1 HLPSL Basics

AVISPA provides a language called the

```
A -> S : {Kab}_Kas
S -> B : {Kab}_Kbs
```

This simple protocol illustrates A-B notation as well as some of the naming conventions we local variables of `Alice`: in this case, one called `State` which is a `nat` (a natural number) and another called `Kab`, which will represent the new session key. `Kab` is also declared to be `(fresh)`, which intuitively means that `A` will generate its value randomly.[5] The local `State` variable is initialized to `0` in the `init` with `Kab`, then we update the value of `State` to be equal to `2`, we update the value of `Kab` to be

The `Session` role usually declares all channels used by the roles as variables of type `channel (dy)`. These variables are not instantiated with concrete constants. The `dy` stands for the Doley-Yao intruder model. [5] This means that the intruder has full control over the network, such that all messages sent by agents will go to the intruder. He may suppress, analyze, and/or modify messages (as far as he knows the required keys), and send them to whoever he pleases. As a

The second transition is trickier: `RCV({Nb'}_K)`. Firstly, `Alice` receives a message. Provided that this message is of the form `{*}_K`, for some value `*`, `Alice` sets `Nb` to be the received value encrypted under `K`. In the same transition, the newly received `Nb` value is encrypted with a new key represented as `Hash(Na.Nb')`. This key is computed from both `Na` and `Nb`.

The full solution for this example is provided below. Note that it contains a number of aspects r-47217 37.5595576(b)-28e595576expltaied(.)-42(Fy)82formpln, the trmts

```
    2. State  = 2 /\ RCV({Nb'}_K) =|>
       State' = 4 /\ K1' = Hash(Na.Nb')
                  /\ SND({Nb'}_K1')
                  /\ witness(A,B,nb,Nb')

  end role
```

```
  role Bob(
      A,B     : agent,
      K       : symmetric_key,
      Hash    : function,
      SND,RCV : channel(dy))
  played_by B def=

    local
      State   : nat,
```

The first attempt at this example is given below. Please note that there are a lot of errors which you may already recognize.

```
role Alice (A,B:agent,...)
played_by A def=

  local State: t0sh12,%.9111
```

```
    step1. State = 0 /\ RCV(A.B.{K.Na’.Ns’}_Kb.{Na.Nb’}_K) =|>
           SND(A.B.{Nb’.Na}_K) /\ State’ = 1

  end role
```

```
  role Server(....)
  played_by S def=

    local A, B : agent,
          Na   : text,
          Ns   : text(fresh),
          State: nat

    init State = 0

    transition

    step1. State = 0 /\ RCV(A.B.{Na’}_Ka) =|>
           SND(A.B.{K.Na.Ns’}_Ka.{K.Na.Ns}_Kb}) /\ State’ = 1

  end role
```

Whilst attempting to correct this version, much confusion centered around when to prime variables. In the above example, there are several examples of incorrect priming. Priming is actually quite simple if yorbsimp-1if

On the right hand side of a transition, use a primed variable name when assigning a new value to a variable. For example:

```
State’ = 3
```

```
  Server(A, S, B, Ka, Kb, SAS, RAS)
```

```
      X     : {symmetric_key.text.text}_symmetric_key

const na    : protocol_id

init   State = 0
```

```
  /\  Bob   (A, S, B, Kb, SAB, RAB)

end role
```

```
role Environment()
def=

  const a, b, s    : agent,
        ka, kb, ki : symmetric_key

  intruder_knowledge = {921-514({921ml]Tk28.892)1(tr)1(ude)1(r_)1(kn)1(owl)1(ed)1(ge)
```

```
  OFMC
COMMENTS
STATISTICS
```

by sending the special

before. The value `N1b` is sent for use in the future. Both `K1ab` and `N1b` are generated by `B`.

Our first attempt at the protocol is below, keep in mind that the initial definition was misunderstood.[13]

First, buggy version of Example 3:

```
role Alice (A, B    : agent,
played_by B def=

  local State  : nat,
        Na, Nb : text(fresh)

  init State = 0

  transition

  step1. State  = 0 /\ RCV({Kab}_Na’) =|>
         State’ = 2 /\ SND({Na’+1.Nb’}_Kab)

  step2. State  = 2 /\ RCV({Nb+1}Kab) =|>
         State’ = 4 /\ SND({K’ab.N’b}_Kab)

end role
```

On a glance we can spot something wrong: `SND({Kab}_Na’)` should read `SND({Na’}_Kab)` in `step1` of `A`. [14]

We can also see the variables `K’ab` and `N’b`

```
            Kab      : symmetric_key,
            Succ     : function,
            SND, RCV : channel(dy))
played_by A
def=

  local State  : nat,
  local State  : nat,
        Nb     : text (fresh),
        Na     : text,
        K1ab   : symmetric_key (fresh),
        N1b    : text (fresh)

  const k1ab, na, nb : protocol_id

  init State = 1

  transition

  1. State  = 1 /\ RCV(A.{Na'}_Kab) =|>
     State' = 3 /\ SND({Succ(Na').Nb'}_Kab)
                /\ witness(B,A,na,Na')

  3. State  = 3 /\ RCV({Succ(Nb)}_Kab) =|>
     State' = 5 /\ SND({K1ab'.N1b'}_Kab)
                /\ witness(B,A,k1ab,K1ab')
                /\ request(B,A,nb,Nb)
                /\ secret(K1ab', A) /\ secret(K1ab', B)
                /\ secret(N1b' , A) /\ secret(N1b' , B)

end role
```

```
role Session(A, B  : agent,
             Kab   : symmetric_key,
             Succ  : function)
def=
```

This reads: ”I am `B`, please remember that I have presented `Na` to `A` so that we can check this later.”

The semantics of the `witness` and `(w)request` predicates when used to describe authen-

[illegible]

```
DETAILS
  ATTACK_FOUND
PROTOCOL
  ex31.if
GOAL
  a authenticates b on k1ab
%wrequest(a,b,k1ab,K1ab(7))
BACKEND
  OFMC
COMMENTS
STATISTICS
  parseTime: 0.00s
  searchTime: 1.19s
  visitedNodes: 809 nodes
  depth: 8 plies
ATTACK TRACE
i -> (a,3): start
ATTA0503a,3}:st{]Tu04-c46T1(t)]T0(t)2A.446Td.1(t)2.3043-14.446Td[((a)1(,3)1())-iesAT61(
(a,3) -> i: a.{Na(1)}_k(V1)1(s1)1N0->visL (a,3): start
```

3.2 Witness and Request

When using `witness` and `(w)request`, the third argument is an identifier of type `protocol_id` declared in the top-level role. This is used to associate the `witness` and `(w)request` predicates with each other.

3.3 Secrecy

If you want to express that a certain value (represented by a term `T`) produced or selected by a role `A` is [illegible] of(n)27(s(,)412(sab)27(y82[(,)]TJ/F3911.955Tf806.6050Td[BT)]TJ/F1911.955Tf6.152

Figure 3: A valid representation of role instantiation

```
end role

role Bob(B: agent,...)
played_by B def=
  ...
end role
```

So what does this mean? Well, there are three agents (or principals) taking part in this scenario: `a`, `b` and `i`. In two of the sessions, `a` plays role Alice: Alice 1 and Alice 2 (see Figure 3).

```
       Session(a,b,kab)
    /\ Session(a,b,kab)
       ...
```

Essentially, this code sample is stating that there are two identical sessions between the same client and the same server (a

Do not worry too much about this. The most important thing is that each agent must have

A SYMBOLS AND KEYWORDS

References

[1] AVISPA. Deliverable 2.1: The High-Level Protocol Specification Language. Available at `http://www.avispa-project.org`, 2003.

[2]