www.avispa-project.org

IST-2001-39252

Automated Validation of Internet Security Protocols and Applications

Contents

CONTENTS

2 User Section

This section describes the easiest way to use the AVISPA tool: to specify protocols in HLPSL, then to run the `avispa` script for analysing it.

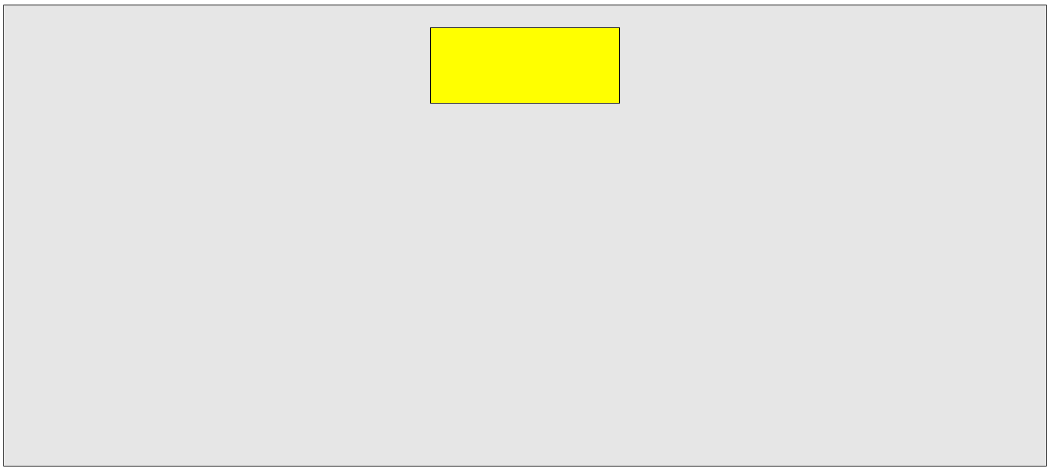

Figure 1: Architecture of the AVISPA tool v.1.0

2.1 Specifying Protocols: HLPSL

Protocols to be studied by the AVISPA tool have to be specified in HLPSL (standing for *High Level Protocols Specification Language*), and written in a *file with extension* `hlpsl`.

c. Definition of roles. The roles in a specification are of two kinds: basic roles played by agents, and composition roles describing the scenario to consider during analysis (for example, describing what is a session of the protocol, or what instances of sessions should be used).

```
% Roles may be either basic or compositional:
```

e. Declarations in roles. The first element in a role is its header. It contains the role name (a constant) and its parameters (a list of declarations of variables with their type).

```
Role_header ::=
  const_ident "(" Formal_arguments? ")"

Formal_arguments ::=
  (Variable_declaration ",")* Variable_declaration
```

A role may contain numerous declarations:

- local declarations: declarations of variables with their type;
- constants declarations: declaring constants with their type is not local to the role; any constant in one role can b.rat inheire;

2 USER SECTION

A predicate of the last form has to correspond to the reception of a message in a channel (for example: `Rcv({M'}_K)`).

Contrarily to spontaneous actions, immediate reactions happen when the player of the role is in

```
C214l,14ate14la14te14d_v14ar14iab14le14s_l14is14t :14:=
  C214l,14ate14la14ted14_v14ari14ab14le14s
| "(14" C14on14ca14ten14at14ed_14va14ria14bl14es ")14"

C214l,14ate14la14te14d_v14ar14iab14le14s :14:=
```

The available macros correspond to:

- the secrecy of some information,
- the strong authentication of agents on some information,
- the weak authentication of agents on some information.

Each goal is identified by a constant, referring to predefined predicates (`secret`, `witness`, `request` and `wrequest`) added in transitions by the user. For more details on those predicates, see the description of actions, page 15.

```
Goal_declaration ::=
  "goal" Goal_formula+ "end" "goal"

Goal_formula ::=
  "secrecy_of" Constants_list
| "authentication_on" Constants_list
| "weak_authentication_on" Constants_list
| "[]" LTL_unary_formula

LTL_unary_formula ::=
  LTL_unary_predicate
| "<->" LTL_unary_formula
| "(-)" LTL_unary_formula
| "[-]" LTL_unary_formula
| "~" LTL_unary_formula
| "(" LTL_formula ")"

LTL_formula ::=
  LTL_predicate
| "<->" LTL_unary_formula
| "(-)" LTL_unary_formula
| "~" LTL_unary_formula
```

```
  | "(" Subtype_of ")"
  | Compound_type
```

no attack is reported for this value.)
The label `id` (of type `protocol_id`) is used to identify the goal. In the HLPSL goal section the statement `secrecy_of id` should be given to refer to it.

```
% Receipt of response from key server
learn.  State  = 1 /\ RCV({B.Kb'}_inv(Ks))
    =|> State':= 0 /\ KeyRing':=cons(B.Kb', KeyRing)
```

```
& AddToSet({a}_(set_117))
& AddToSet({b}_(set_117))
```

```
Options:
  --types       Print identifiers and their types
  --init        Print initial state
  --rules       Print protocol rules
  --goals       Print goals
  --all         Print everything (default)
```

3.1 Generating an IF Specification

```
AtorTf6-1TferTfm ::=
  coTfnsTft_iTfdeTfnt
| naTft_iTfdeTfnt
| vaTfr_iTfdeTfnt
```

f. Section for equations. This section represents the equaTftionaTfl theoTfry that haTfs to be considered for some specific functioTfn oTfpe such as paTfi,xoTfn and exTfp.

```
EqTfuaTftioTfnsTfSeTfction ::=
 "sTfecTftioTfn eqTfuatioTfns:" EquTfation*

EqTfuaTftioTfn ::=
  TeTfrm "=" TerTfm

TeTfrm ::=
  AtorTf6-1TferTfm
| CoTfmpoTfseTfd1Tferm

CoTfmposeTfd1TferTfm ::=
  IF_OTfperr3mF_Oe100f(14inocg46Tstfperr3m46F)11"" TerTfm
```

i. Section for properties.

```
AttackStateDeclaration ::=
  "attack_state" AttackStateID "(" VariableList ")" ":=" CState

AttackStateID ::=
  "secrecy_of_"const_ident
| "authentication_on_"const_ident
| "weak_authentication_on_"const_ident
```

k. Section for intruder behaviour. This section contains the description of the intruder behaviour, represented by transition rules.

```
IntruderSection ::=
  "section intruder:" RuleDeclaration*
```

In the current version of the AVISPA tool, this section is unique because only the Dolev-Yao

```
pair     : message * message -> message
% asymmetric encryption: crypt(Key,Message)
crypt    : message * message -> message
% inverse of a public key (=private key): inv(Key)
inv      : message -> message
% symmetric encryption: scrypt(Key,Message)
```

```
inv(inv(PreludeM)) = PreludeM

% commutation of exponents:
```

3.1.4 Example The IF specification given in the following has been automatically generated from the HLPSL specification of the Needham-Schröder Public Key Protocol with Key Server (Section 2.1.3).

```
initial_state init1 :=
 iknows(start).
 iknows(ki).
 iknows(inv(ki)).
 iknows(a).
 iknows(b).
 iknows(ks).
 iknows(ka).
 iknows(kb).
 iknows(i).
 state_server(s,kn,set_91,dummy_agent,dummy_agent,dummy_pk,2).
 state_alice(a,b,ka,ks,set_93,0,dummy_nonce,dummy_nonce,dummy_pk,
   set_105,set_106,4).
 state_bob(b,a,kb,ks,set_94,0,dummy_nonce,dummy_nonce,dummy_pk,
   set_116,set_117,5).
 state_alice(a,i,ka,ks,set_93,0,dummy_nonce,dummy_nonce,dummy_pk,
   set_123,set_124,6).
 contains(pair(a,ka),set_91).
 contains(pair(b,kb),set_91).
 contains(pair(i,ki),set_91).
 contains(pair(a,ka),set_93).
 contains(pair(b,kb),set_93).
 contains(pair(b,kb),set_94)


section rules:

step step_0 (S,Kn,KeyMap,Dummy_A,Dummy_B,Dummy_Kb,SID,A,B,Kb) :=
 state_server(S,Kn,KeyMap,Dummy_A,Dummy_B,Dummy_Kb,SID).
 iknows(pair(A,B)).
 contains(pair(B,Kb),KeyMap)
 =>
 state_server(S,Kn,KeyMap,A,B,Kb,SID).
 iknows(crypt(inv(Ks),pair(B,Kb))).
 contains(pair(B,Kb),KeyMap)

step step_1 (A,B,Ka,Kn,KeyRing,Dummy_Na,Nb,Dummy_Kb,Set_23,Set_27,SID,Na,Kb) :=
 state_alice(A,B,Ka,Ks,KeyRing,0,Dummy_Na,Nb,Dummy_Kb,Set_23,Set_27,SID).
 iknows(start).
 contains(pair(B,Kb),KeyRing)
```

```
  not(contains(i,ASGoal))

 attack_state secrecy_of_snb (MGoal,ASGoal) :=
  iknows(MGoal).
  secret(MGoal,snb,ASGoal) &
  not(contains(i,ASGoal))
```

3.2 Analysing a IF Specification

- `lr`

```
 Unforgeable terms : inv(ks) inv(kca)
```

Computed list of term that the intruder cannot forge.

```
Interpreted protocol specification
----------------------------------

Role server played by (s,7):
```

First instance of the role "server".

```
| start => s, ks, n26(Ns)
```

First step: receives *start* and send a nonce $n26(Ns)$.

```
| Choice Point
```

Second step: chose one branch or the other.

```
|  | Csus(27), {i,ki}_(inv(kca)) => n27(SeID)
```

Third step: assumes $\{i, ki\}_{inv(kca)}$ was received.

```
|  | .....
```

Other steps.

```
| Or
| g| Csus(31), {s,ks}_(inv(kca)) =n27(Se31(,)3114(i)1(26)1((Ns)1())]T/F151172.688f430.66
```

$_{inv(kca)}$ was received.

- Using the `-p` option, one can "manually browse" the search tree, e.g.:

  `-p` is the root node,

  `-p 0` is the first (left-most) successor of the root node, schema [3] and the other one applying the explanatory frame schema); it can be set to either `linear`, `gp-bca` or `gp-efa` (default value).

- `mutex`: level of the mutex relations to be used during the SAT-reduction; if set to `0`, then the abstraction/refinement strategy provided by SATMC (see [2

These examples about `ta4spv2` runs concern the two protocols: Needham Schroeder Public Key protocol (NSPK.if) and its corrected version (NSPK-fix.if).

1. `./ta4spv2 --2AgentsOnly --level 0 NSPK-fix.if`:

```
SUMMARY
   SAFE

DETAILS
   TYPED_MODEL
   OVER_APPROXIMATION
   UNBOUNDED_NUMBER_OF_SESSIONS

PROTOCOL
   NSPK-fix.if
...
COMMENTS
   TA4SP uses abstractions '2AgentsOnly'
   For the given initial state, an over-
   approximation is used with an unbounded
   number of sessions.
   Terms supposed not to be known by the
   intruder are still secret.
...
```

2. `./ta4spv2 --2AgentsOnly --level 0 NSPK.if`:

3.

```
StepNumber ::=
```

A XEmacs mode

The former are immediately accessible in the Options submenu, while the latter are accessible *via* the $\boxed{\text{More Options}}$

The drawback is that XEmacs will hang if the backend does not terminate. Note also that Ofmc is not sensitive to this value, and will always be launched asynchronously.

> *When a backend or the compiler is launched asynchronously, one need to use the navigation buttons* << *and* >> *to go to the result buffer. Once in the right buffer, one should use* Update *to see the result. This should be done only once the tool has terminated.*

- Fetch Result: Set this value to nil if you do not want the mode to automatically display the result of a process, i.e. compilation or verification. There is no automatic fetching when

B HLPSL Semantics

B.1 Preliminaries

The semantics o--1(eHL(o)P(s)-LPSL448(iics)-45bema)9(s)-1dheThee(s)-mpoo

$$TLA(B) \triangleq Init(B) \quad \Box\, Next(B) \tag{1}$$

where $Next(B)$ is defined as:

$$\Box_j\, ev_j \qquad act_j \;\Box$$

RCV_k (with $1 \leq i \leq S$ and $1 \leq k \leq R$) refer to sending and receiving DY channels, respectively. The DY intrudeicmΩ7eads(icmΩude)-v(D)2eryheendingcmΩ4(c)27(hann24)]T/F2011.955T244.809880Td

$$Interleaving_{DY} \triangleq \square_{i=1}^{R} \left(\mathrm{RCV\_flag'}_i = \mathrm{RCV\_flag}_i \quad \square_{j=1, j=i}^{R} \mathrm{RCV\_flag'}_j = \mathrm{RCV\_flag}_j \right)$$

Figure 7: Dolev-Yao intruder behaviour: necessary condition for interleaving semantics.

However, the formula in Figure 7

One may hence see composed types as syntactic sugar, but they allow us to write the rules for

[12]