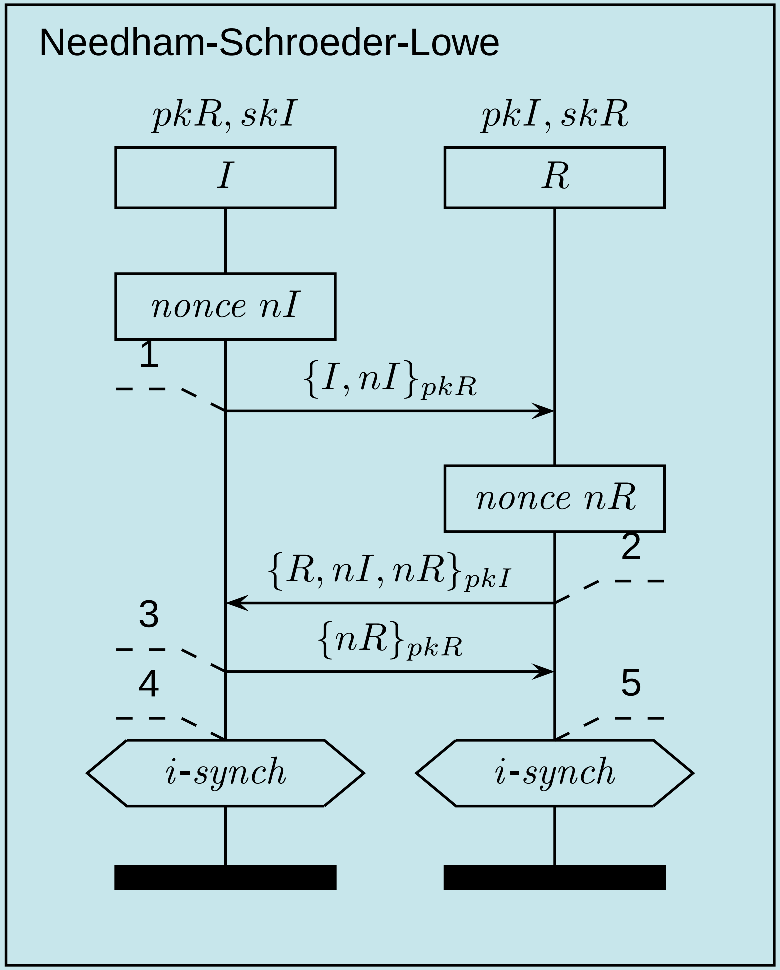
Events occurring in a protocol specification:

$$\begin{aligned} RoleEvent = \{\ & send_\ell(r, r', m), read_\ell(r, r', m), claim_\ell(r, c) \mid \\ & \ell \in Label, r, r' \in Role, m \in RoleMess, c \in Claim\ \} \end{aligned}$$

A protocol specification is a mapping from roles to lists of role events.

$$p \in Role \to RoleEvent^*$$

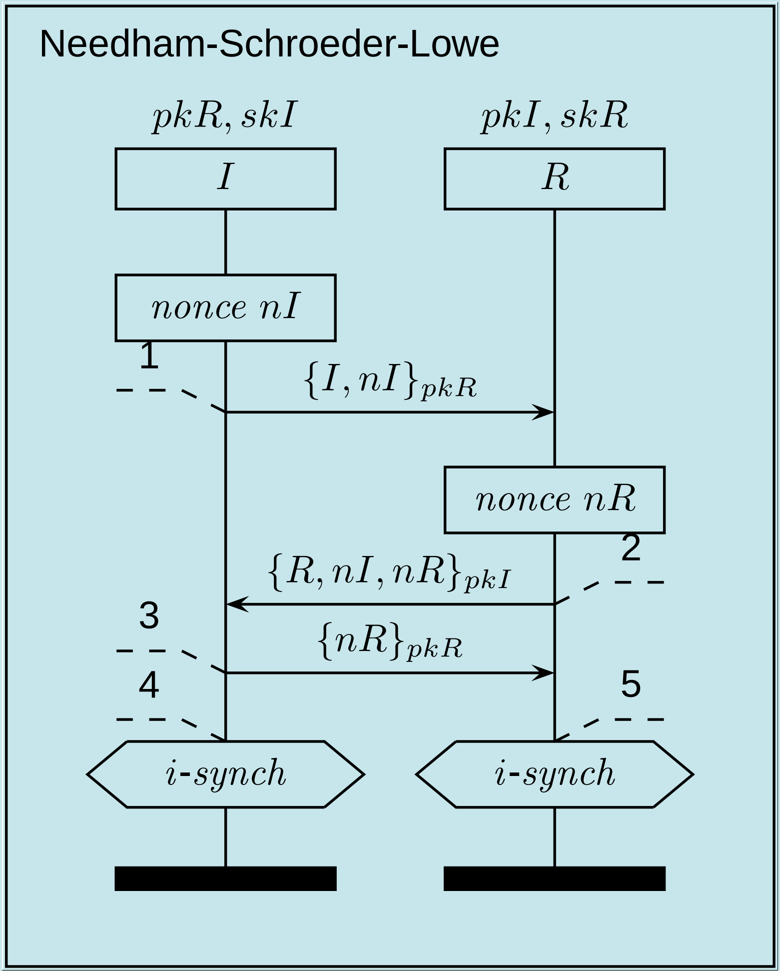
Example: NSL protocol



$$NSL(I) =$$
$$send_1(I, R, \{I, nI\}_{pkR})\cdot$$
$$read_2(R, I, \{R, nI, nR\}_{pkI})\cdot$$
$$send_3(I, R, \{nR\}_{pkR})\cdot$$
$$claim_4(I, i\text{-}synch)$$
$$NSL(R) =$$
$$read_1(I, R, \{I, nI\}_{pkR})\cdot$$
$$send_2(R, I, \{R, nI, nR\}_{pkI})\cdot$$
$$read_3(I, R, \{nR\}_{pkR})\cdot$$
$$claim_5(R, i\text{-}synch)$$

TU/e

The model (2): Causality preorder $\prec_p$



$$\begin{tikzcd}
send_1 \arrow[r] \arrow[d] & read_1 \arrow[d] \\
read_2 \arrow[d] & send_2 \arrow[l] \arrow[d] \\
send_3 \arrow[r] \arrow[d] & read_3 \arrow[d] \\
claim_4 & claim_5
\end{tikzcd}$$

The model (3): Semantics

- A *run* is the execution of a role by an agent.
- An agent may execute several (parallel) runs.
- Several agents may execute the same role in parallel.
- All runs have a unique *run identifier*.
- Executing a role event in a run gives a run event.

$$\begin{aligned} &RunEvent = \\ &\quad \{\, send_\ell(a,b,m)\sharp rid, read_\ell(a,b,m)\sharp rid, claim_\ell(a,c)\sharp rid \mid \\ &\quad rid \in RunId, \ell \in Label, a, b \in Agent, m \in RunMess, c \in Claim \,\} \end{aligned}$$

- An execution trace is a list of run events; the semantics of a protocol p is a set of traces.

$$Tr(p) \subseteq RunEvent^*$$

$Tr(p)$ closed under swapping

1. For $e' \neq read$ and $rid \neq rid'$

$$\alpha; e\sharp rid; e'\sharp rid'; \alpha' \in Tr(p) \;\Rightarrow\; \alpha; e'\sharp rid'; e\sharp rid; \alpha' \in Tr(p)$$

2. For $rid \neq rid'$

$$\alpha; send(m)\sharp rid''; \alpha'; e\sharp rid; read(m)\sharp rid'; \alpha'' \in Tr(p)$$

$$\Rightarrow\; \alpha; send(m)\sharp rid''; \alpha'; read(m)\sharp rid'; e\sharp rid; \alpha'' \in Tr(p)$$

Consequence
Let $\alpha \in Tr(p)$ and E be the set of events causally preceding a claim. Let α' be the trace obtained from α by shifting all events from E to the beginning of α. Then $\alpha' \in Tr(p)$.

The model (4): Authentication

NI-$SYNCH$ A protocol satisfies non-injective synchronization iff for every trace of the protocol there is an assignment of runs to roles such that the causal order of the protocol is respected and corresponding send and read events agree on the message sent.

I-$SYNCH$ A protocol satisfies injective synchronization if it satisfies non-injective synchronization and the assignment function is injective.

Preliminaries

- Assume protocol with just two roles, I and R.
- The I role has synchronization claim.
- Need function $partner : RunId \rightarrow RunId$
  $partner(rid1) = rid2$ means that $rid1$ executes the I-role, reaches the claim, while $rid2$ executes the corresponding R-role.

TU/e

Synchronization

$$
\begin{aligned}
&\mathit{NI\text{-}SYNCH} \iff \\
&\forall_{\alpha \in Tr(p)} \exists_{partner: RunId \to RunId} \forall_{i,rid}\ \alpha_i = claim\sharp rid \Rightarrow \\
&\qquad \forall_{read_\ell(I,R,t) \prec_p claim}\ \exists_{i,j,i<j,a,b \in Agent, m \in RunMess} \\
&\qquad\qquad \alpha_i = send_\ell(a,b,m)\sharp rid \wedge \\
&\qquad\qquad \alpha_j = read_\ell(a,b,m)\sharp partner(rid) \\
&\qquad \wedge\ \forall_{read_\ell(R,I,t) \prec_p claim}\ \exists_{i,j,i<j,a,b \in Agent, m \in RunMess} \\
&\qquad\qquad \alpha_i = send_\ell(a,b,m)\sharp partner(rid) \wedge \\
&\qquad\qquad \alpha_j = read_\ell(a,b,m)\sharp rid
\end{aligned}
$$

Synchronization

$$
\begin{aligned}
&\mathit{NI\text{-}SYNCH} \iff \\
&\forall_{\alpha \in Tr(p)} \exists_{partner: RunId \to RunId} \forall_{i,rid}\ \alpha_i = claim \sharp rid \Rightarrow \\
&\quad \forall_{read_\ell(I,R,t) \prec_p claim}\ \exists_{i,j,i<j,a,b \in Agent, m \in RunMess} \\
&\qquad \alpha_i = send_\ell(a,b,m) \sharp rid \wedge \\
&\qquad \alpha_j = read_\ell(a,b,m) \sharp partner(rid) \\
&\quad \wedge\ \forall_{read_\ell(R,I,t) \prec_p claim}\ \exists_{i,j,i<j,a,b \in Agent, m \in RunMess} \\
&\qquad \alpha_i = send_\ell(a,b,m) \sharp partner(rid) \wedge \\
&\qquad \alpha_j = read_\ell(a,b,m) \sharp rid
\end{aligned}
$$

$\mathit{I\text{-}SYNCH} \iff$ same, but $partner$ function should be injective

The $LOOP$ property

For all $e \prec_p claim$, such that $role(e) \neq role(claim)$ there exist e' and e'' such that

$$\begin{aligned}
&e' \prec_p e'' \prec_p claim \wedge \\
&role(e') = role(claim) \wedge \\
&role(e'') = role(e)
\end{aligned}$$

This property can be easily verified on the syntactic description of the protocol.

Main theorem

Given a *swap-closed* trace model, we have that

$$NI\text{-}SYNCH \wedge LOOP \Rightarrow I\text{-}SYNCH$$

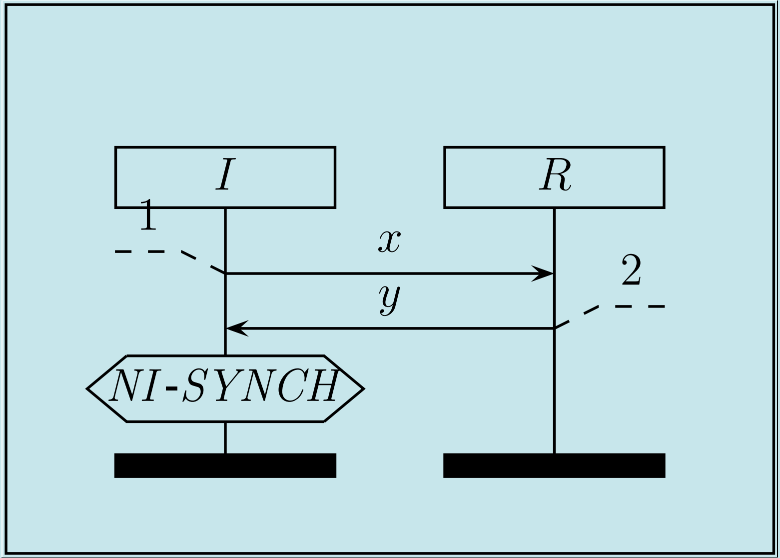
So, for synchronizing protocols, injectivity follows from the $LOOP$ property.

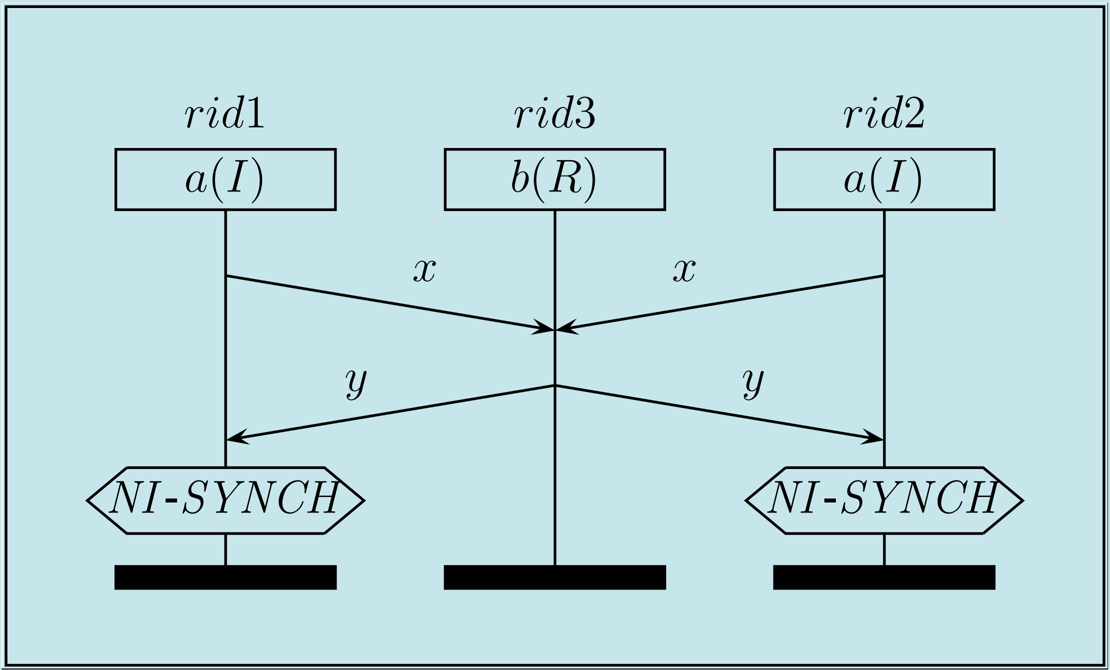
Proof sketch

Proof by contradiction.
Assume that

$$NI\text{-}SYNCH \wedge LOOP \wedge \neg I\text{-}SYNCH$$

Example protocol with $LOOP$





Transformation of trace α

$$
\begin{aligned}
\alpha = \\
&\alpha_0; \\
&send_1(x)\sharp rid1; \alpha_1; \\
&send_1(x)\sharp rid2; \alpha_2; \\
&read_1(x)\sharp rid3; \alpha_3; \\
&send_2(y)\sharp rid3; \alpha_4; \\
&read_2(y)\sharp rid1; \alpha_5; \\
&read_2(y)\sharp rid2; \alpha_6; \\
&claim_3\sharp rid1; \alpha_7; \\
&claim_3\sharp rid2; \\
&\alpha_8;
\end{aligned}
$$

Transformation of trace α

$$\begin{array}{ll}
\alpha = & \alpha' = \\
\quad \alpha_0; & \quad send_1(x)\sharp rid1; \\
\quad send_1(x)\sharp rid1; \alpha_1; & \quad send_1(x)\sharp rid2; \\
\quad send_1(x)\sharp rid2; \alpha_2; & \quad read_1(x)\sharp rid3; \\
\quad read_1(x)\sharp rid3; \alpha_3; & \quad send_2(y)\sharp rid3; \\
\quad send_2(y)\sharp rid3; \alpha_4; & \quad read_2(y)\sharp rid1; \\
\quad read_2(y)\sharp rid1; \alpha_5; & \quad read_2(y)\sharp rid2; \\
\quad read_2(y)\sharp rid2; \alpha_6; & \quad claim_3\sharp rid1; \\
\quad claim_3\sharp rid1; \alpha_7; & \quad claim_3\sharp rid2; \\
\quad claim_3\sharp rid2; & \quad \alpha_0; \alpha_1; \alpha_2; \alpha_3; \alpha_4; \\
\quad \alpha_8; & \quad \alpha_5; \alpha_6; \alpha_7; \alpha_8;
\end{array}$$

α': swap events preceding $claim_3\sharp rid1$ and $claim_3\sharp rid2$ to the beginning

Transformation of trace α

$\alpha =$	$\alpha' =$	$\alpha'' =$
$\alpha_0;$	$send_1(x)\sharp rid1;$	$send_1(x)\sharp rid1;$
$send_1(x)\sharp rid1; \alpha_1;$	$send_1(x)\sharp rid2;$	$read_1(x)\sharp rid3;$
$send_1(x)\sharp rid2; \alpha_2;$	$read_1(x)\sharp rid3;$	$send_2(y)\sharp rid3;$
$read_1(x)\sharp rid3; \alpha_3;$	$send_2(y)\sharp rid3;$	$read_2(y)\sharp rid1;$
$send_2(y)\sharp rid3; \alpha_4;$	$read_2(y)\sharp rid1;$	$claim_3\sharp rid1;$
$read_2(y)\sharp rid1; \alpha_5;$	$read_2(y)\sharp rid2;$	$send_1(x)\sharp rid2;$
$read_2(y)\sharp rid2; \alpha_6;$	$claim_3\sharp rid1;$	$read_2(y)\sharp rid2;$
$claim_3\sharp rid1; \alpha_7;$	$claim_3\sharp rid2;$	$claim_3\sharp rid2;$
$claim_3\sharp rid2;$	$\alpha_0; \alpha_1; \alpha_2; \alpha_3; \alpha_4;$	$\alpha_0; \alpha_1; \alpha_2; \alpha_3; \alpha_4;$
$\alpha_8;$	$\alpha_5; \alpha_6; \alpha_7; \alpha_8;$	$\alpha_5; \alpha_6; \alpha_7; \alpha_8;$

α'': next, swap events preceding $claim_3\sharp rid1$ to the beginning

Transformation of trace α

$$
\begin{array}{lll}
\alpha = & \alpha' = & \alpha'' = \\
\quad \alpha_0; & \quad send_1(x)\sharp rid1; & \quad send_1(x)\sharp rid1; \\
\quad send_1(x)\sharp rid1; \alpha_1; & \quad send_1(x)\sharp rid2; & \quad read_1(x)\sharp rid3; \\
\quad send_1(x)\sharp rid2; \alpha_2; & \quad read_1(x)\sharp rid3; & \quad send_2(y)\sharp rid3; \\
\quad read_1(x)\sharp rid3; \alpha_3; & \quad send_2(y)\sharp rid3; & \quad read_2(y)\sharp rid1; \\
\quad send_2(y)\sharp rid3; \alpha_4; & \quad read_2(y)\sharp rid1; & \quad claim_3\sharp rid1; \\
\quad read_2(y)\sharp rid1; \alpha_5; & \quad read_2(y)\sharp rid2; & \quad send_1(x)\sharp rid2; \\
\quad read_2(y)\sharp rid2; \alpha_6; & \quad claim_3\sharp rid1; & \quad read_2(y)\sharp rid2; \\
\quad claim_3\sharp rid1; \alpha_7; & \quad claim_3\sharp rid2; & \quad claim_3\sharp rid2; \\
\quad claim_3\sharp rid2; & \quad \alpha_0; \alpha_1; \alpha_2; \alpha_3; \alpha_4; & \quad \alpha_0; \alpha_1; \alpha_2; \alpha_3; \alpha_4; \\
\quad \alpha_8; & \quad \alpha_5; \alpha_6; \alpha_7; \alpha_8; & \quad \alpha_5; \alpha_6; \alpha_7; \alpha_8;
\end{array}
$$

- $\alpha \in Tr(p) \Rightarrow \alpha' \in Tr(p) \Rightarrow \alpha'' \in Tr(p)$

Transformation of trace α

$\alpha =$	$\alpha' =$	$\alpha'' =$
$\alpha_0;$	$send_1(x)\sharp rid1;$	$send_1(x)\sharp rid1;$
$send_1(x)\sharp rid1; \alpha_1;$	$send_1(x)\sharp rid2;$	$read_1(x)\sharp rid3;$
$send_1(x)\sharp rid2; \alpha_2;$	$read_1(x)\sharp rid3;$	$send_2(y)\sharp rid3;$
$read_1(x)\sharp rid3; \alpha_3;$	$send_2(y)\sharp rid3;$	$read_2(y)\sharp rid1;$
$send_2(y)\sharp rid3; \alpha_4;$	$read_2(y)\sharp rid1;$	$claim_3\sharp rid1;$
$read_2(y)\sharp rid1; \alpha_5;$	$read_2(y)\sharp rid2;$	$send_1(x)\sharp rid2;$
$read_2(y)\sharp rid2; \alpha_6;$	$claim_3\sharp rid1;$	$read_2(y)\sharp rid2;$
$claim_3\sharp rid1; \alpha_7;$	$claim_3\sharp rid2;$	$claim_3\sharp rid2;$
$claim_3\sharp rid2;$	$\alpha_0; \alpha_1; \alpha_2; \alpha_3; \alpha_4;$	$\alpha_0; \alpha_1; \alpha_2; \alpha_3; \alpha_4;$
$\alpha_8;$	$\alpha_5; \alpha_6; \alpha_7; \alpha_8;$	$\alpha_5; \alpha_6; \alpha_7; \alpha_8;$

- $\alpha \in Tr(p) \Rightarrow \alpha' \in Tr(p) \Rightarrow \alpha'' \in Tr(p)$
- Because we assumed $NI\text{-}SYNCH$, we have that run $rid2$ of α'' must synchronize. This cannot be the case. Contradiction.

Conclusions

- Loop-property can be checked easily.
- Sufficient condition for large class of security protocol semantics.
- Necessary condition for standard Dolev-Yao intruder.
- Loop plus agreement not sufficient to imply injective agreement.
- Generalizes easily to multi-party protocols with multiple claims.

Loop plus agreement not sufficient