Special topics in Logic and Security I

Master Year II, Sem. I, 2022-2023

Ioana Leuștean
FMI, UB

References

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

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

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

- $\{i, r, ni, sk(i), pk(i), sk(r)\}$ 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

$P(R) = (KN_0(R), s)$

- $KN_0(R) \subseteq \mathcal{P}(RoleTerm)$ 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 initial knowledge is subsequently used in order to infer the adversary knowledge;
- 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 $\sqsubseteq_{acc} \subseteq RoleTerm^2$ 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 $\rho \in RoleEvent^*$ is *well-formed* if the first occurrence of any message variable is in an accessible position of a *receive* event, i.e.

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

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

  The predicate $wellformed(\rho)$ 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.

Protocol execution

- The specification of a protocol describes each role.
- When a protocol is executed, an agent can play any role, one or more times, sequentially or in parallel (concurrently).
- A single execution of a role is called a *run*. Different runs are described using *Run Identifiers* (RID). The concrete execution of a protocol is described using *Run Terms*.
- Turning a role description into a run with the help of run identifiers is called *instantiation*. Note that the fresh values should be uniquely identified in each run.

The description of a run: *RunTerm*

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

- ${}^{-1} : RunTerm \rightarrow RunTerm$
- *AdversaryFresh* are run terms generated by an adversary

Deduction system on $Term = RoleTerm \cup RunTerm$

We extend the deduction to

$$Term = RoleTerm \cup RunTerm$$

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

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

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

$$\begin{array}{lrcl}
\text{if} & t \in M & \text{then} & M \vdash t \\
\text{if} & M \vdash t_1 \text{ and } M \vdash t_2 & \text{then} & M \vdash (t_1, t_2) \\
\text{if} & M \vdash (t_1, t_2) & \text{then} & M \vdash t1 \text{ and } M \vdash t_2 \\
\text{if} & M \vdash t \text{ and } M \vdash k & \text{then} & M \vdash \{\!|\, t \,|\!\}_k \\
\text{if} & M \vdash \{\!|\, t \,|\!\}_k \text{ and } M \vdash k^{-1} & \text{then} & M \vdash t \\
\text{if} & M \vdash t_1 \text{ and } \ldots \text{ and } M \vdash t_n & \text{then} & M \vdash f(t_1, \ldots, t_n)
\end{array}$$

The description of a run: *RunTerms*

- In order to specify a protocol we used generic terms from *RoleTerm*, which will be instantiated when we describe a concrete run.

  For example:
  the generic term i that designates the initiator role will be instantiated with A (Alice) which designates a concrete agent;

  the generic fresh value ni will be instantiated with $ni^{\#1}$, $ni^{\#2}$, ... which are the concrete values generated in the first run, the second run.

- An *instantiation* is a triplet

$$(\theta, \rho, \sigma) \in \mathit{RID} \times (\mathit{Role} \rightharpoonup \mathit{Agent}) \times (\mathit{Var} \rightharpoonup \mathit{RunTerm})$$

Term instantiation

Let *Inst* be the set of all instantiations:

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

To any $inst = (\theta, \rho, \sigma) \in Inst$ we associate a function

$$inst : RoleTerm \to RunTerm$$

defined as follows:

- $inst(n) = n^{\#\theta}$ pentru $n \in Fresh$
- $inst(R) = \rho(R)$ for $R \in Role \cap dom(\rho)$
  $inst(R) = R^{\#\theta}$ for $R \in Role \setminus dom(\rho)$
- $inst(V) = \sigma(V)$ for $V \in Var \cap dom(\sigma)$
  $inst(V) = V^{\#\theta}$ for $V \in Var \setminus dom(\sigma)$

Term instantiation

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

$inst = (\theta, \rho, \sigma) \in Inst$

- $inst(f(t_1, \ldots, t_n)) = f(inst(t_1), \ldots, inst(t_n))$
- $inst(t_1, t_2) = (inst(t_1), inst(t_2))$
- $inst(\{\!|\ t_1\ |\!\}_{t_2}) = \{\!|\ inst(t_1)\ |\!\}_{inst(t_2)}$
- $inst(sk(t)) = sk(inst(t)),\ inst(pk(t)) = pk(inst(t))$
- $inst(k(t_1, t_2)) = k(inst(t_1), inst(t_2))$

Term instantiation

Example:

$$inst = (\theta, \rho, \sigma) \in RID \times (Role \rightharpoonup Agent) \times (Var \rightharpoonup RunTerm)$$

$$inst = (2, \{i \mapsto B, r \mapsto A\}, \{W \mapsto ni^{\#1}\})$$

$$t = \{\!|\, W, nr, r \,|\!\}_{pk(i)} \in RoleTerm$$

$$inst(\{\!|\, W, nr, r \,|\!\}_{pk(i)}) = \{\!|\, ni^{\#1}, nr^{\#2}, A \,|\!\}_{pk(B)} \in RunTerm$$

Matching

We define a predicate *Match* that matches an incoming message (from *RunTerm*) with a given pattern (from *RoleTerm*) and extends the instantiation:

$$Match \subseteq Inst \times RoleTerm \times RunTerm \times Inst$$

$Match(inst, pt, m, inst')$ holds if

- $inst = (\theta, \rho, \sigma)$,
  $inst' = (\theta, \rho, \sigma')$
- $inst'(pt) = m$, $pt \in RoleTerm$, $m \in RunTerm$
- $dom(\sigma') = dom(\sigma) \cup vars(pt)$
- $\sigma \subseteq \sigma'$
- $\sigma'(v) \in type(v)$ for any $v \in dom(\sigma')$,

where $vars(pt)$ is the set of variables from *Var* which appear in *pt*, and $type(v)$ is a function that depends on the agent model.

Matching

Example:

We consider $type(V) \in \{S_1, S_2, S_3, S_4, S_5\}$ such that

$$
\begin{array}{lcl}
S_1 & ::= & Agent \\
S_2 & ::= & Func(RunTerm^*) \\
S_3 & ::= & Fresh|\ AdversaryFresh \\
S_4 & ::= & sk(RunTerm) \mid pk(RunTerm) \\
S_5 & ::= & k(RunTerm, RunTerm)
\end{array}
$$

If $type(X) = S_3$ then

- $Match((1, \rho, \emptyset), X, nr^{\#2}, (1, \rho, \{X \mapsto nr^{\#2}\})$
- $\neg\ Match((1, \rho, \emptyset), nr, nr^{\#2}, inst')$ wrong instantiation
- $\neg\ Match((1, \rho, \emptyset), X, (nr^{\#1}, nr^{\#2}), inst')$ wrong type

Operational semantics

Possible executions

The set of all possible executions is

$$Run = Inst \times RoleEvent^{*}$$

- The runs that can be created by a protocol P are defined by

  $runsof : Protocol \times Roles \to \mathcal{P}(Run)$

  $$\begin{aligned} runsof(P, R) = \{(inst, s) \mid\ & \text{there exists } kn \text{ such that } P(R) = (kn, s) \\ & inst = (\theta, \rho, \sigma) \text{ with } dom(\rho) = roles(s)\} \end{aligned}$$

  where $R \in dom(P)$.

- For $F \subseteq Run$ we set

  $runIds(F) = \{\theta \mid ((\theta, \rho, \sigma), s) \in F \text{ for some } \rho, \sigma, s\}$

States

$Run = Inst \times RoleEvent^*$

$$State = \mathcal{P}(RunTerm) \times \mathcal{P}(Run)$$

$st = \langle\langle AKN, F \rangle\rangle \in State$

- AKN is the adversary knowledge,
- $F \subseteq Run$ are the runs that has to be executed.

Exemple:

$st_1 = \langle\langle \{A, B, pk(A), pk(B)\}, \{(\,(2, \{i \mapsto A, r \mapsto B\}, \emptyset), [send_1(i, r, \{\!|\, ni \,|\!\}_{pk(r)})]\,)\}\rangle\rangle$

$st_2 = \langle\langle \{A, B, pk(A), pk(B), \{\!|\, ni^{\#2} \,|\!\}_{pk(A)}\}, \emptyset \rangle\rangle$

Labelled transition system (LTS)

We recall that a *labelled transition system* is a tuple $(St, L, \rightarrow, st_0)$ where:

- St is the set of states
- L is the set of labels
- $\rightarrow \subseteq S \times L \times S$ is the transition relation
- $st_0 \in St$ is the initial state

An *execution*: $[st_0, \alpha_1, st_1, \alpha_2, \ldots, \alpha_n, st_n]$ such that $st_i \stackrel{\alpha_{i+1}}{\rightarrow} st_{i+1}$

A *trace*: $[\alpha_1, \alpha_2, \ldots, \alpha_n]$

The operational semantics of a security protocol P is defined using a labelled transition system

$$(State, RunEvent, \rightarrow, st_0(P))$$

RunEvent

In order to specify a protocol we use:

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

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

In order to describe the *concrete execution* of a protocol we define:

$$\mathit{RunEvent} = \mathit{Inst} \times (\mathit{RoleEvent} \cup \{\mathit{create}(R) \mid R \in \mathit{Role}\})$$

- $\mathit{create}(R)$ is used to mark a new run of a role.

Operational semantics

We are now able to define the operational semantics of a security protocol P using the labelled transition system

$$(State, RunEvent, \rightarrow, st_0(P))$$

$State = \mathcal{P}(RunTerm) \times \mathcal{P}(Run)$

$st_0(P) = \langle\langle AKN_0(P), \emptyset \rangle\rangle$ where $AKN_0(P)$ is the initial adversary knowledge.

In order to model the adversary knowledge, the set of agents is partitioned in *honest agents* and *corrupted agents*: $Agent = Agent_H \cup Agent_C$. The (Dolev-Yao) adversary controls the network, (s)he creates fresh terms and (s)he knows the initial knowledge of the compromised agents.

For example, in the Needham-Schroeder protocol

$$AKN_0(NS) = AdversaryFresh \ \cup \ Agent \cup \{pk(A) \mid A \in Agent\} \cup \{sk(A) \mid A \in Agent_C\}$$

Operational semantics

$$(State, RunEvent, \rightarrow, st_0(P))$$

- $State = \mathcal{P}(RunTerm) \times \mathcal{P}(Run)$ where $Run = Inst \times RoleEvent^*$
- $st_0(P) = \langle\langle AKN_0(P), \emptyset \rangle\rangle$ where $AKN_0(P)$ is the initial adversary knowledge
- $RunEvent = Inst \times (RoleEvent \cup \{create(R) \mid R \in Role\}$
- The *transition system* has four rules, one for each of the events:

  create, *send*, *recv*, *claim*

Operational semantics: transitions

$$(State, RunEvent, \rightarrow, st_0(P))$$

$$[create_P]\ \frac{R \in dom(P) \quad ((\theta,\rho,\emptyset),s) \in runsof(P,R) \quad \theta \notin runsIDs(F)}{\langle\!\langle AKN, F\rangle\!\rangle \xrightarrow{((\theta,\rho,\emptyset),create(R))} \langle\!\langle AKN, F \cup \{((\theta,\rho,\emptyset),s)\}\rangle\!\rangle}$$

Recall that

$runIds(F) = \{\theta \mid ((\theta,\rho,\sigma),s) \in F$ for some $\rho,\sigma,s\}$ and

$F \subseteq Run = Inst \times RoleEvent^*$.

Operational semantics : transitions

$$(\mathit{State}, \mathit{RunEvent}, \rightarrow, \mathit{st}_0(P))$$

$$[\mathit{send}]\ \frac{e = \mathit{send}_l(R_1, R_2, m) \quad (\mathit{inst}, [e] \cdot s) \in F}{\langle\!\langle \mathit{AKN}, F \rangle\!\rangle \xrightarrow{(\mathit{inst}, e)} \langle\!\langle \mathit{AKN} \cup \{\mathit{inst}(m)\}, F \setminus \{(\mathit{inst}, [e] \cdot s)\} \cup \{(\mathit{inst}, s)\} \rangle\!\rangle}$$

$$[\mathit{claim}]\ \frac{e = \mathit{claim}_l(R, c, t) \quad (\mathit{inst}, [e] \cdot s) \in F}{\langle\!\langle \mathit{AKN}, F \rangle\!\rangle \xrightarrow{(\mathit{inst}, e)} \langle\!\langle \mathit{AKN}, F \setminus \{(\mathit{inst}, [e] \cdot s)\} \cup \{(\mathit{inst}, s)\} \rangle\!\rangle}$$

Operational semantics : transitions

$$(State, RunEvent, \rightarrow, st_0(P))$$

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

Recall that $Match(inst, pt, m, inst')$ holds if the incoming message m is matched with the pattern pt and the instantiation $inst'$ is $inst$ extended with the new assignments.

Operational semantics: traces

$$(State, RunEvent, \rightarrow, st_0(P))$$

- Execution:
  $[st_0, \alpha_1, st_1, \alpha_2, \dots, \alpha_n, st_n]$ where $\alpha_i \in RunEvent$ și $st_i = \langle\langle AKN_i, F_i \rangle\rangle$
- Knowing the initial state we define the execution using traces $[\alpha_1, \alpha_2, \dots, \alpha_n]$.

  Given a protocol P, we define $traces(P)$ as the set of the finite traces of the labelled transition system $(State, RunEvent, \rightarrow, st_0(P))$ associated to P.

Example: trace for the Needham-Schroeder protocol

$((1, \rho, \emptyset), create(i))$

$((1, \rho, \emptyset), send_1(i, r, \{\!|\, ni, i \,|\!\}_{pk(r)}))$

$((2, \rho, \emptyset), create(r))$

$((2, \rho, \{W \mapsto ni^{\#1}\}), recv_1(i, r, \{\!|\, W, i \,|\!\}_{pk(r)}))$

$((2, \rho, \{W \mapsto ni^{\#1}\}), send_2(r, i, \{\!|\, W, nr \,|\!\}_{pk(i)}))$

$((1, \rho, \{V \mapsto nr^{\#2}\}), recv_2(r, i, \{\!|\, ni, V \,|\!\}_{pk(i)}))$

$((1, \rho, \{V \mapsto nr^{\#2}\}), send_3(i, r, \{\!|\, V \,|\!\}_{pk(r)}))$

$((1, \rho, \{V \mapsto nr^{\#2}\}), claim_4(i, synch))$

$((2, \rho, \emptyset), recv_3(i, r, \{\!|\, nr \,|\!\}_{pk(r)}))$

$((2, \rho, \emptyset), claim_5(r, synch))$

Thank you!

References

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