A Short SPAN+AVISPA Tutorial

Thomas Genet

A Short SPAN+AVISPA Tutorial

Thomas Genet
IRISA/Université de Rennes 1
genet@irisa.fr

February 27, 2017

Abstract

The objective of this short tutorial is to show how to use SPAN to understand and debug HLPSL specifications used in the AVISPA cryptographic protocol verification tool. The reader is supposed to be familiar with the HLPSL language.

1 Installing and starting SPAN+AVISPA

Download SPAN+AVISPA from `http://people.irisa.fr/Thomas.Genet/`. This tutorial was done using the virtual box disk installation. The virtual disk is the simplest solution to have a complete and fully working SPAN+AVISPA installation. First, install VirtualBox. Uncompress the `span_on_ubuntu10.7z` file using the appropriate tool (depending on the operating system you use). This phase may take some time. Start the VirtualBox application. Then click on File>Import a virtual disk. Select the file that you just uncompress (should be named `ubuntu 10.10 light.ova`) and click on Import.

Launch the "Ubuntu 10.10 light" virtual machine. You are automatically logged in but, if you need it, the login is `span` and the password is `span`. Then, click on the "SPAN" shortcut located on the desktop. The main window of SPAN opens. The role of each part of the SPAN tool is described in Figure 1 (except the "View CAS+" button that will be discussed later).

2 A first protocol specification and first simulation

Go back to the Ubuntu desktop. In the "Tutorial" folder, double click on the file `exercise0.hlpsl`. The file is opened in a text editor. In SPAN, open the same file: click on the "File" menu, then click on "Open HLPSL file". In the file browser open the "Desktop" folder, then the "tutorial" folder and open the `exercise0.hlpsl` file. This file describes a simple protocol of this form: $initiator \hookrightarrow responder : M$ where *initiator* and *responder* are agents and M is a constant message known by both agents. The HLPSL specification describes the two agents *initiator*/*responder*, the combination of the two agents into a session and finally the environment in which the session is run and including the initial intruder knowledge. In the SPAN window, click on the "Protocol simulation" button. A new window opens where you can build a Message Sequence Chart (MSC) of the protocol, see Figure 2. In the rightmost window you can see the MSC under construction and on the leftmost part the "Incoming events" window gives you the list of all the incoming messages. Here there is only one possible message sending between the initiator and the responder. Double click on this event. The event disappears from the incoming events list and appears in the MSC, see Figure 3. You can use the "Previous step" and "Next step" buttons to go backward and forward in the simulation.

Now we are going to search for attacks on this simple protocol. In the HLPSL file, the secrecy property states that the message M is a secret shared between the agents playing the initiator and the responder role. This property is clearly false since any intruder observing the protocol can read the message M in clear. Let's go back to the SPAN window and select one verification tool between `OFMC` and `ATSE`[1] and click on the "Execute" button. In the edition window, you can see the trace of the found attack. To obtain a MSC of the attack, simply click on the "Attack simulation" button. A new simulation window opens where you can see that the message sent by the initiator can be read (and blocked) by the intruder. Note also that there is still one "Incoming event" in the list. This is due to the fact that the responder is still waiting for a message. However, the attack ends here, there is no need to trigger this additional messages. In the leftmost, red frame you can also read what is the intruder knowledge: he knows all the agent names: `a`, `b` and `i` (his name) and also the message `m1`,

[1] `SATMC` and `TA4SP` are not fully functional in this version.

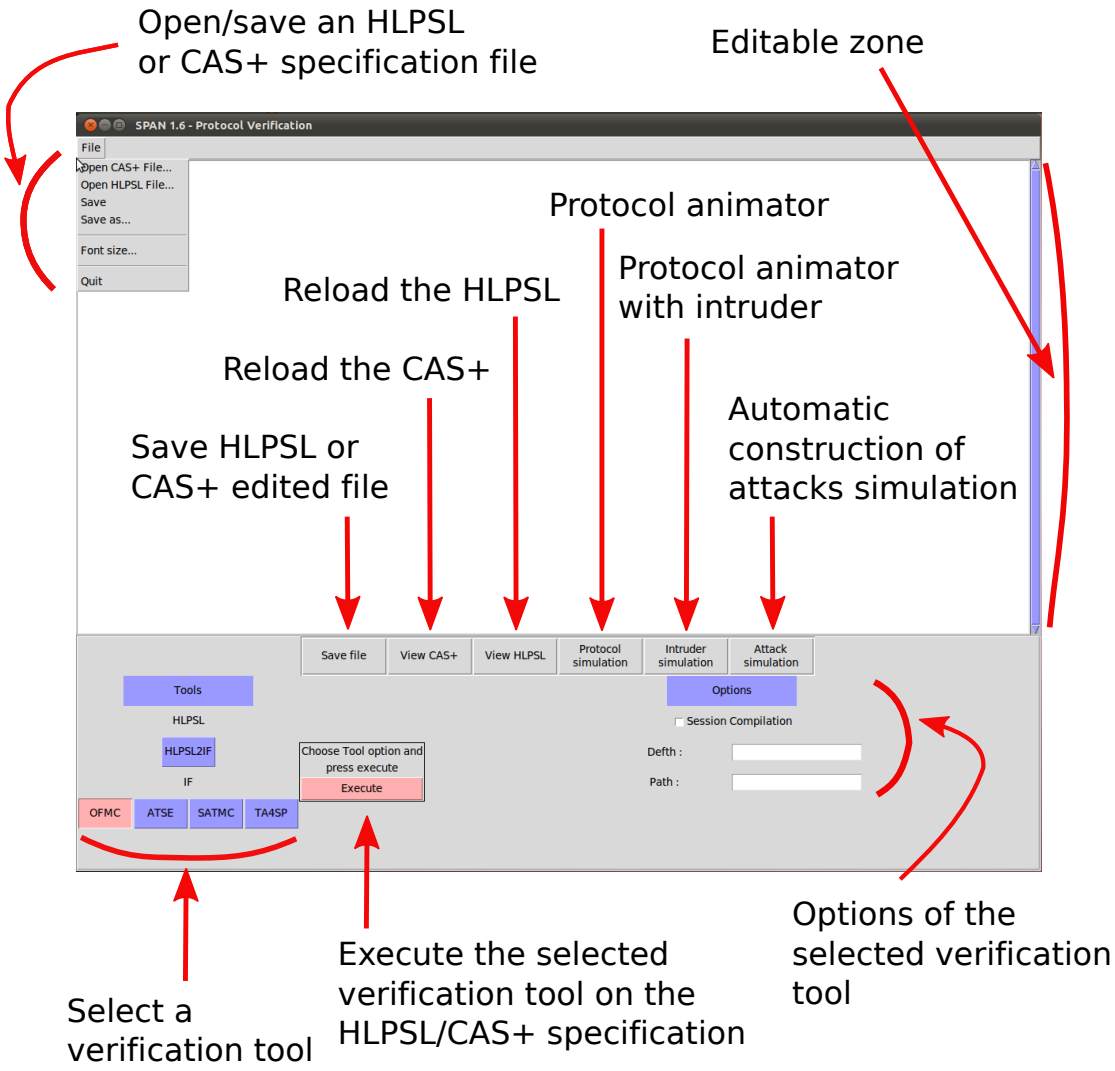

Figure 1: The full SPAN main graphical interface.

see Figure 4. Clicking on the "Previous step" button, you can go one step back in the simulation, and remark that before he can intercept the message, the intruder does not know `m1`.

In this MSC, the initiator role is played by `a`. There is a number associated to `a`, here 3. This number is only needed by verification tools to disambiguate the name of the player when it plays several roles. In general, you do not need to care about this number. The responder is played by `b`, and the message is the constant `m1` as it is

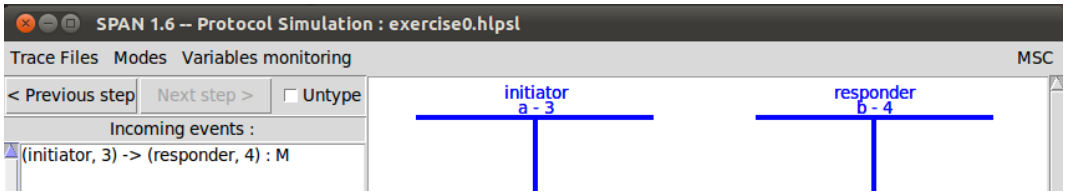

Figure 2: The initial MSC in the protocol simulation window.

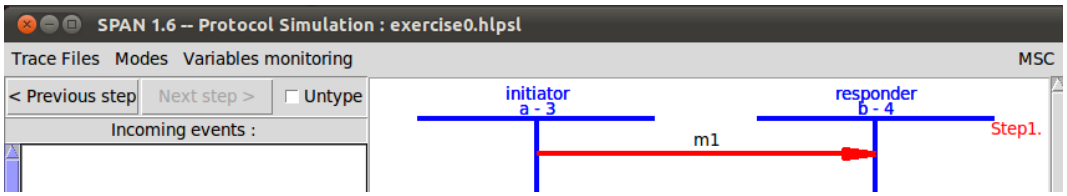

Figure 3: The MSC obtained after sending the first (and unique) message.

defined in the HLPSL specification. Now we are going to modify the HLPSL specification and see the effect in the simulation. We could modify it directly in the edition window of SPAN, but it is not recommended since the implemented text editing functionalities are very basic. Go back to the specification in the **text editor** (and not in SPAN). In the `environment()` role, note that there are two sessions but the second one is commented using `%`. Uncomment the second session. In the second session the role initiator is played by `b` and the responder is played by `a` and the message is `m2`. Save the HLPSL file. Go back to the SPAN editing window and click on the button "View HLPSL" to refresh the edition window with the modification you just made using the text editor. Now click on the "Protocol simulation" button. A new simulation window opens where you have a MSC with 4 vertical lines, corresponding to the two different sessions and two different incoming events. You can build two different MSC either by clicking on the first event and then on the second, or the opposite (see Figure 5).

3 Debugging HLPSL

Now we are going to see how SPAN can help you to debug your HLPSL specifications. In the text editor, open the `exercise1.hlpsl` file. This specification describe the following protocol:

1. $A \hookrightarrow B : \{N_A\}_{K_{AB}}$
2. $A \hookrightarrow B : N_A$

where N_A is a nonce generated by A and K_{AB} is a symmetric key known by A and B before starting the protocol. In this protocol, N_A is clearly not a secret: once B sends it in clear to A, the intruder can read it. Open `exercise1.hlpsl` in SPAN. Select a verification tool and click on "Execute". Surprisingly, the protocol is claimed to be safe! The reason is that the HLPSL specification is not fully executable... and a protocol that does not run cannot leak any secret! The fact that the specification is not executable is revealed by simulation: click on "Protocol simulation". In the window that opens, you can see that there are no "Incoming events" meaning that no message can be exchanged in the protocol. Thus, there is a mistake in the HLPSL specification. Generally in HLPSL, when a message cannot be exchanged it is either because the sent message was wrong or because the pattern of the receiver was erroneous. Let's have a look to the `exercise1.hlpsl` file in the text editor. In `role_B`, the receiving pattern is wrong: `RCV({Na}_Kab)` means that B receives a message ciphered with a key K_{AB} that he knows (which is correct) and whose content is a value `Na` that he also knows. However, `Na` is a nonce that has been generated by A and which is, thus, unknown to B. What is missing here is a quote on `Na` meaning that B receives a *new* value for `Na`. Change the two occurrences of `Na` into `Na'`. The transition should, in the end, look like this:

```
1. State = 1 /\ RCV({Na'}_Kab) =|> State':=3 /\ SND(Na')
```

Save the file in the editor and go back in the SPAN window. Click on "View HLPSL" button and launch "Protocol simulation". Now, the first message can be triggered by clicking on the first "Incoming event". Note that the last message, sent by B does not appear. This is not an error. This is due to the fact that the simulation can simply display messages that can be sent **and** received. In the HLPSL specification, no transition was defined to receive this last message. However, the intruder can receive this message and the protocol is no longer safe. We can observe this by selecting a verification tool and clicking on the "Attack simulation" button.

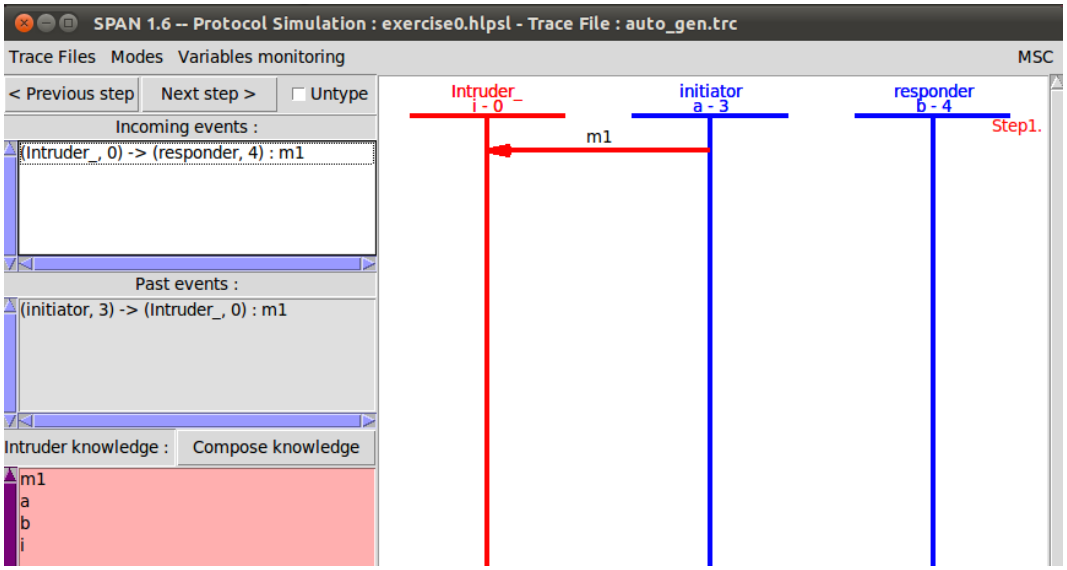

Figure 4: An attack displayed in "Attack simulation" window, with intruder knowledge.

In the MSC of the attack (see Figure 6), note that the nonce is the constant `nonce-1` (*i.e.* the first generated nonce). Note also that it is not the shortest attack, but it is the shortest attack found by the verification tools. The first message is sent by A to the intruder that simply forwards it to B. To simplify the attack, you can go back in the simulation using "Previous step" and start by the message sent by A to B and finish by the message sent by B to the intruder.

4 Displaying the variable values for more debugging

Now open in the text editor and in SPAN the file `NSPK_to_correct.hlpsl`. This is the famous Needham-Schroeder Public Key protocol.

1. $A \hookrightarrow B : \{N_A, A\}_{K_B}$
2. $B \hookrightarrow A : \{N_A, N_B\}_{K_A}$
3. $A \hookrightarrow B : \{N_B, \}_{K_B}$

This protocol is known to be flawed. However, if you select a verification tool and click on "Execute" the specification is said to be safe. If you click on protocol simulation, you can see, again, that no "Incoming event" is available. Note that, in the simulation, there are 2 lines for `alice` because in the specification there is one session between `alice` and `bob` and one between `alice` and the `intruder`. Since we are in simulation mode, the intruder is not figured out. The protocol specification is not executable and, thus, safe. All the patterns for sending and receiving messages are correct. However, no messages can be exchanged. Now, click on "Intruder simulation". This time, two possible "Incoming events" are available but it is not possible to run the protocol to then end. Trigger one of those messages. Then in the menu "Variable monitoring" select one of the `alice` (say `(alice,3)`). A new window opens and permit to select the variables (of the role) that you want to monitor. Select `Ka`, `Kb`, `Na`, `Nb` and click "OK". Now, a small pink rectangle has appeared on the blue line of `(alice,3)`. Note that, there should be at least one sent message in the simulation for the pink rectangle to show up. Click on this rectangle to display/hide the values of the variables. The values displayed are the values of the variables just before sending/receiving the message (if any). Do the same variable monitoring for `(bob,4)` and click on

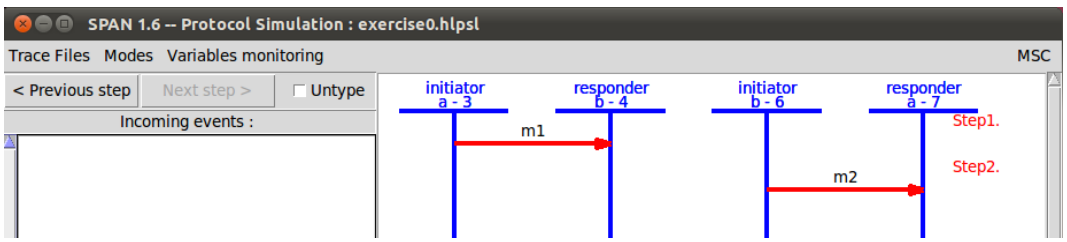

Figure 5: One of the possible MSC obtained for the same protocol with two sessions.

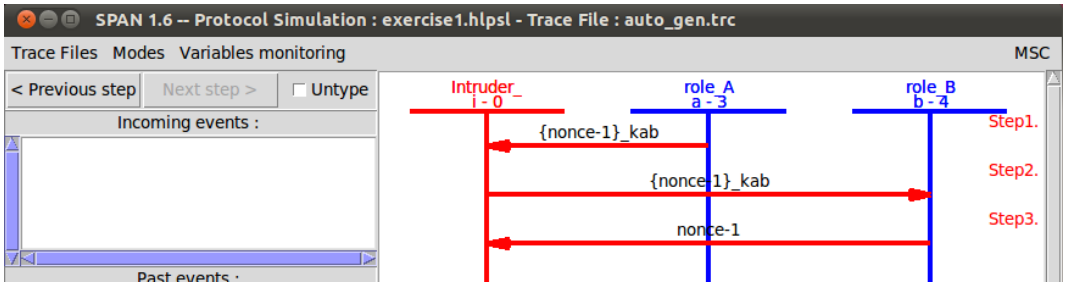

Figure 6: The attack found by verification tools.

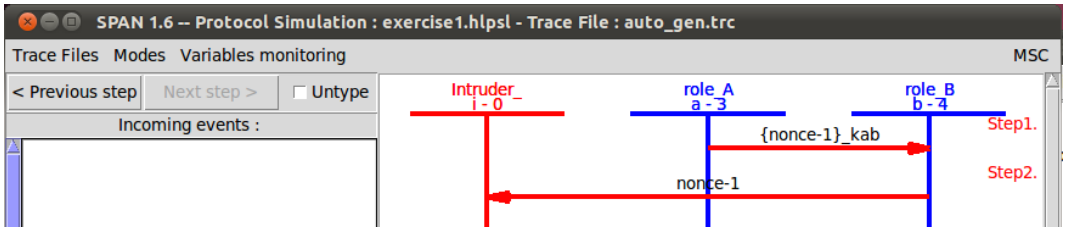

Figure 7: The same attack where unnecessary messages have been removed by hand.

the pink rectangle on the blue line of `(bob,4)`. You should obtain something similar to Figure 8.

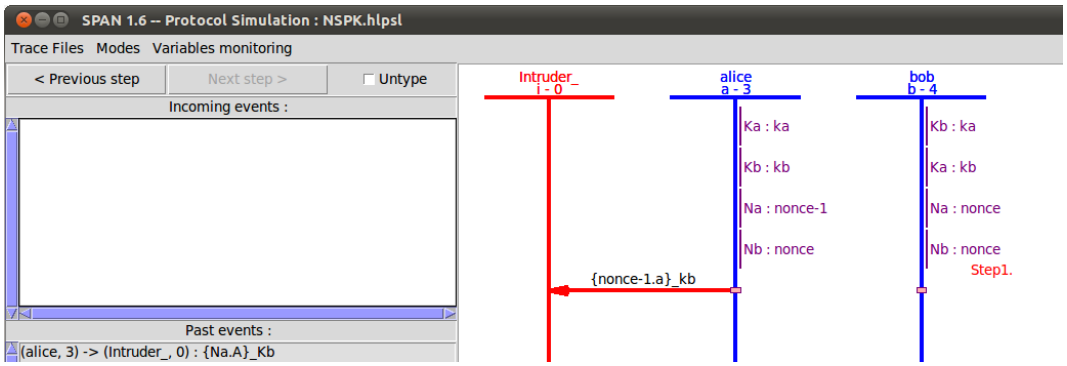

Figure 8: Variable monitoring and the clickable pink rectangle to display/hide variables.

By a careful inspection of the variable values, you can notice that the value of the keys `Ka` and `Kb` have been exchanged in the role `bob`: the variable `Ka` has value `kb` and vice versa. Go back to the specification in the text editor. The header of the role bob looks like this:

```
role bob(A, B: agent, Kb, Ka: public_key, SND, RCV: channel (dy))
[...]
```

and in the session definition the role is called with `bob(A,B,Ka,Kb,SB,RB)`, thus the key associated to variable `Ka` in the session is used under the name `Kb` in the role. To correct this, simply exchange the order of `Ka` and `Kb` in the header of role `bob`. It should finally be of the form:

```
role bob(A, B: agent, Ka, Kb: public_key, SND, RCV: channel (dy))
[...]
```

Save it and click on “View HLPSL” in the SPAN main window. Click on “Protocol simulation”. Now the protocol can be run to the end: the three messages can be sent and received. You should observe a MSC of the Figure 9. Note that, if you have difficulty in relating those messages to the protocol specification, it is possible to switch between different visualization modes. In the “Modes” menu, select for instance the item “view sender pattern in principal position”. This relabels the arrows using the pattern used in the HLPSL specification. Now, in the SPAN window, select a verification tool and click on execute. The protocol specification is now unsafe and the classical attack on NSPK can be nicely displayed by clicking on “Attack simulation”.

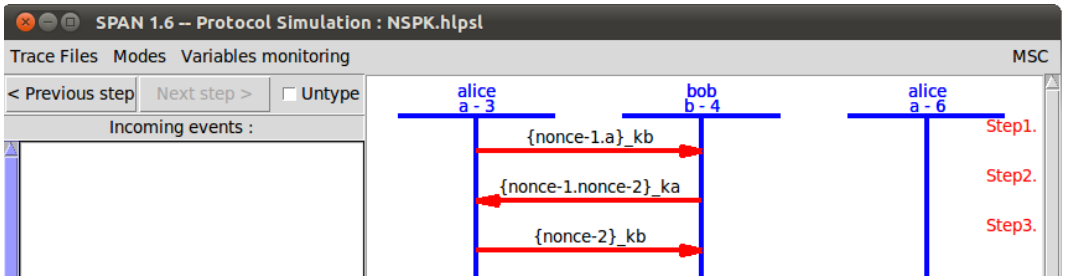

Figure 9: The Needham Schroeder Public Key protocol simulation.

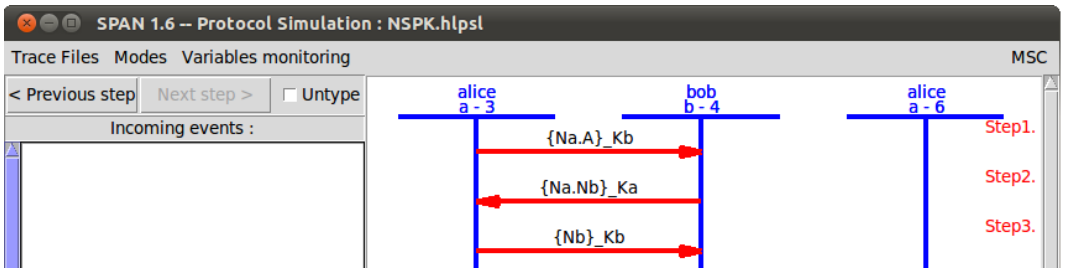

Figure 10: The Needham Schroeder Public Key protocol simulation, with relabelling of arrows.

5 Using the CAS+ syntax to ease writing HLPSL specifications

CAS+ is a simple protocol specification syntax very close to the usual Alice&Bob notation. To see what CAS+ files looks like, open the CAS+ file `simpleProtocol.cas` in SPAN. Here is the complete commented code.

```
protocol simple;
identifiers
A, B, S : user;          %%% Declare all values, agent and keys occurring in the "messages" section
Na, Nb : number;         %%% Variable names should start by a capital letter.
Kab, Kas, Kbs : symmetric_key;

messages
1. A -> S : B, {B, Kab}Kas
2. S -> B : {A, Kab}Kbs


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% In the knowledge part, for each agent, we declare the information
%% he knows BEFORE STARTING THE PROTOCOL SESSION.
%% Known informations can only be variables of the previous "identifier" section.
%% Here for instance, A knows himself, S, B and the long term shared key Kas. But,
%% A DOES NOT KNOW Kab!!! Because, in this protocol, we want Kab to be a session key
%% generated (like a nonce) for each session. Same for B: he does not know Kab
%% befor starting the protocol, he will receive it in a message. S knows both
%% keys Kas and Kbs, but not Kab.

knowledge
A : A, S, B, Kas;
B : A, S, B, Kbs;
S : A, S, B, Kas, Kbs;

%% This "knowledge" section SHOULD NOT STATE ANYTHING ABOUT THE INTRUDER!
%% All sections above "session_instances" define the *** standard protocol ***
%% without intruder. Thus, we never define any intruder or intruder keys
%% etc... They will be defined below.
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
session_instances
[A:alice,B:bob,S:server,Kas:kas,Kbs:kbs]

    %% this is a standard session where alice, bob, server, kas, kbs
    %% are arbitrary constants chosen to instanciate the variables
    %% defined in the ***"knowledge" section*** exclusively. In particular
    %% you should NOT define a constant for Kab, since we want it to
    %% be generated DURING the protocol execution.

[A:i,B:bob,S:server,Kas:kis,Kbs:kbs]
    %% This is a session where the role A is going to be played by
    %% i which is a reserved constant representing the intruder.
    %% Since S and B are played by the same agents than in the previous
    %% session, the SAME constant kbs is used for this second session to denote
    %% the long term secret key between B and S.
    %% Since A is played by i, we create a new constant kis for the key
    %% shared between the intruder and the server.

;   %% The list of sessions ends by a semi-colon.

intruder_knowledge
alice,bob,server,kis; %% the intruder knows alice,bob, the server and kis
```

SPAN provides a last tool which translates CAS+ specifications to HLPSL. Once you loaded the `simpleProtocol.cas` file, you are asked if you want to generate HLPSL. Say yes. If there are mistakes in your CAS+ file, they are displayed in the editing window. Otherwise the CAS+ file loads and the HLPSL is generated. You can consult the generated HLPSL code by clicking on the "View HLPSL" button. As usual, you can also simulate the protocol using "Protocol simulation". **Beware: the CAS+ language permits to state some verification goals, but the translator does not correctly translate them!** So, do not define the verification goals in the CAS+ file. Once you are happy with your CAS+ specification in the protocol simulation, save the generated HLPSL file and **add by hand all the needed verification tags** in the HLPSL file. The tutorial folder contains some other CAS+ examples:

- `NSPK.cas` to see an example with public and private keys.
- `B0.cas` to see a very simple encoding of the old B0 protocol used by offline authentication of credit cards with chips. Additionally to standard Dolev-Yao channels `->`, CAS+ provides some secure channels: $\sim$`>` (write protected) and `=>` (read and write protected). In this file, we use a protected channel `=>` to represent the communication between `A` and `T` (when Alice types its code on the Terminal), between `T` and `C` (when the Terminal sends the code to the Card) and when the Card send the `ok` final message). Those protected channels are, then, encoded into HLPSL using ciphering of those messages with fresh symmetric keys, shared between the agents.
- `Diffie.cas` to see an encoding of the Diffie-Hellman protocol (relying on exponentiation) in CAS+.

6 To configure the shared folder between the virtual and host OS

If you want to have a shared folder between the guest OS (with Ubuntu and SPAN) and your host OS, you have to do the following: in the VirtualBox application click on the virtual machine name, then click on "Configuration", then "Shared folders", then add a permanent virtual shared folder associated to a path on your host OS. You can choose the path on your OS, and VirtualBox chooses a name for the **virtual folder**. Let us denote this virtual folder by the name `myVirtualFolder`. Check the automatic mounting option. Start the Ubuntu virtual machine. When Ubuntu is started, click on the ubuntu menu (upper leftmost corner), choose "Accessories", then "Terminal". In the terminal window, type:

```
sudo mount -t vboxsf -o uid=1000,gid=1000 myVirtualFolder /home/span/Desktop/VBoxShared
```

where you should replace `myVirtualFolder` by the virtual folder name that you noted above. If you are asked for a password simply give the password for the SPAN account: `span`. Now, the `VBoxShared` folder of the desktop should be associated to the path you chose.

7 Going further

More HLPSL and CAS+ specifications are available in the `testsuite` folder.

8 FAQ

- **Is it possible to see HLPSL's silent transitions in "Protocol simulation"?** the answer is no.
- **If there are no events available in the "Incoming events list", how can I have an idea of the sent message?** You can use "Intruder simulation" to send at least messages to the intruder. Then you can guess what is the problem. If there are no messages to send to the intruder, this means that the problem is in the sender's role.
- **I switched variable monitoring on, but I cannot see the "pink rectangle", what is going wrong?** There should be at least one sent message in the simulation for the pink rectangle to show up. If it is impossible to send a message (no messages in the "Incoming events" list), see the answer to the previous question to fix this.