Special topics in Logic and Security I

Master Year II, Sem. I, 2025-2026

Ioana Leuştean
FMI, UB

Formal analysis of security protocols

- In formal analysis we define and analyze a protocol within a consistent mathematical theory.
- One studies abstract versions of real protocols (for example the (real, computer-network authentication) protocol Kerberos is based on the (academic) protocol Needham-Scroeder.
- Verification based on:
  - epistemic logics, for example BAN logic, 1990
  - model-checking tools: Proverif, AVISPA, Scyther, Tamarin, ...
  - ...

In the following we shall present the *operational semantics approach* from [OSVSP]:

- Cremers C. and Mauw S. Operational Semantics and Verification of Security Protocols. Springer, 2012.
- Cremers, C. J. F. (2006). Scyther : semantics and verification of security protocols Eindhoven: Technische Universiteit Eindhoven DOI: 10.6100/IR614943

Operational semantics for programming languages

- Formal programming languages semantics:
  - natural language,
  - axiomatic,
  - denotational,
  - operational
- The operational semantics represents the execution of a program as a sequence of computational steps, formally defined by a transition system between configurations

$$\langle code\ ,\ \sigma\rangle \rightarrow \langle code\ ,\ \sigma'\rangle$$

where σ and σ' are states. For a simple programming language, states are defined as partial functions from variables to values:

$$\sigma : Var \rightharpoonup Int$$

$$\sigma = x_1 \mapsto v_1, \ldots, x_n \mapsto v_n$$

Operational semantics for programming languages

- Language

```
E ::= n | x | E + E
C ::= x=E;|C C|
      { C } | {}
P ::= int x = n ; P | C
```

- Rules for transitions:

$$\langle int\ x = i; p, \sigma\rangle \rightarrow \langle int\ p, \sigma_{x\leftarrow i}\rangle$$

$$\frac{\langle e_1, \sigma\rangle \rightarrow \langle e_1', \sigma\rangle}{\langle e_1 + e_2, \sigma\rangle \rightarrow \langle e_1' + e_2, \sigma\rangle}$$

- The program execution is a sequence of transitions:

$$\begin{array}{lcl} \langle x = 0; x = x + 1;\ ,\ \perp\rangle & \rightarrow & \langle x = x + 1;\ ,\ x \mapsto 0\rangle \\ & \rightarrow & \langle x = 0 + 1;\ ,\ x \mapsto 0\rangle \\ & \rightarrow & \langle x = 1;\ ,\ x \mapsto 0\rangle \\ & \rightarrow & \langle \{\}\ ,\ x \mapsto 1\rangle \end{array}$$

Formal analysis of security protocols

- A formal approach should:
  - specify the security protocol,
  - provide a model for agents,
  - provide a model for communication,
  - provide a threat model,
  - express the security requirements.

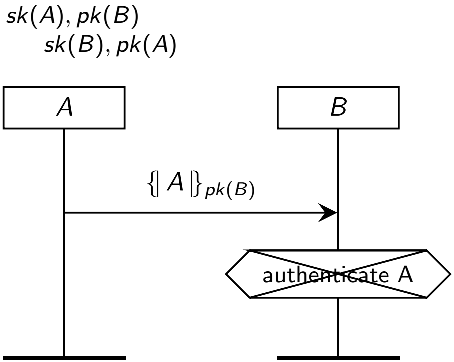
- In the following, we shall use a *many-sorted language*:
  - the roles and the messages are represented by terms,
  - a protocol is defined by specifying each role,
  - when a protocol is executed, an agent can play any role, one or more times,
  - a single execution of a role is called a run,
  - the (adversary) knowledge is derived using a deduction system,
  - the security properties are formally defined and analyzed.

The operational semantics of a security protocol is defined as a (complex) transition system.

Example: Simple Protocol

What does authentication mean?

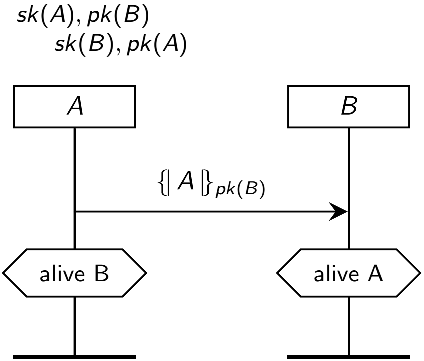
> "In its most basic form, authentication is a simple statement about the existence of communication partner.
> [...]
> *Aliveness* is a form of authentication that aims to establish that an intended communication partner has executed some event."



[OVSP] Cremers C. and Mauw S. Operational Semantics and Verification of Security Protocols. Springer, 2012.

Scyther: Simple Protocol

```
protocol simple(I,R)
{
role I
   {
     send_1(I,R, {I}pk(R) );

     claim(I,Alive);
    }
role R
    {
      recv_1(I,R, {I}pk(R));

      claim(R,Alive);
     }
}
```



The claims are locally analyzed!

Scyther: Simple Protocol

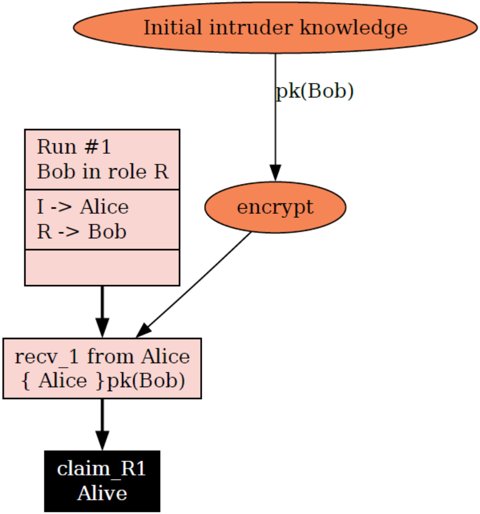
```
protocol simple(I,R)
{role I
   {send_1(I,R, {I}pk(R) );
    claim(I,Alive);}
role R
    {recv_1(I,R, {I}pk(R));
     claim(R,Alive);
     }}
```

Scyther results : verify

Claim				Status		Comments	Pattern
simple	I	simple,I1	Alive	Fail	Falsified	Exactly 1 attack.	1 attack
	R	simple,R1	Alive	Fail	Falsified	Exactly 1 attack.	1 attack

Done.

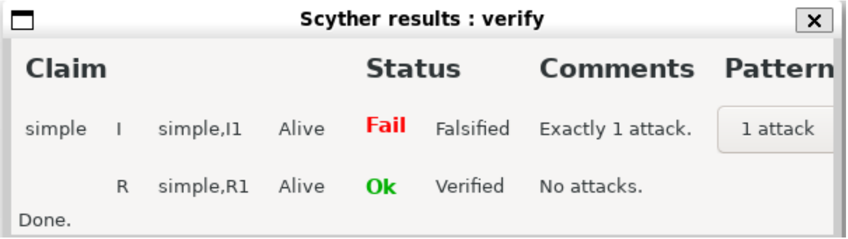
Scyther: Simple Protocol



[Id 2] Protocol simple, role R, claim type Alive

Scyther: Simple Protocol

```
protocol simple(I,R)
{role I
   {send_1(I,R, {I}sk(I) );
    claim(I,Alive);}
role R
    {recv_1(I,R, {I}sk(I));
     claim(R,Alive);
     }}
```

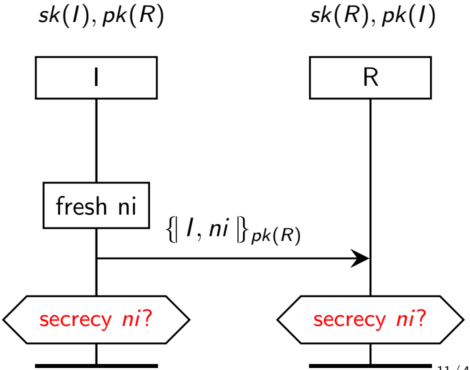


Example: OSS Protocol

One-Sided Secrecy Protocol (OSS)

$I \longrightarrow R : \{\!| I, ni |\!\}_{pk(R)}$

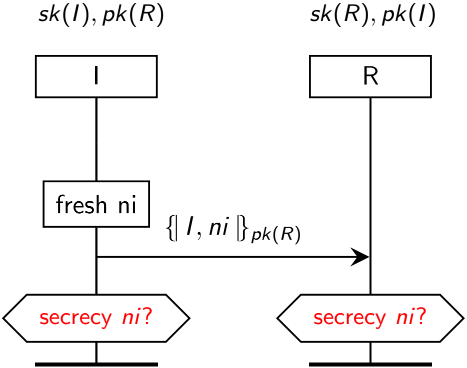
What does secrecy mean?



Example: OSS Protocol

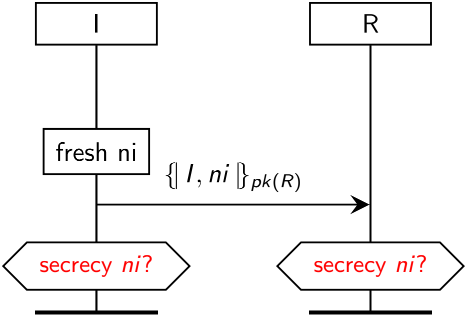
What does secrecy ni mean?

[OSVSP]
We say that the claim secrecy ni holds *for a role* whenever a run of the role is completed with trusted communication partners and the nonce *ni* generated in the run does not become known to the intruder



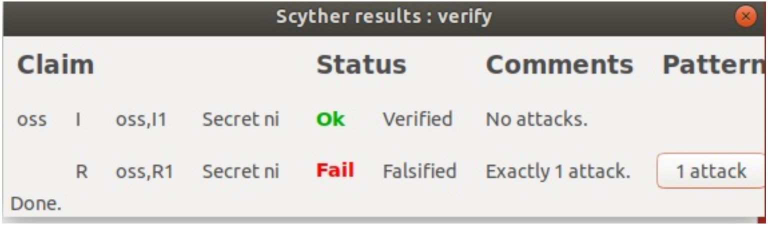
Scyther: OSS Protocol

```
protocol oss(I,R)
{role I
{fresh ni: Nonce;
send_1(I,R, {I,ni}pk(R) );
claim(I,Secret,ni);}

role R
{var ni: Nonce;
recv_1(I,R, {I,ni}pk(R) );
claim(R,Secret,ni);
}}
```

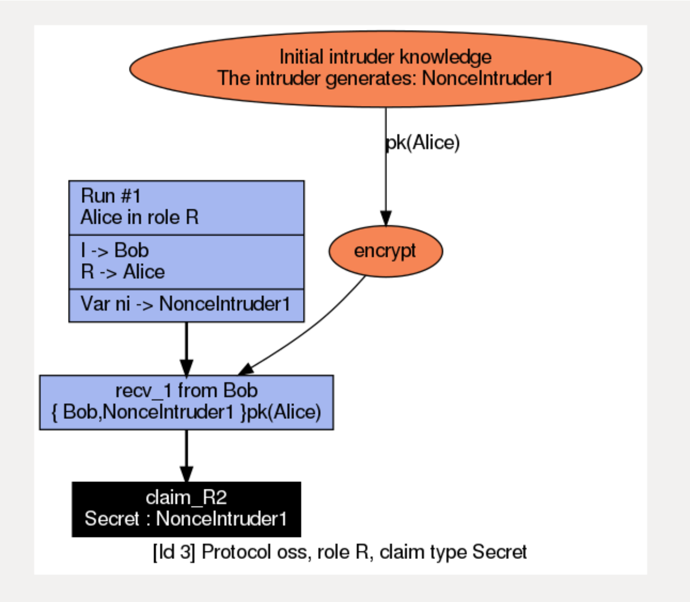


Scyther: OSS Protocol

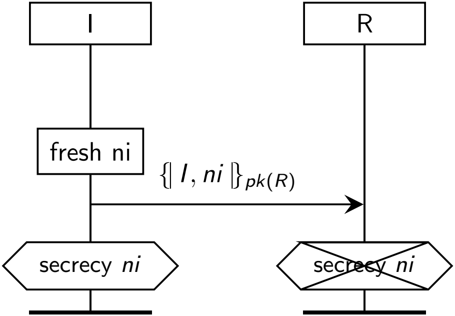
```
protocol oss(I,R)
{role I
{fresh ni: Nonce;
send_1(I,R, {I,ni}pk(R) );
claim(I,Secret,ni);}

role R
{var ni: Nonce;
recv_1(I,R, {I,ni}pk(R));
claim(R,Secret,ni);
}}
```



Scyther: OSS Protocol



Scyther: OSS Protocol



Scyther: Otway-Rees Protocol

Step 1.	$A \longrightarrow B:$	$M, A, B, \{N_A, M, A, B\}_{K_{AS}}$
Step 2.	$B \longrightarrow S:$	$M, A, B, \{N_A, M, A, B\}_{K_{AS}}, \{N_B, M, A, B\}_{K_{BS}}$
Step 3.	$S \longrightarrow B:$	$M, \{N_A, K_{AB}\}_{K_{AS}}, \{N_B, K_{AB}\}_{K_{BS}}$
Step 4.	$B \longrightarrow A:$	$M, \{N_A, K_{AB}\}_{K_{AS}}$

```
protocol otwayrees(I,R,S)
role I
    {fresh Ni : Nonce;
     fresh M : String;
     var Kir : SessionKey;

     send_1(I,R, M,I,R,{Ni,M,I,R}k(I,S) );
     recv_4(R,I, M,{Ni,Kir}k(I,S) );

     claim_I1(I, Secret,Kir);
     claim_I2(I, Nisynch);}
```

`Nisynch` means that everything happens as it should (in one round).

Scyther: Otway-Rees Protocol

Step 1. $A \longrightarrow B: \quad M, A, B, \{N_A, M, A, B\}_{K_{AS}}$
Step 2. $B \longrightarrow S: \quad M, A, B, \{N_A, M, A, B\}_{K_{AS}}, \{N_B, M, A, B\}_{K_{BS}}$
Step 3. $S \longrightarrow B: \quad M, \{N_A, K_{AB}\}_{K_{AS}}, \{N_B, K_{AB}\}_{K_{BS}}$
Step 4. $B \longrightarrow A: \quad M, \{N_A, K_{AB}\}_{K_{AS}}$

```
protocol otwayrees(I,R,S)
role R
    {    var M : String;
        fresh Nr : Nonce;
        var Kir : SessionKey;
        var T1,T2: Ticket;

        recv_1(I,R, M,I,R, T1 );
        send_2(R,S, M,I,R, T1, { Nr,M,I,R }k(R,S) );
        recv_3(S,R, M, T2, { Nr,Kir }k(R,S) );
        send_4(R,I, M, T2 );

        claim_R1(R, Secret,Kir);
        claim_R2(R, Nisynch);
    }
```

Scyther: Otway-Rees Protocol

Step 1.	$A \longrightarrow B:$	$M, A, B, \{N_A, M, A, B\}_{K_{AS}}$
Step 2.	$B \longrightarrow S:$	$M, A, B, \{N_A, M, A, B\}_{K_{AS}}, \{N_B, M, A, B\}_{K_{BS}}$
Step 3.	$S \longrightarrow B:$	$M, \{N_A, K_{AB}\}_{K_{AS}}, \{N_B, K_{AB}\}_{K_{BS}}$
Step 4.	$B \longrightarrow A:$	$M, \{N_A, K_{AB}\}_{K_{AS}}$

```
protocol otwayrees(I,R,S)
role S
    {
      var Ni,Nr : Nonce;
      var M : String;
      fresh Kir : SessionKey;

      recv_2(R,S, M,I,R, { Ni,M,I,R}k(I,S), { Nr,M,I,R }k(R,S) );
      send_3(S,R, M, { Ni,Kir }k(I,S) , { Nr,Kir }k(R,S) );
    }
```

Scyther: Otway-Rees Protocol

```
role I
    {fresh Ni : Nonce; fresh M : String; var Kir : SessionKey;

     send_1(I,R, M,I,R,{Ni,M,I,R}k(I,S) );
     recv_4(R,I, M,{Ni,Kir}k(I,S) );

     claim_I1(I, Secret,Kir); claim_I2(I, Nisync)}

role R
    {  var M : String; fresh Nr : Nonce; var Kir : SessionKey; var T1,T2: Ticket;

        recv_1(I,R, M,I,R, T1 );
        send_2(R,S, M,I,R, T1, { Nr,M,I,R }k(R,S) );
        recv_3(S,R, M, T2, { Nr,Kir }k(R,S) );
        send_4(R,I, M, T2 );

        claim_R1(R, Secret,Kir); claim_R2(R, Nisynch); }
```

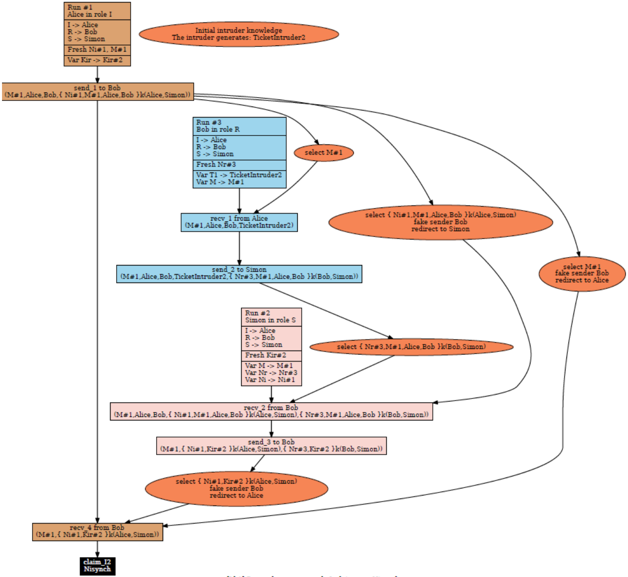
Scyther results : verify

Claim				Status		Comments	Patterns
otwayrees	I	otwayrees,I1	Secret Kir	Ok		No attacks within bounds.	
		otwayrees,I2	Nisynch	Fail	Falsified	At least 1 attack.	1 attack
	R	otwayrees,R1	Secret Kir	Ok		No attacks within bounds.	
		otwayrees,R2	Nisynch	Fail	Falsified	At least 1 attack.	1 attack

Done

Scyther: Otway-Rees Protocol

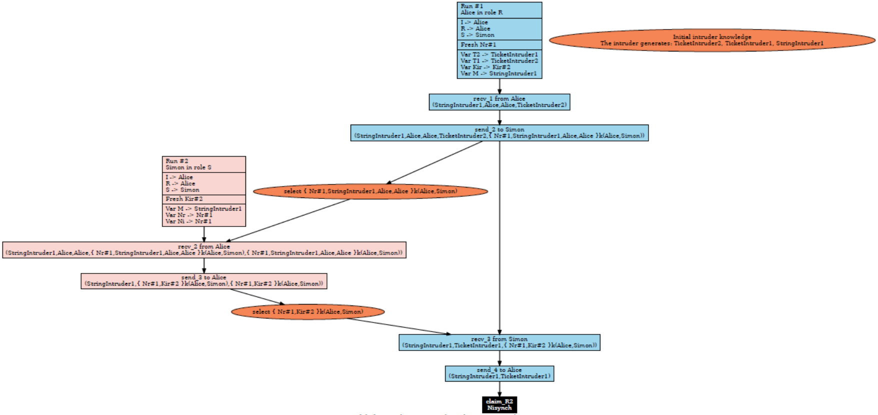
Protocol otwayrees, role I, claim type Nisynch



[Id 1] Protocol otwayrees, role I, claim type Nisynch

Scyther: Otway-Rees Protocol

Protocol otwayrees, role R, claim type Nisynch



[Id 2] Protocol otwayrees, role R, claim type Nisynch

Protocol specification

Basic elements

- There are few types of variables and constants:
  - *Var* : $V, X, Y, Z, \ldots$ (variables for messages),
  - *Fresh* : $ni, nr, sessionkey, \ldots$ (local constants, freshly generated for each instance of a role),
  - *Role* : $i, r, s, \ldots$ (variables for roles: initiator, respondent, server etc.)

- *Func* is a set of function symbols
  - each symbol has a fixed arity;
  - the global constants are functions of arity zero;
  - example: hash functions.

Terms: *RoleTerm*

We use *role terms to specify: messages, nonces, roles, keys.*

$$
\begin{aligned}
\mathit{RoleTerm} \quad ::= \quad & \mathit{Var} \mid \mathit{Fresh} \mid \mathit{Role} \\
& \mid \mathit{Func}\,(\mathit{RoleTerm}^{*}) \\
& \mid (\mathit{RoleTerm}, \mathit{RoleTerm}) \\
& \mid \{\!|\, \mathit{RoleTerm} \,|\!\}_{\mathit{RoleTerm}} \\
& \mid \mathit{sk}(\mathit{RoleTerm}) \mid \mathit{pk}(\mathit{RoleTerm}) \mid k(\mathit{RoleTerm}, \mathit{RoleTerm})
\end{aligned}
$$

We denote $\{\!|\,(t_1, t_2)\,|\!\}$ *by* $\{\!|\, t_1, t_2 \,|\!\}$.

The inverse function. Encryption

$$^{-1} : \mathit{RoleTerm} \to \mathit{RoleTerm}$$

- for any term $rt \in \mathit{RoleTerm}$ we define *the inverse* $rt^{-1} \in \mathit{RoleTerm}$ as follows:

$$rt^{-1} = \begin{cases} sk(t) & \text{if } rt = pk(t) \text{ for some } t \in \mathit{RoleTerm} \\ pk(t) & \text{if } rt = sk(t) \text{ for some } t \in \mathit{RoleTerm} \\ rt & \mathit{otherwise} \end{cases}$$

We further assume that sk and pk are the secret and, respectively, the public key for asymmetric encryption, while k defines a symmetric key.

Example: digital signature

Assume that $m \in \mathit{RoleTerm}$ is a message and $R \in \mathit{Role}$ is a role.

- $\{\!|\, m \,|\!\}_{sk(R)}$ is the encryption of m with the secret key of R
- a digitally signed message is a pair

$$(m, \{\!|\, h(m) \,|\!\}_{sk(R)})$$

  where
  - $h(m)$ is *the message digest*, computed using the hash function h,
  - $\{\!|\, h(m) \,|\!\}_{sk(R)}$, the signed message digest, is computed with the signer's private key $sk(R)$,
  - the message in plain text together with the signed message digest form the digitally signed message.
- One can verify the digital signature, in order to prove who the signer was and the integrity of the document: find the message digest using the signer's public key and compare the message digest with the result obtained by applying the hash function to the plain message.

Deduction on *RoleTerm*

$$\vdash \subseteq \mathcal{P}(RoleTerm) \times RoleTerm$$

$M \vdash t$ means that t can be deduced knowing M

$\vdash$ is the least relation with the following properties:

if	$t \in M$	then	$M \vdash t$
if	$M \vdash t_1$ and $M \vdash t_2$	then	$M \vdash (t_1, t_2)$
if	$M \vdash (t_1, t_2)$	then	$M \vdash t1$ and $M \vdash t_2$
if	$M \vdash t$ and $M \vdash k$	then	$M \vdash \{\!\mid t \mid\!\}_k$
if	$M \vdash \{\!\mid t \mid\!\}_k$ and $M \vdash k^{-1}$	then	$M \vdash t$
if	$M \vdash t_1$ and $\ldots$ and $M \vdash t_n$	then	$M \vdash f\,(t_1, \ldots, t_n)$

where n is the arity of $f \in Func$.

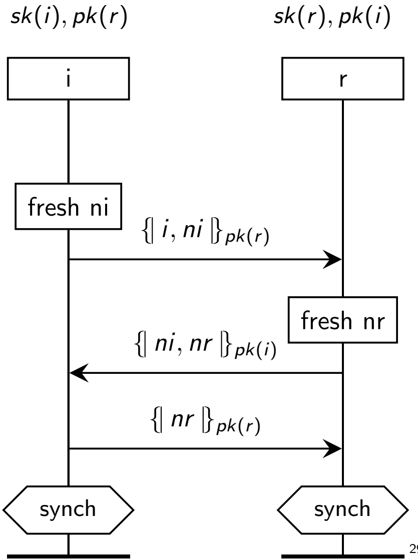
Notation: $Cons(M) = \{t \in RoleTerm \mid M \vdash t\}$

Exercise: Prove that $\{\{\!\mid m \mid\!\}_k, \{\!\mid k^{-1} \mid\!\}_{pk(b)}, sk(b)\} \vdash \{\!\mid m \mid\!\}_{sk(b)}$.

Protocol specification

Informally, in order to specify a protocol we have to describe each role, i.e.

- we define the initial knowledge of an agent playing that role as a set of *RoleTerms*
  - for example, the set $\{i, r, ni, sk(i), pk(i), sk(r)\}$ defines the initial knowledge of i
- we define the actions of an agent playing that role using a sequence of *RoleEvents*



Terms: *RoleEvent*

We use *role events* to specify protocol actions and claims.

- Given two disjoint sets:
  - *Label* : $1, 2, 3, \ldots$ (labels)
  - *Claim* : $\mathit{secret}, \mathit{alive}, \ldots$ (denotations for security properties)

  and $R \in \mathit{Role}$ we define

$$\begin{aligned} \mathit{RoleEvent}_R \quad ::= \quad & \mathit{send}_{\mathit{Label}}(R, \mathit{Role}, \mathit{RoleTerm}) \\ & \mid \mathit{recv}_{\mathit{Label}}(\mathit{Role}, R, \mathit{RoleTerm}) \\ & \mid \mathit{claim}_{\mathit{Label}}(R, \mathit{Claim}[, \mathit{RoleTerm}]) \end{aligned}$$

$$\mathit{RoleEvent} = \bigcup_{R \in \mathit{Role}} \mathit{RoleEvent}_R$$

Termeni: *RoleEvent*

$$
\begin{aligned}
\mathit{RoleEvent}_R \quad ::= \quad & \mathit{send}_{\mathit{Label}}(R, \mathit{Role}, \mathit{RoleTerm}) \\
& \mid \mathit{recv}_{\mathit{Label}}(\mathit{Role}, R, \mathit{RoleTerm}) \\
& \mid \mathit{claim}_{\mathit{Label}}(R, \mathit{Claim}[, \mathit{RoleTerm}])
\end{aligned}
$$

- $\mathit{send}_l(R, R', rt)$ means that R sends the message rt to R',
- $\mathit{recv}_l(R', R, rt)$ means that R receives the message rt, (apparently) sent by R',
- $\mathit{claim}_l(R, c, rt)$ is the security property that should be satisfied after the execution of the role R.
- The labels uniquely identify the events and establish the correspondence between *send* and *receive* events.

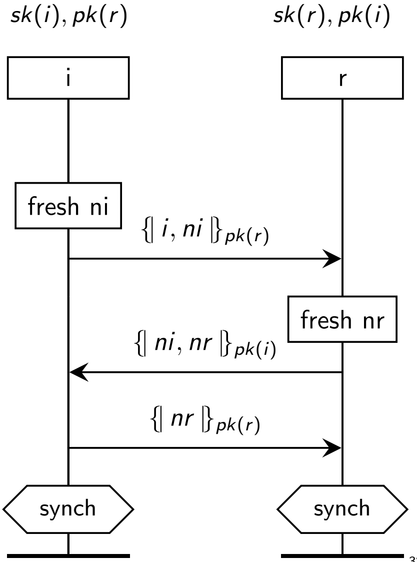
Running example

The Needham-Scroeder Public Key Protocol

$$A \xrightarrow{\{\!|\, A, na \,|\!\}_{pk(B)}} \qquad \xrightarrow{\{\!|\, x, y \,|\!\}_{pk(B)}} B$$

$$A \xleftarrow{\{\!|\, na, z \,|\!\}_{pk(A)}} \qquad \xleftarrow{\{\!|\, y, nb \,|\!\}_{pk(x)}} B$$

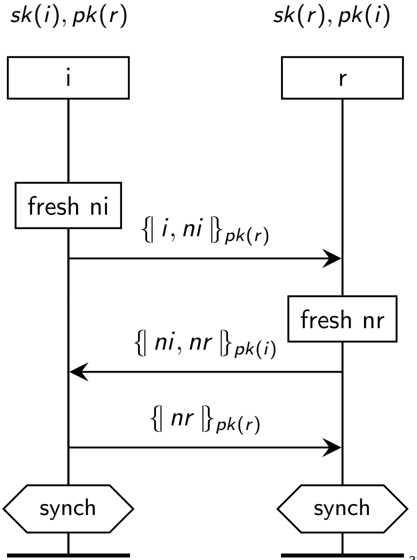
$$A \xrightarrow{\{\!|\, z \,|\!\}_{pk(B)}} \qquad \xrightarrow{\{\!|\, nb \,|\!\}_{pk(B)}} B$$



Protocol specification

Informally, in order to specify a protocol one should describe each role.

$$\begin{aligned}NS(i) = \quad & (\{i, r, ni, sk(i), pk(i), pk(r)\},\\ & [send_1(i, r, \{\!|\, ni, i \,|\!\}_{pk(r)}),\\ & recv_2(r, i, \{\!|\, ni, V \,|\!\}_{pk(i)}),\\ & send_3(i, r, \{\!|\, V \,|\!\}_{pk(r)}),\\ & claim_4(i, synch)])\end{aligned}$$



Role description

$$NS(i) = (\{i, r, ni, sk(i), pk(i), pk(r)\},$$
$$\begin{array}{l}[send_1(i, r, \{\!|\, ni, i \,|\!\}_{pk(r)}),\\ recv_2(r, i, \{\!|\, ni, V \,|\!\}_{pk(i)}),\\ send_3(i, r, \{\!|\, V \,|\!\}_{pk(r)}),\\ claim_4(i, synch)])\end{array}$$

- $\{i, r, ni, sk(i), pk(i), sk(r)\}$ is the *initial knowledge* of the role i,
- $s = [send_1(\cdots), \ldots, claim_4(\cdots)]$ is the sequence of events that are executed by i during a protocol session.

Protocol specification

Let P be a protocol and R a role in P.

The specification of the role R in P, denoted $P(R)$, is a pair from $\mathcal{P}(RoleTerm) \times RoleEvent_R^*$.

Example: The roles i and r of NSPK are specified as follows:

$$
\begin{aligned}
NS(i) = \; & (\{i, r, ni, sk(i), pk(i), pk(r)\}, \\
& [send_1(i, r, \{\!| \, ni, i \, |\!\}_{pk(r)}), \\
& recv_2(r, i, \{\!| \, ni, V \, |\!\}_{pk(i)}), \\
& send_3(i, r, \{\!| \, V \, |\!\}_{pk(r)}), \\
& claim_4(i, synch)])
\end{aligned}
\qquad
\begin{aligned}
NS(r) = \; & (\{i, r, nr, sk(r), pk(r), pk(i)\}, \\
& [recv_1(i, r, \{\!| \, W, i \, |\!\}_{pk(r)}), \\
& send_2(r, i, \{\!| \, W, nr \, |\!\}_{pk(i)}), \\
& recv_3(i, r, \{\!| \, nr \, |\!\}_{pk(r)}), \\
& claim_5(r, synch)])
\end{aligned}
$$

Protocol specification

The roles i and r of NSPK are specified as follows:

$$
\begin{aligned}
NS(i) = \quad & (\{i, r, ni, sk(i), pk(i), pk(r)\}, \\
& [send_1(i, r, \{\!|\, ni, i \,|\!\}_{pk(r)}), \\
& recv_2(r, i, \{\!|\, ni, V \,|\!\}_{pk(i)}), \\
& send_3(i, r, \{\!|\, V \,|\!\}_{pk(r)}), \\
& claim_4(i, synch)])
\end{aligned}
\qquad
\begin{aligned}
NS(r) = \quad & (\{i, r, nr, sk(r), pk(r), pk(i)\}, \\
& [recv_1(i, r, \{\!|\, W, i \,|\!\}_{pk(r)}), \\
& send_2(r, i, \{\!|\, W, nr \,|\!\}_{pk(i)}), \\
& recv_3(i, r, \{\!|\, nr \,|\!\}_{pk(r)}), \\
& claim_5(r, synch)])
\end{aligned}
$$

Protocol specification

$$NS(i) = (\{i, r, ni, sk(i), pk(i), pk(r)\},$$
$$\begin{array}{l}[send_1(i, r, \{\!|\ ni, i\ |\!\}_{pk(r)}), \\ recv_2(r, i, \{\!|\ ni, V\ |\!\}_{pk(i)}), \\ send_3(i, r, \{\!|\ V\ |\!\}_{pk(r)}), \\ claim_4(i, synch)])\end{array}$$

- $\{i, r, ni, sk(i), pk(i), sk(r)\}$ is the *initial knowledge* of the role i,
- $s = [send_1(\cdots), \ldots, claim_4(\cdots)]$ is the sequence of events that are executed by i during a protocol session,
- The (many-sorted) language associated with NSPK is:

  $Role = \{i, r\}$, $Fresh = \{ni, nr\}$, $Func = \emptyset$,
  $Var = \{V, W\}$, $Label = \{1, 2, 3, 4, 5\}$.

If P be a protocol and R a role in P then the specification of the role R in P, denoted $P(R)$, is a pair from

$$\mathcal{P}(RoleTerm) \times RoleEvent_R^*$$

.

Role description

$P(R) = (KN_0(R), s) \in \mathcal{P}(RoleTerm) \times RoleEvent_R^*$

- $KN_0(R) \subseteq \mathcal{P}(RoleTerm)$ is the initial knowledge of R
- $s \in RoleEvent_R^*$ is the sequence of events executed by R during a protocol session.

Remarks:

- the initial knowledge contains only terms without elements for *Var* (message variables);
- the labels are needed in order to disambiguate similar occurrences of an event term; consequently, each term from *RoleEvent* is unique in a protocol specification;
- the sequence of events might contain message variables from *Var*, which will be instantiated with concrete messages; within a role specification, the first occurrence of a message variable should be in an *accessible position* of a *receive* event.

Accessibility and well-formed sequences

- The accessibility relation $\sqsubseteq_{acc} \subseteq \mathit{RoleTerm}^2$ is the reflexive and transitive closure of the relation:

$$t_1 \sqsubseteq_{acc} (t_1, t_2),\ t_2 \sqsubseteq_{acc} (t_1, t_2),\ t_1 \sqsubseteq_{acc} \{\!|\ t_1\ |\!\}_{t_2}$$

- A sequence of role events $\rho \in \mathit{RoleEvent}^*$ is *well-formed* if the first occurrence of any message variable is in an accessible position of a *receive* event, i.e.

$$\forall\, V \in \mathit{vars}(\rho)\, \exists\, \rho', l, R, R', rt, \rho'' \\ (\rho = \rho' \cdot [\mathit{recv}_l(R, R', rt)] \cdot \rho'') \wedge V \notin \mathit{vars}(\rho') \wedge V \sqsubseteq_{acc} rt$$

  where $\mathit{vars}(\rho) \subseteq \mathit{Var}$ denotes the set of all message variables from ρ.

  The predicate $\mathit{wellformed}(\rho)$ denotes the fact that the sequence ρ is well-formed.

Protocol specification

- Role specification

$$RoleSpec = \{(kn, s) \mid kn \in \mathcal{P}(RoleTerm) \wedge \forall rt(rt \in kn \rightarrow vars(rt) = \emptyset) \\ \wedge\, s \in RoleEvent^* \wedge \; wellformed(s)\}$$

- Protocol specification

$$Protocol = Role \rightharpoondown RoleSpec$$

$P(R)$ is the specification of the role R for $P \in Protocol$ and $R \in Role$.

A protocol specification is a partial function from roles to role specifications.