Special Topics in Logic and Security I

Seminar & Lab

November 17, 2022

Contents

1 Theory

In this section we extract the mathematical concepts, definitions and notation used throughout in *Operation Semantics and Verification of Security Protocols*

1.1 Basic mathematical concepts

Powerset. Given a set A, we write $\mathcal{P}(A)$ or 2^A to denote the powerset of A.

Concatenation. Given two sequences t_0 and t_1, where $t_0 = [t_{00}, t_{01}, ...t_{0n}]$, and $t_1 = [t_{10}, t_{11}, ..., t_{1m}]$, the concatenation of these two sequences is denoted by $t \cdot t'$.

Sequence order. We write t_i to denote $(i+1)$th element of a sequence t, and we write $e <_t e'$ to denote $\exists i, j$ such that $i < j \wedge t_i = e \wedge t_j = e'$.

Projection function. Given a tuple $(x_1, x_2, ...x_n)$ we use the π_i function to project the ith component of the pair.

Function domains and codomain. Given a function f we write $dom(f)$ to denote the domain, and $ran(f)$ to denote the codomain. (range)

1.2 Labelled Transition Systems

A *labelled transition system* LTS is a four-tuple $(S, L, \rightarrow, s_0)$ where

- S is a set of states;
- L is a set of labels;
- $\rightarrow: S \times L \times S$ is a ternary transition relation;
- $s_0 \in S$ is the initial state.

We abbreviate $(p, \alpha, q) \in \rightarrow$ as $p \xrightarrow{\alpha} q$.

A *finite execution* of a LTS $P = (S, L, \rightarrow, s_0)$ is an alternating sequence σ of states and labels, starting with s_0 and ending with a state s_n, such that if $\sigma = [s_0, \alpha_1, s_1, \alpha_2 ..., \alpha_n, s_n]$, then $s_i \xrightarrow{\alpha_{i+1}} s_{i+1}$. If σ is a *finite execution* of LTS P, then $[\alpha_1, \alpha_2, ...\alpha_n]$ is called a *finite trace* of P.

A *transition rule* has a number of premises $Q_1, ...Q_n$ which must holds before a conclusion can be drawn:

$$\frac{Q_1 \quad Q_2 \quad ... \quad Q_n}{p \xrightarrow{\alpha} q}$$

1.3 Role Terms

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

Basic terms. We say a term is a *basic term* if it does not contains pairs or encryptions.

Vars function. $vars : RoleTerm \to \mathcal{P}(Var)$ determine the variables occurring in a term.

Roles function. $roles : RoleTerm \to \mathcal{P}(Role)$ determine the roles occurring in a term.

Unpair operator. $unpair : RoleTerm \to \mathcal{P}(RoleTerm)$, defined by

$$
unpair(t) = \begin{cases} unpair(t_1) \cup unpair(t_2) & \text{iff } t = (t_1, t_2) \\ \{t\} & otherwise \end{cases}
$$

Subterm relation. We define the syntatic subterm relation $\sqsubseteq$ as the reflexive, transitive closure of the smallest relation satisfying the following for all terms $t_1, ...t_n$, $1 \le i \le n$ and function names f: $t_1 \sqsubseteq (t_1, t_2)$; $t_2 \sqsubseteq (t_1, t_2)$; $t_1 \sqsubseteq \{t_1\}_{t_2}$; $t_2 \sqsubseteq \{t_1\}_{t_2}$; $t_i \sqsubseteq f(t_1, ...t_n)$; $t_1 \sqsubseteq k(t_1, t_2)$; $t_2 \sqsubseteq k(t_1, t_2)$; $t_1 \sqsubseteq pk(t_1)$; $t_1 \sqsubseteq sk(t_1)$.

Terms inference relation. Let M be a set of terms. The term inference relation $\vdash: \mathcal{P}(Term) \times Term$ is defined as the smallest relation satifying for all terms t, t_i, k and function names f:

$$
\begin{aligned}
t \in M &\Longrightarrow M \vdash t \\
M \vdash t_1 \wedge M \vdash t_2 &\Longrightarrow M \vdash (t_1, t_2) \\
M \vdash t \wedge M \vdash k &\Longrightarrow M \vdash \{t\}_k \\
M \vdash (t_1, t_2) &\Longrightarrow M \vdash t_1 \wedge M \vdash t_2 \\
M \vdash \{t\}_k \wedge M \vdash k^{-1} &\Longrightarrow M \vdash t \\
\bigwedge_{1 \le i \le n} M \vdash t_i &\Longrightarrow M \vdash f(t_1, ..., t_n)
\end{aligned}
$$

1.4 Role Events

$$
\begin{aligned}
RoleEvent_R ::= & send_{Label}(R, Role, RoleTerm) \\
& \mid recv_{Label}(Role, R, RoleTerm) \\
& \mid claim_{Label}(R, Claim[, RoleTerm])
\end{aligned}
$$

$$
RoleEvent = \bigcup_{R \in Role} RoleEvent_R
$$

Accessible subterm relation. The accessible subterm relation $\sqsubseteq_{acc}$ is defined as the reflexive, transitive closure of the smallest relation satisfying the following for terms t_1, t_2: $t_1 \sqsubseteq_{acc} (t_1, t_2)$; $t_2 \sqsubseteq_{acc} (t_1, t_2)$; $t_1 \sqsubseteq_{acc} \{t_1\}_{t_2}$.

Generalised vars function. $vars : RoleEvent^* \rightarrow \mathcal{P}(Vars)$ is a straightforward generalisation of the *vars* function for *Role Terms*,

Well-Formedness. The predicate $wellformed : RoleEvent^*$ is defined by

$$\begin{aligned} wellformed(\rho) \iff \forall V \in vars(\rho) : \exists \rho', l, R, R', rt, \rho'' \text{ such that} \\ \rho = \rho' \cdot [recv_l(R, R', rt)] \cdot \rho'' \wedge V \notin vars(\rho') \wedge V \sqsubseteq_{acc} rt \end{aligned}$$

Role Specification. Given a role R, role specification is defined as follows:

$$\begin{aligned} RoleSpec = \{(m, s) \mid m \in \mathcal{P}(RoleTerm) \wedge \forall rt \in m : vars(rt) = \emptyset \wedge \\ s \in (RoleEvent_R)^* \wedge wellformed(s)\} \end{aligned}$$

We require that the initial role knowledge does not contain variables.

Protocol Specification. We define $Protocol : Role \rightarrow RoleSpec$ the set of all possible protocol specifications. For every protocol $P \in Protocol$, and for each role $R \in Role$, $P(R)$ is the role specification of R. We can write $P(R) = (KN_0(R), s)$ where $KN_0(R)$ is a shorthand for the initial knowledge of the role R, and s is a sequence of events.

1.5 Event Order

Role event order. Let R be a role with specification $P(R) = (M, [\varepsilon_1, ..., \varepsilon_n])$. For R, the role event order $<_R$: $RoleEvent \times RoleEvent$ is defined as the strict total order defined by the sequence $[\varepsilon_1, ..., \varepsilon_n]$.

Communication Relation. The *communication relation* $\dashrightarrow : RoleEvent \times RoleEvent$ is defined as

$$\varepsilon_1 \dashrightarrow \varepsilon_2 \iff \exists l, R, R', rt_1, rt_2 \text{ such that } \varepsilon_1 = send_l(R, R', rt_1) \wedge \varepsilon_2 = recv_l(R, R', rt_2)$$

for all $\varepsilon_1, \varepsilon_2 \in RoleEvent$.

Protocol Order. Let P be a protocols with roles *Role*. The transitive closure of the union of the role event order and the communication relation is called the protocol order $\prec_P$:

$$\prec_P = \left(\dashrightarrow \cup \bigcup_{R \in Role} <_R \right)^+$$

1.6 Runs

Executing a role turns a role descripton into a run

Run identifiers. We denote the set of run identifiers as RID.

Agents. We denote the set of agents as $Agent$.

$$\begin{aligned} RunTerm ::= & Fresh^{\#RID} \mid Role^{\#RID} \mid Var^{\#RID} \\ & \mid Agent \\ & \mid Func([RunTerm[, RunTerm]*]) \\ & \mid (RunTerm, RunTerm) \\ & \mid \{RunTerm\}_{RunTerm} \\ & \mid AdversaryFresh \\ & \mid pk(RunTerm) \mid sk(RunTerm) \mid k(RunTerm, RunTerm) \end{aligned}$$

Instantiations. We define $Inst$, the *instantiations set*, as

$$Inst = RID \times (Role \to Agent) \times (Var \to RunTerm)$$

Function runidof. We define $runidof : Inst \to RID$, a projection function to denote the run identifier from an instantiation *inst*.

Term Instantiation. Let $inst \in Inst$, where $inst = (\theta, \rho, \sigma)$. Let $f \in Func$, and $rt, rt_1, ..., rt_n$ be role terms such that $roles(rt) \subseteq dom(\rho)$ and $vars(rt) \subseteq dom(\sigma)$. We define $< inst >: RoleTerm \to RunTerm$ by:

$$< inst > (rt) = \begin{cases} n^{\#\theta} & rt = n \in Fresh \\ \rho(R) & rt = R \in Role \wedge R \in dom(\rho) \\ R^{\#\theta} & rt = R \in Role \wedge R \notin dom(\rho) \\ \sigma(v) & rt = v \in Var \wedge v \in dom(\sigma) \\ v^{\#\theta} & rt = v \in Var \wedge v \notin dom(\sigma) \\ f(< inst > (rt_1), ..., < inst > (rt_n)) & rt = f(rt_1, ..., rt_n) \\ (< inst > (rt_1), < inst > (rt_2)) & rt = (rt_1, rt_2) \\ \{< inst > (rt_1)\}_{<inst>(rt_2)} & rt = \{rt_1\}_{rt_2} \\ sk(< inst > (rt_1)) & rt = sk(rt_1) \\ pk(< inst > (rt_2)) & rt = pk(rt_2) \\ k(< inst > (rt_1), < inst > (rt_2)) & rt = k(rt_1, rt_2) \end{cases}$$

Runs. The set of all possible runs is defined as $Run = Inst \times RoleEvent^*$.

1.7 Matching

Matching predicate. $Match : Inst \times RoleTerm \times RunTerm \times Inst$. The purpose of this predicate is to match an incoming message to a pattern specified by a role term, in the context of a particular instantiation.

Type function. The function *type* : $Var \to \mathcal{P}(RunTerm)$ defines the set of run terms that are valid values for a variable. The definition of the *type* function *depends on the agent model.*

Match. For all $inst = (\theta, \rho, \sigma)$ and $inst' = (\theta', \rho', \sigma') \in Inst$, $pt \in RoleTerm$ and $m \in RunTerm$, the predicate $Match(inst, pt, m, inst')$ holds iff

$$\theta = \theta' \wedge \rho = \rho' \wedge$$
$$< inst' > (pt) = m \wedge \forall v \in dom(\sigma') : \sigma'(v) \in type(v) \wedge \sigma \subseteq \sigma' \wedge dom(\sigma') = dom(\sigma) \cup vars(pt)$$

The definition of *Match* ensures that

- the instantiation of the pattern is equal to the message;
- the instatiation is well typed;
- the new variable assignment extends the old one;
- the instantiation is only extended for the variables that occur in the pattern.

Type matching. For all variables V,

$$\begin{aligned} &type(V) \in \{S_1, S_2, S_3, S_4, S_5\} \text{ where} \\ &\quad S_1 ::= Agent \\ &\quad S_2 ::= Func([RunTerm[, RunTerm]*]) \\ &\quad S_3 ::= pk(RunTerm) \mid sk(RunTerm) \\ &\quad S_4 ::= k(RunTerm, RunTerm) \\ &\quad S_5 ::= Fresh^{\#RID} \mid AdversaryFresh \end{aligned}$$

Constructor matching. For all variables V,

$$\begin{aligned} &type(V) \in \{T_1, T_2, RunTerm - (T_1 \cup T_2)\} \text{ where} \\ &\quad T_1 ::= \{RunTerm\}_{RunTerm} \\ &\quad T_2 ::= (RunTerm, RunTerm) \end{aligned}$$

1.8 Run Events

Run Event. We define the set of run events *RunEvents* as $Inst \times (RoleEvent \cup \{create(R) \mid R \in Role\})$. We write *RecvRunEv* for the set of run events corresponding to receive events, *SendRunEv* for those corresponding to send events, and *ClaimRunEv* for claim events.

Contents of event. $cont : (RecvRunEv \cup SendRunEv) \rightarrow RunTerm$ is a function that specify the contents of an event, i.e.

$$cont((inst, send_l(R, R', m))) =< inst > (R, R', m)$$
$$cont((inst, recv_l(R, R', m))) =< inst > (R, R', m)$$

Possible runs. $runsof : Protocol \times Role \rightarrow \mathcal{P}(Run)$ is a function for the runs that can be created by a protocol P for a role $R \in dom(P)$:

$$\begin{aligned} runsof(P, R) =& \{((\theta, \rho, \emptyset), s) \mid s = \pi_2(P(R)) \wedge \\ & \theta \in RID \wedge dom(\rho) = roles(s) \wedge ran(\rho) = Agent\} \end{aligned}$$

Active run identifiers. $runIDs(F) = \{\theta \mid ((\theta, \rho, \sigma), s) \in F\}$, where F is a set of runs.

1.9 Threat Model

We have that $Agent = Agent_H \cup Agent_C$ (reunion between *honest agents* and *compromised agents*).

Initial Adversary Knowledge. For a protocol P, we define the initial adversary knowledge $AKN_0(P)$ as

$$AKN_0(P) = AdversaryFresh \cup Agent \cup \bigcup_{R \in Role, \rho \in Role \to Agent, \rho(R) \in Agent_C} \{< \theta, \rho, \emptyset > (rt) \mid \theta \in RID \wedge rt \in KN_0(R) \wedge \forall rt' \sqsubseteq rt : rt' \notin Fresh\}$$

1.10 Operational semantics for security protocols

The operational semantics for security protocols P is defined using a LTS

$$(State, RunEvent, \to, s_0(P))$$

State. The set of possible states is $State = \mathcal{P}(RunTerm) \times \mathcal{P}(Run)$.

Initial State. The initial state of a protocol is defined as

$$s_0(P) =<< AKN_0(P), \emptyset >>$$

Transitions. We have AKN, the current intruder knowledge, and F, the set of active runs. The operational semantics rules are:

$$[create]\frac{R \in dom(P) \quad ((\theta, \rho, \emptyset), s) \in runsof(P, R) \quad \theta \notin runIDs(F)}{<< AKN, F >> \xrightarrow{((\theta,\rho,\emptyset),create(R))} << AKN, F \cup \{((\theta, \rho, \emptyset), s)\} >>}$$

$$[send]\frac{e = send_l(R_1, R_2, m) \quad (inst, [e] \cdot s) \in F}{<< AKN, F >> \xrightarrow{(inst,e)} << AKN \cup \{< inst > (m)\}, (F - \{(inst, [e] \cdot s)\}) \cup \{(inst, s)\} >>}$$

$$[recv]\frac{e = recv_l(R_1, R_2, pt) \quad (inst, [e] \cdot s) \in F \quad AKN \vdash m \quad Match(inst, pt, m, inst')}{<< AKN, F >> \xrightarrow{(inst',e)} << AKN, (F - \{(inst, [e] \cdot s)\}) \cup \{(inst', s)\} >>}$$

$$[claim]\frac{e = claim_l(R, c) \vee e = claim_l(R, c, t) \quad (inst, [e] \cdot s) \in F}{<< AKN, F >> \xrightarrow{(inst,e)} << AKN, (F - \{(inst, [e] \cdot s)\}) \cup \{(inst, s)\} >>}$$

Traces. We define $traces(P)$ as the set of finite traces of the LTS associated with a protocol P.

1.11 Security properties

Honestity. We define the predicate *honest* for instantiations as $honest((\theta, \rho, \sigma)) \iff ran(\rho) \subseteq Agent_H$.

Actor function. $actor : Inst \times RoleEvent \to Agent$ is defined as $actor((\theta, \rho, \sigma), \varepsilon) = \rho(role(\epsilon))$.

Secrecy Claim. Let P be a protocol with role R. The secrecy claim event $\gamma = claim_l(R, secret, rt)$ is corect iff

$$\forall t \in traces(P) : \forall((\theta, \rho, \sigma), \gamma) \in t : honest((\theta, \rho, \sigma)) \Rightarrow AKN(t) \nvdash < (\theta, \rho, \sigma) > (rt)$$

2 Solved Exercises

Exercise 1 *Give role terms s and t such that $\{s\} \vdash t$ but not $t \sqsubseteq s$.*

Solution. We can choose $s = (rt_1, rt_2)$ and $t = (rt_2, rt_1)$.

Exercise 2 *Compute $unpair\big(\{p\}_{k(R,R')}, h(a,b)\big)$.*

Solution. Using the unpair operator definition, we have

$$unpair(t) = \begin{cases} unpair(t_1) \cup unpair(t_2) & \text{iff } t = (t_1, t_2) \\ \{t\} & \text{otherwise} \end{cases}$$

In this exercise,

$$unpair\big(\{p\}_{k(R,R')}, h(a,b)\big) = unpair(\{p\}_{k(R,R')}) \cup unpair(h(a,b))$$

Exercise 3 *Prove that the following role description is not well-formed.*

$$\begin{aligned} P(i) = (&\{i, r, k\}, \\ &[send_1(i, r, \{i, r, V\}_k \\ &\ read_2(r, i, \{V, r\}_k)]) \end{aligned}$$

Solution. We use the well-formedness predicate. The $wellformed : RoleEvent^*$ holds iff

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

We have that $\rho' = [send_1(i, r, \{i, r, V\}_k)]$, and $V \in vars(\rho')$, but also $V \sqsubseteq_{acc} rt$, where $rt = \{V, r\}_k$, so the predicate does not hold.

Exercise 4 *Compute the term instantiation for $< 1, \{i \to A, r \to B\}, \emptyset > (\{n_i, i\}_{pk(r)})$.*

Solution.

$$\begin{aligned} &< 1, \{i \to A, r \to B\}, \emptyset > (\{n_i, i\}_{pk(r)}) \\ &= \{< 1, \{i \to A, r \to B\}, \emptyset > (n_i, i)\}_{<1,\{i \to A, r \to B\}, \emptyset > (pk(r))} \\ &= \{< 1, \{i \to A, r \to B\}, \emptyset > (n_i), < 1, \{i \to A, r \to B\}, \emptyset > (i)\}_{pk(<1,\{i \to A, r \to B\}, \emptyset > (r))} \\ &= \{n_i^{\#1}, A\}_{pk(B)} \end{aligned}$$

Exercise 5 *Prove that the term $h(k)$ can be inferred from the set $\big\{\{m^{-1}\}_k, \{k^{-1}\}_{pk(b)}, \{h(k)\}_m, sk(b)\big\}$.*

Solution. We denote the set as Γ. Then,

$$\dfrac{\dfrac{\dfrac{\dfrac{\Gamma}{sk(b)} \quad \{k^{-1}\}_{pk(b)}}{k} \quad \{m^{-1}\}_k}{m^{-1}} \quad \{h(k)\}_m}{h(k)}$$

Exercise 6 *Assume* $\rho = \{i \to A, r \to B\}$ *and assume* $type(X) = S_5$. *Show that the predicate* $Match((1, \rho, \emptyset), X, nr^{\#2}, (1, \rho, \{X \to nr^{\#2}\}))$ *holds.*

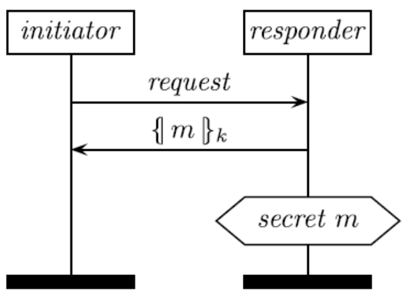
Solution. By definition, $Match((\theta, \rho, \sigma), pt, m, (\theta', \rho', \sigma'))$ holds if and only if $\theta = \theta'$, $\rho = \rho'$ and $< (\theta', \rho', \sigma') > (pt) = m$ and forall $v \in dom(\sigma')$ we have that $\sigma'(v) \in type(v)$ and $\sigma \subseteq \sigma'$ and $dom(\sigma') = dom(\sigma) \cup vars(pt)$.
We have that $1 = 1$ and $\rho = \rho$, so the first two conditions are satisfied. We can immediate show that $< (1, \rho, \{X \to nr^{\#2}\}) > (X) = nr^{\#2}$ using the term-instantiation definition.
For all $v \in dom(\sigma')$ means for all $v \in \{X\}$; so for $v = X$, we have to show that $\sigma'(X) \in type(X)$. We already have that $\sigma \subseteq \sigma'$ (because $\emptyset \subseteq \{X \to nr^{\#2}\}$), and $dom(\sigma') = dom(\sigma) \cup vars(pt)$, that is $\{X\} = \emptyset \cup \{X\} = \{X\}$.
The last thing to prove is that $\sigma'(X) \in type(X)$, and $type(X) = S_5$ using the assumption, so the predicate $Match((1, \rho, \emptyset), X, nr^{\#2}, (1, \rho, \{X \to nr^{\#2}\}))$ holds.

Exercise 7 *Assume* $\rho = \{i \to A, r \to B\}$ *and assume* $type(X) = S_5$. *Show that the predicate* $Match((1, \rho, \emptyset), X, (nr^{\#2}, ni^{\#1}), inst')$ *does not hold, for any instantiation* $inst'$.

Solution. This exercise is immediate because the $\sigma'(v) \notin type(v)$, forall $v \in dom(\sigma')$ (types does not match). For any $v \in dom(\sigma')$, $type(\sigma'(v)) \in RunTerm - (T_1 \cup T_2)$ (using constructor matching), and $type(v) \in T_2$.

Exercise 8 *Consider the SSC protocol. Give role specifications of the SSC protocol, determine* $<_{initiator}$, $<_{responder}$ *and* $\dashrightarrow$. *Determine* $\prec_{SSC}$.

protocol Simple Secret Communication (SSC)



Solution. We'll consider $initiator = i$ and $responder = r$.

a. Roles description.

$$SSC(i) = (\emptyset, [send_1(i, r, request), recv_2(r, i, \{m\}_k)])$$
$$SSC(r) = (\emptyset, [recv_1(i, r, request), send_2(r, i, \{m\}_k), claim_3(r, secret, m)])$$

b. Roles events order.

$$send_1(i, r, request) <_{initiator} recv_2(r, i, \{m\}_k)$$
$$recv_1(i, r, request) <_{responder} send_2(r, i, \{m\}_k) <_{responder} claim_3(r, secret, m)$$

The role event order is defined as the strict total order for a sequence, so if we have $[\varepsilon_1, \varepsilon_2, \varepsilon_3]$ for a role R, then $\varepsilon_1 <_R \varepsilon_2$ and $\varepsilon_2 <_R \varepsilon_3$ and $\varepsilon_1 <_R \varepsilon_3$.

c. Communication relation.

By definition, *communication relation* $\dashrightarrow : RoleEvent \times RoleEvent$ is defined as

$$\varepsilon_1 \dashrightarrow \varepsilon_2 \iff \exists l, R, R', rt_1, rt_2 \text{ such that } \varepsilon_1 = send_l(R, R', rt_1) \land \varepsilon_2 = recv_l(R, R', rt_2)$$

for all $\varepsilon_1, \varepsilon_2 \in RoleEvent$, so in this case

- $send_1(i, r, request) \dashrightarrow recv_1(i, r, request)$
- $send_2(r, i, \{m\}_k) \dashrightarrow recv_2(r, i, \{m\}_k)$

Now, we can compute $\prec_{SSC}$,

$$\prec_{SSC} = \left(\dashrightarrow \cup \bigcup_{R \in Role} <_R \right)^+$$

We have:

$$\begin{aligned}
&send_1(i, r, request) \prec_{SSC} recv_2(r, i, \{m\}_k) \\
&recv_1(i, r, request) \prec_{SSC} send_2(r, i, \{m\}_k) \\
&send_2(r, i, \{m\}_k) \prec_{SSC} claim_3(r, secret, m) \\
&recv_1(i, r, request) \prec_{SSC} claim_3(r, secret, m) \\
&send_1(i, r, request) \prec_{SSC} recv_1(i, r, request) \\
&send_2(r, i, \{m\}_k) \prec_{SSC} recv_2(r, i, \{m\}_k) \\
&\ldots
\end{aligned}$$

3 Other exercises

Exercise 9 *Assume* $\rho = \{i \to A, r \to B\}$ *and assume* $type(X) = S_5$*. Prove that the following predicate holds.*

$$Match((1, \rho, \emptyset), \{ni, r\}_{pk(i)}, \{ni^{\#1}, B\}_{pk(A)}, (1, \rho, \emptyset))$$

Exercise 10 *Prove that the following predicate does not hold, for any instantiation* $inst'$*.*

$$Match((1, \rho, \emptyset), nr, nr^{\#2}, inst')$$

Exercise 11 *Given the SSC protocol from Exercise 8, determine* $AKN_0(SSC)$*.*

4 Scyther Tool

4.1 Installation

Download the archive from https://people.cispa.io/cas.cremers/scyther/install-generic.html.
Now you can run:

```
sudo apt-get install graphviz python python-wxgtk3.0
```

If there are errors, run the following commands:

```
sudo apt install python2.7 python-pip
sudo apt-get install python-wxgtk3.0
sudo apt-get install graphviz
```

To run Scyther, you can use the following command:

```
python scyther-gui.py
```

4.2 Scyther Syntax

General protocol structure:

```
protocol <name> (<ag_1>, <ag_2>[, <ag_3, ...])
{
    role <ag_1> { <spec ag_1> }
    role <ag_2> { <spec ag_2> }
    [role <ag_3> { <spec ag_3> } ...]
}
```

Variables types:

```
Agent
Function // in general, one-way functions
Nonce
```

Variable specification:

```
var <name>: <type>
```

Nonce specification:

```
fresh <name>: Nonce;
```

Tuples:

```
(x, y)
(x, y , z) // interpreted as ((x, y), z)
```

Symmetric keys:

```
{<message>}<symmkey>
```

Public / secret keys:

```
{<message>}pk(<agent>)
{<message>}sk(<agent>)
```

Hash functions:

```
hashfunction <name>;
```