INRIA/IRISA
LANDE Project

SPAN

a Security Protocol ANimator for AVISPA

Version 1.0

User Manual

September 2006

Yann GLOUCHE and Thomas GENET, IRISA/Université de Rennes 1

SPAN: a Security Protocol ANimator for AVISPA

SPAN uses the following libraries:

- HLPSL2IF © **Laurent Vigneron**, 2005
- Matching modulo *exp* and *xor*, part of CL-ATSE © **Mathieu Turuani**, 2005

Contents

1 SPAN overview

1.1 What is AVISPA ?

Avispa [ABB+05, avi] is now a commonly used verification tool for cryptographic protocols. The main advantage of this tool is the ability to use different verification techniques on the same protocol specification.

The protocol designer interacts with the tool by specifying a security problem (ie a protocol paired with a security property that the protocol is expected to achieve) in the High-Level Protocol Specification Language (HLPSL for short [Ohe05]). The HLPSL is an expressive, modular, role-based, formal language that is used to specify control-flow patterns, data-structures, alternative intruder models and complex security properties, as well as different cryptographic primitives and their algebraic properties. These features make HLPSL well suited for specifying modern, industrial-scale protocols.

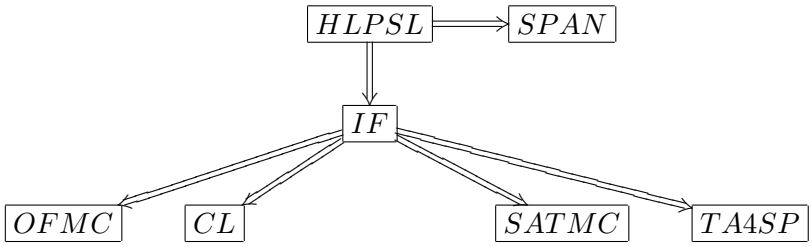


Figure 1: AVISPA system architecture and SPAN

Then, the HLPSL specification is translated into an Intermediate Format (IF) which is used by the various verification tools embedded in AVISPA: OFMC the On-the-Fly Model-Checker [BMV05], CL Constraint-Logic-based model-checker [Tur06], SATMC SAT-based Model-Checker [AC05], and TA4SP Tree Automata based Automatic Approximations for the Analysis of Security Protocols [BHK04]. The role of SPAN is to symbolically execute a HLPSL protocol specification. This turns out to be useful, when writing HLPSL, so as to have a better understanding of the specification, check that it is executable and that it corresponds to what is expected. Figure 1 depicts the overall architecture of the system including SPAN. The initial development of the SPAN tool was done in collaboration with Olivier Heen and Olivier Courtay of Thomson R&D France [GGHC06].

Since HLPSL is a far more expressive language than basic "Alice & Bob" notation, writing an HLPSL specification is still not an easy task. In HLPSL, protocols are defined role by role rather than message by message like it is done using "Alice & Bob" notation. As a result, HLPSL specifications are

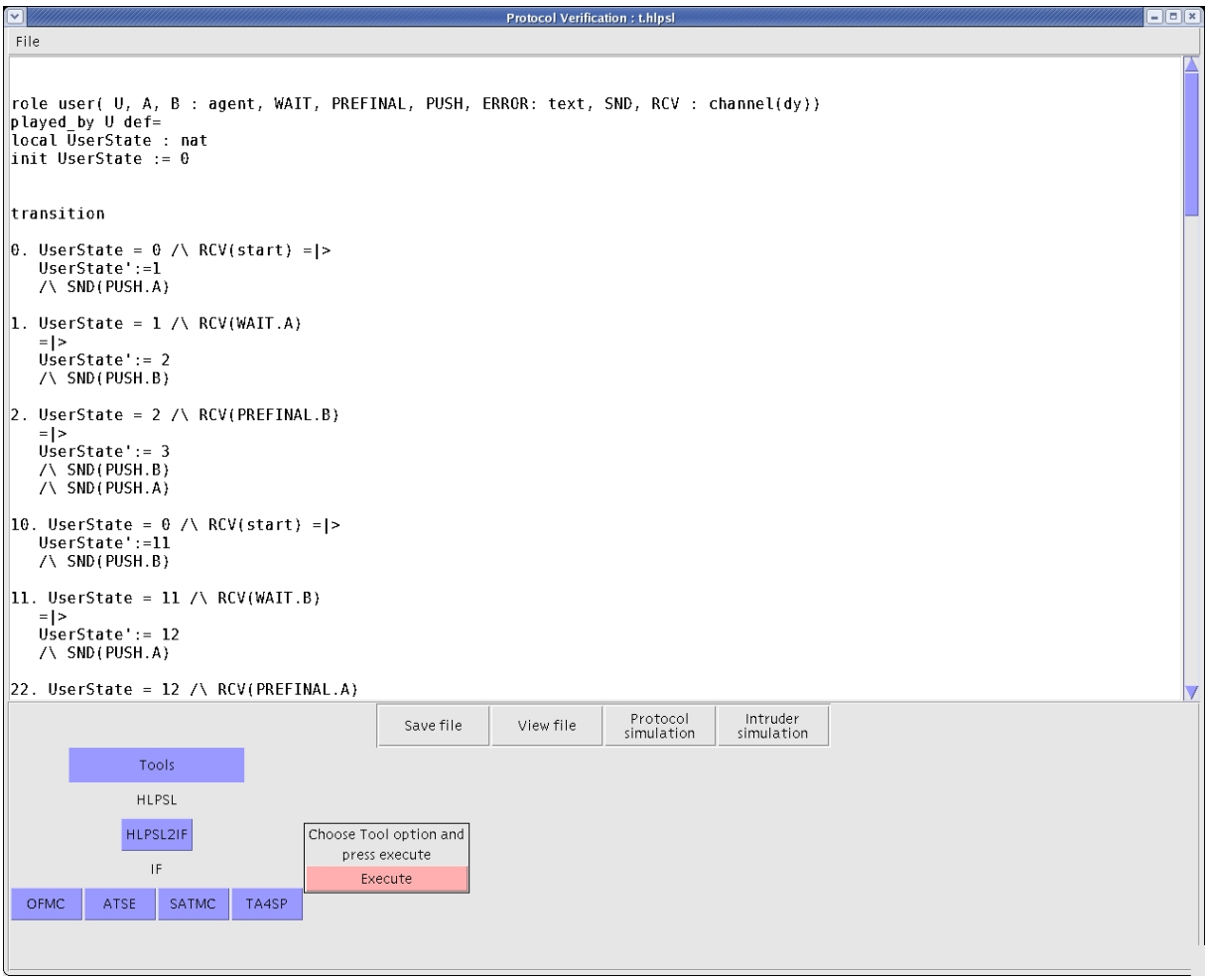


Figure 2: Local version of the AVISPA web interface

far less ambiguous but more difficult to read. Thus, it is sometimes difficult for the protocol designers to figure out if the HLPSL specification they wrote corresponds to the "Alice & Bob" protocol they had in mind.

SPAN helps in interactively producing Message Sequence Charts [HT03] (MSC for short) which can be seen as an "Alice & Bob" trace from an HLPSL specification. SPAN can represent one or more sessions of the protocol in parallel according to the informations given in the role environment. Then, MSCs are produced interactively with the user.

SPAN also includes the possibility to check the values, at every moment, of the variables of each principals : the user chooses the variables of each roles he wants to monitor.

When an intruder is under concern, after each step of protocol execution, SPAN shows the current intruder knowledge and proposes to construct and send malicious messages from this knowledge. Message patterns are proposed to the user conjointly with intruder data, relevant w.r.t. pattern structure and type.

The tool can save and load execution traces corresponding to the execution of the protocol supervised by the user. The MSC can be exported in postscript format or PDF format.

Finally, as shown in Figure 2, SPAN comes with a local version of the web interface of AVISPA that supports the editing of protocol specifications, allows the user to select and configure the back-ends integrated into the tool and launch animations.

1.2 Availability, License and Installation

SPAN is freely available, under the terms of the GNU LIBRARY GENERAL PUBLIC LICENSE.

1.3 Bug report and information

Please report comments and bugs to `Thomas.Genet@irisa.fr`.

2 Installation

SPAN is developped in OCaml. The Tcl/Tk library version 8.4 (for Linux and MacOS) and version 8.3 (Windows) is necessary to use this application.

See `README.txt` file of the distribution for details about installation of binary versions and for compilation of source versions.

3 Specification Language

The HLPSL is an expressive, modular, role-based, formal language that is used to specify control-flow patterns, data-structures, alternative intruder models and complex security properties, as well as different cryptographic primitives and their algebraic properties.

We give a flavour of HLPSL using the specification of the Needham-Schroeder Public Key protocol [NS78]:

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

HLPSL specifications are based on role descriptions, i.e. finite state automata, where transitions are fired when a message is sent or received. With regards to "Alice & Bob" notation, HLPSL makes internal state of roles, nounce generation, message sending and reception explicit. Here is an example of a basic role decalaration extracted from the HLPSL specification of this protocol.

```
role alice (A, B: agent, Ka, Kb: public_key, SND, RCV: channel (dy))
played_by A def=
local State : nat, Na, Nb: text
init State := 0
transition
    0.  State  = 0 /\ RCV(start) =|>
        State'= 2 /\ Na' := new() /\ SND({Na'.A}_Kb)
    2.  State  = 2 /\ RCV({Na.Nb'}_Ka) =|>
        State'= 4 /\ SND({Nb'}_Kb)
end role

role bob(A, B: agent, Ka, Kb: public_key, SND, RCV: channel (dy))
played_by B def=
local State : nat, Na, Nb: text
init State := 1
transition
    1.  State  = 1 /\ RCV({Na'.A}_Kb) =|>
        State'= 3 /\ Nb' := new() /\ SND({Na'.Nb'}_Ka)
    3.  State  = 3 /\ RCV({Nb}_Kb) =|> State'= 5
end role
```

Then, roles are composed together in sessions where the knowledge shared between the roles (public keys for instance) are made explicit.

```
role session(A, B: agent, Ka, Kb: public_key) def=
local SA, RA, SB, RB: channel (dy)
composition
```

```
        alice(A,B,Ka,Kb,SA,RA) /\ bob(A,B,Ka,Kb,SB,RB)
end role
```

Finally, the environment used for protocol execution is defined, where 'i' denotes the intruder. The environment also defines the initial knowledge of the intruder and the initial setting for the sessions, i.e. how many sessions are runned and who run them.

```
role environment() def=
const a, b, c, d  : agent,
ka, kb, ki, kc, kd : public_key,

intruder_knowledge = {a, b, ka, kb, kc, kd, ki, inv(ki)}
composition
        session(a,b,ka,kb) /\ session(c,d,kc,kd) /\ session(a,i,ka,ki)
end role
```

In the example above, four honnest agents are defined, namely `a`,`b`,`c`, and `d`, and the intruder knows all the public keys as well as its own private key `inv(ki)`. For more details about HLPSL refer to [avi, Ohe05].

4 SPAN reference manual

4.1 Local Graphical User Interface for AVISPA

As shown in figure 3, SPAN comes with a local version of the web graphical interface of AVISPA. It looks the same and provides the same features: simple editing of protocol specifications, selection and configuration of the AVISPA verification back-ends and two buttons (those ones are new): Protocol simulation (honnest agents alone) and Intruder simulation (honnest agents and an intruder)

4.2 Basic Protocol Animation

Starting from such an HLPSL specification, such as the one given in Section 3, SPAN helps to build one possible MSC corresponding to that specification. SPAN can represent one or more sessions of the protocol in parallel according to the informations given in the role *environment*. Then, MSCs are produced interactively with the user. At every moment, SPAN proposes to the user to choose between all the transitions for which a message can be sent by a principal and received by another. This approach makes it possible to resolve interactively all the choices that may arise during the construction of MSC (Non-deterministic protocols, choices between two transitions to trigger in two different sessions etc...). The execution of a protocol's transition generally adds a transition on the MSC.

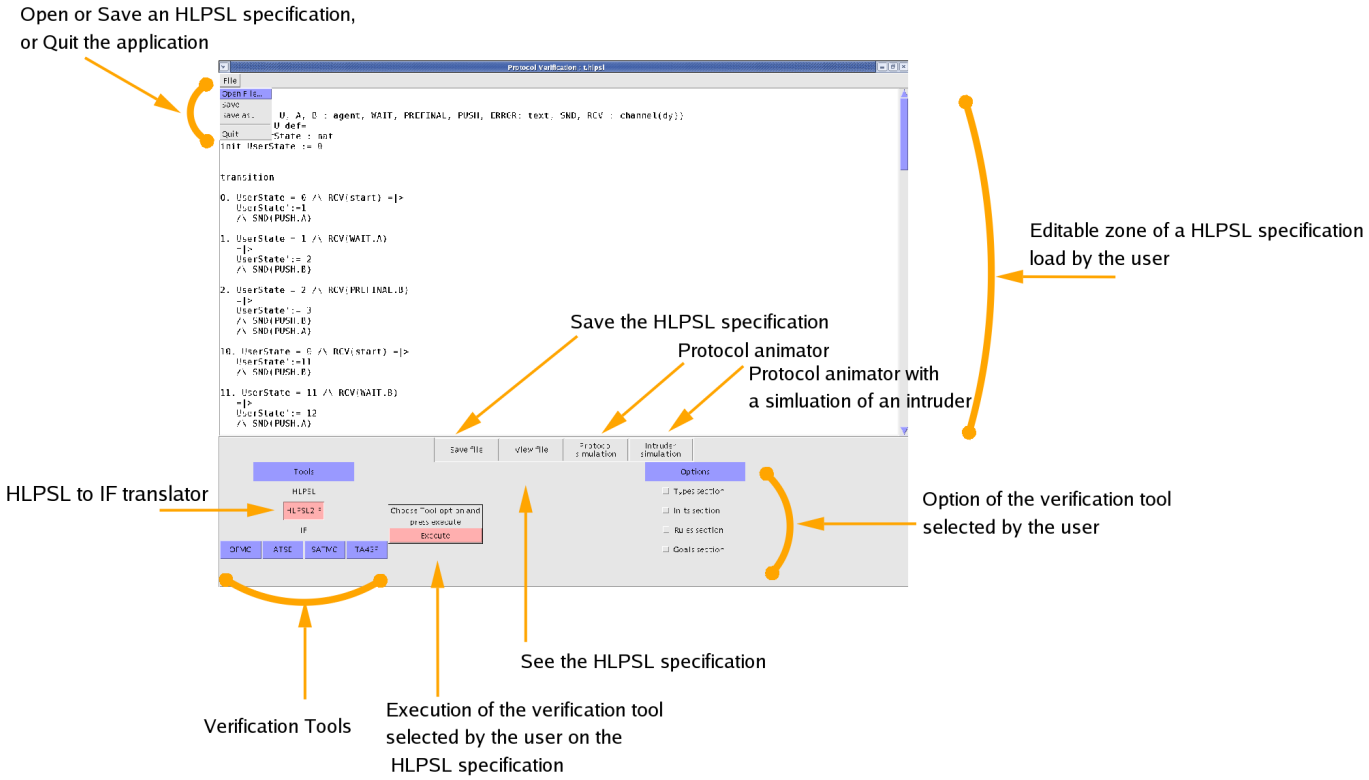


Figure 3: Details of the general verification graphical interface

Figure 4 shows a partial execution of a protocol where the frame *Incoming events* contains all the transitions that can be currently fired (by double click), and the frame *Past events* contains the list of all transitions already fired.

4.3 Variables Monitoring

SPAN also includes the possibility to check the values, at every moment, of the variables of each principals : the user chooses the variables of each roles he wants to monitor.

It is possible to control the values of variables. In the item menu "Variables Monitoring", the user chooses the role for which he wants to control the variables. Then, among all the variables of a given role, he can choose the variables to monitor (See Figure 5(a)). Finally, the user can double click on the rectangular click zones displayed in the MSC in order to display the values at a given step.

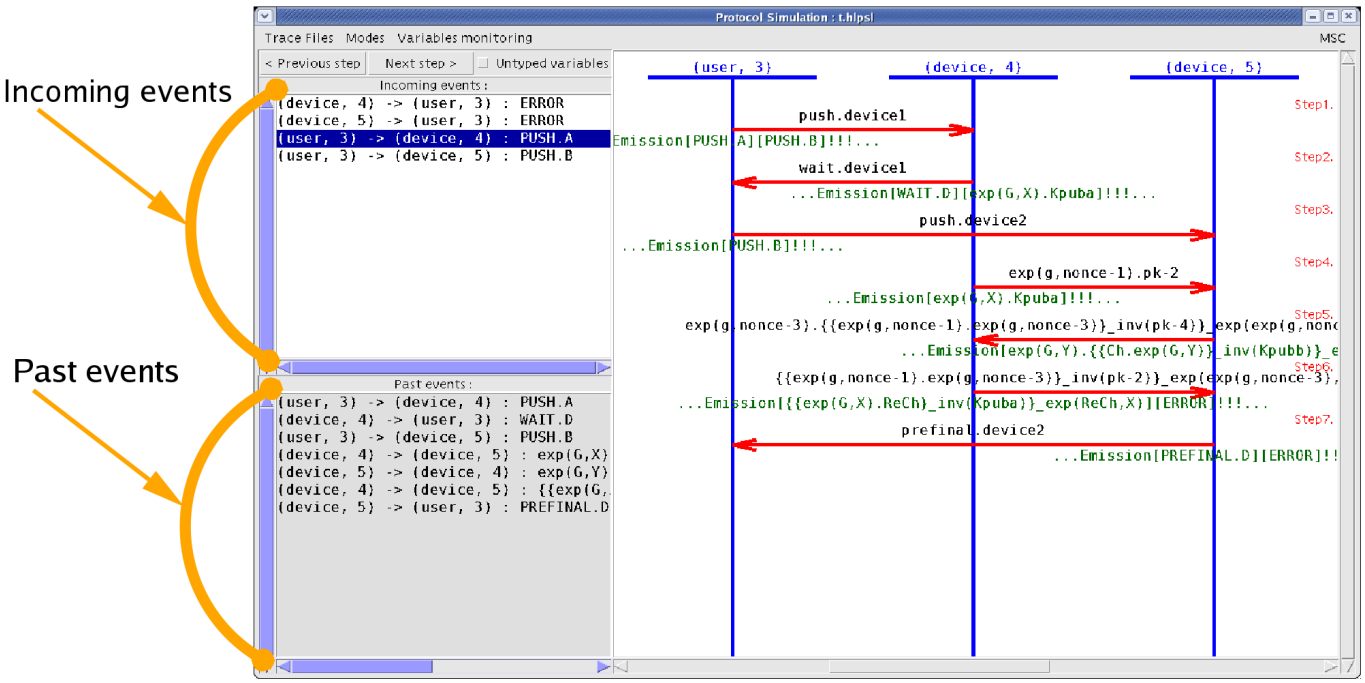
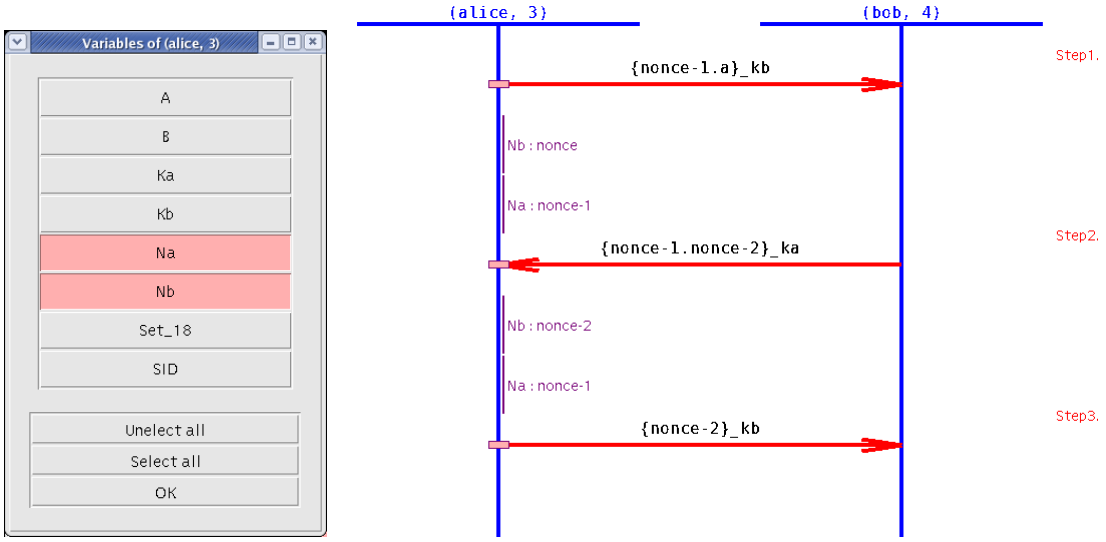


Figure 4: The SPAN animator interface during an interactive execution



(a) Variables selection (b) Content of variables displayed in the MSC

Figure 5: Variables monitoring

4.4 The intruder simulation

It is possible to simulate an intruder who can received all messages, store them in its knowledge, decrypt information if he has the key, build new messages and send them to any other agent. In HLPSL, the intruder is named i, and his initial knowledge is explicitely defined in the specification ($intruder\_knowledge = \{...\}$). After each step of the protocol execution, when the intruder receives a message, he can automatically deduce a new knowledge from its current knowledge and this new message (see Figure 6). For example, if $intruder\_knowledge = \{\{m\}\_k\}$ where m is a message and k is a *symmetric-key* (resp. a *public key*) then, if the intruder receives k (resp. $inv(k)$) then he is abble to deduce and store m. In that case, his knowledge becomes $intruder\_knowledge = \{\{m\}\_k, k, m\}$. In the same way, when the intruder receives a tuple $a.b.c$, then he automatically deduces a, b, and c.

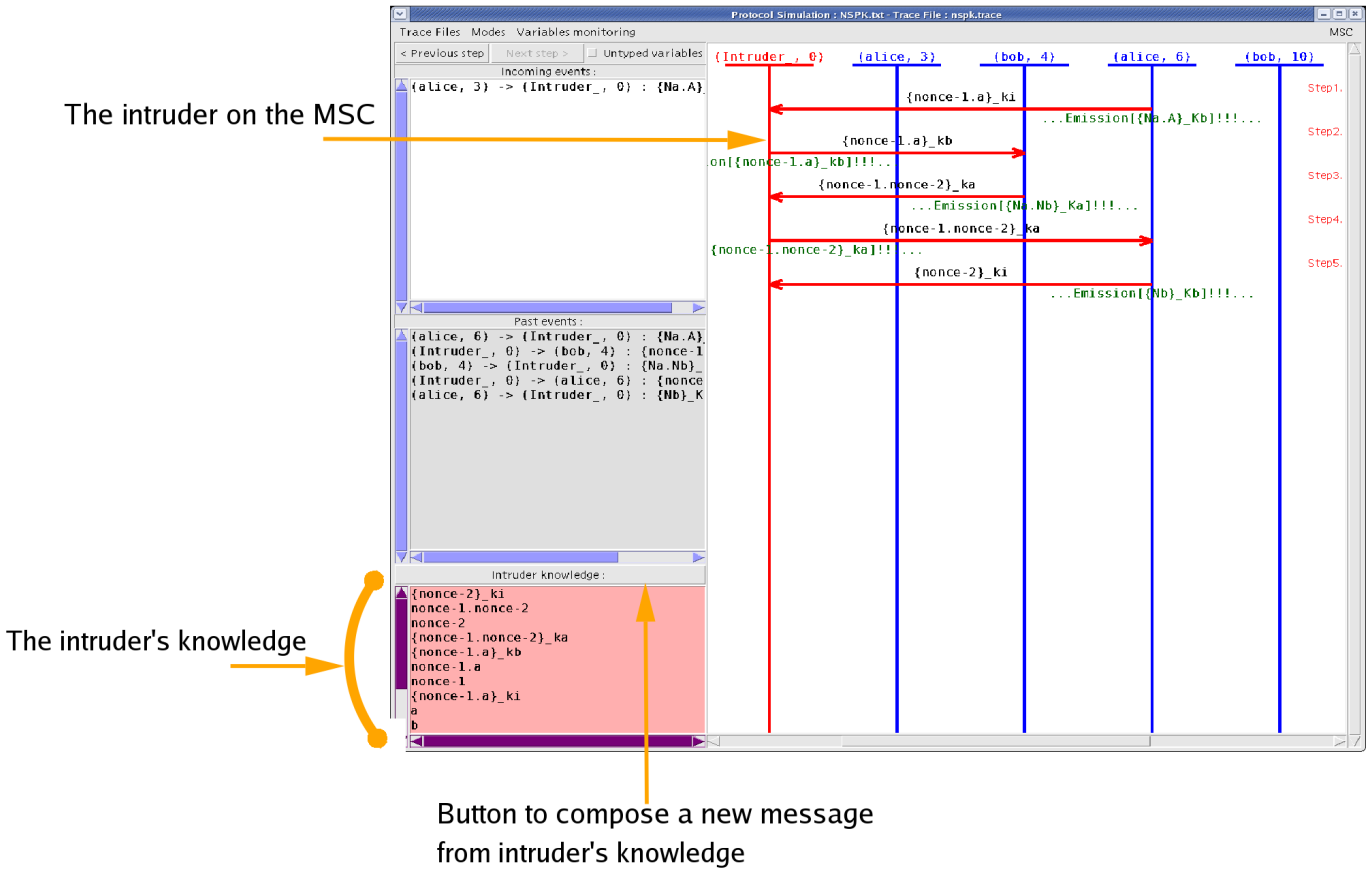


Figure 6: Simulation with the intruder

Using a specific interface (see Figure 7), SPAN can also compose terms

of the intruder's knowledge to generate a new message. In this interface, message patterns and potential receivers are proposed to the user conjointly with intruder data that fit in the pattern holes. In Figure 7, we can see that the upper part of the interface proposes relevant message patterns (messages that are likely to be received by the other principals in the current protocol step). Once a pattern is selected, in the lower part of the interface, the user can select relevant informations to fill the gaps (relevant w.r.t. type, message structure and intruder knowledge). This permits to quickly reconstruct the MSC of a known attack on a protocol or, with a little expertise, to find new ones.

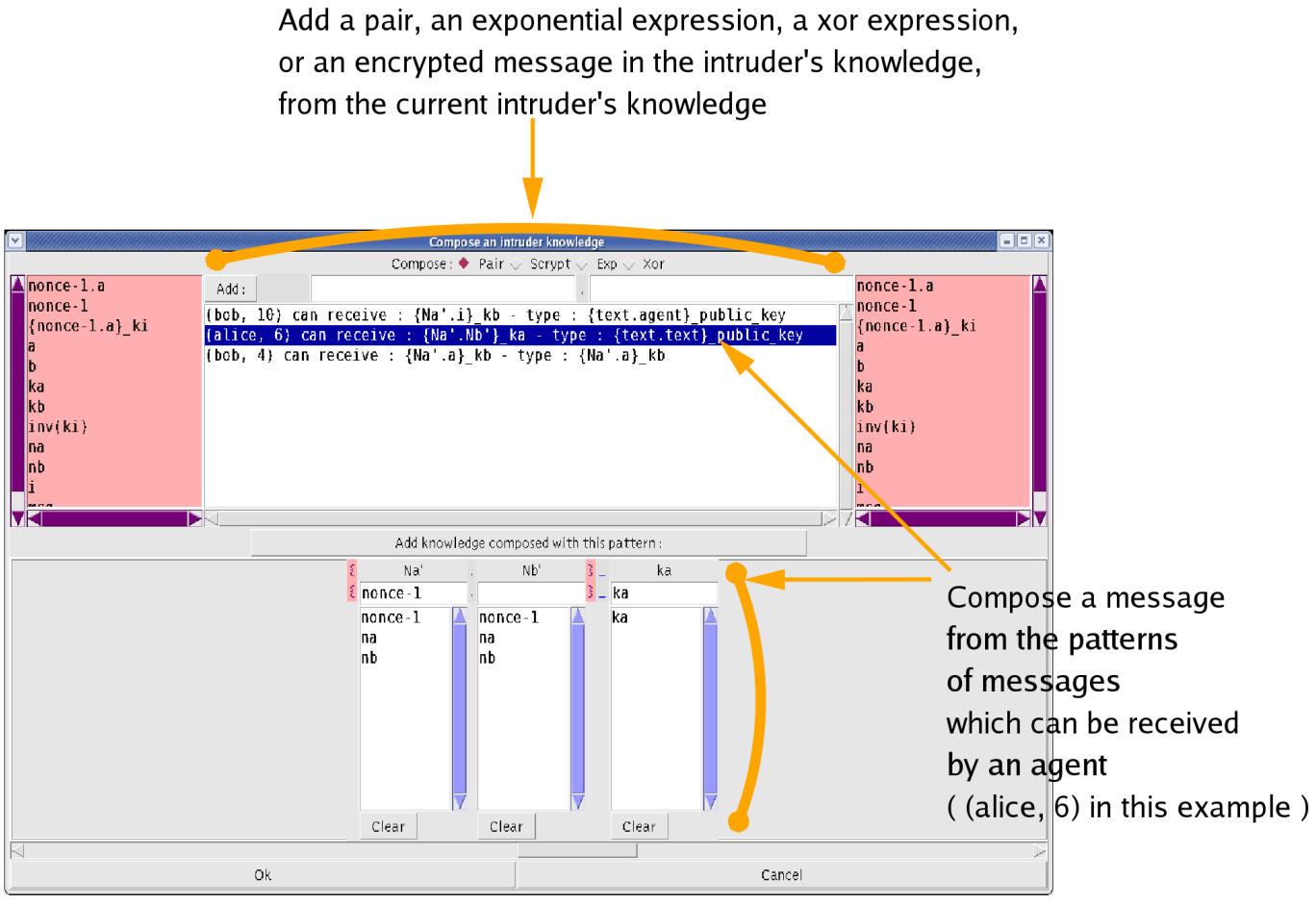


Figure 7: Intruder message construction interface

4.5 Example : the Lowe attack on the Needham-Shroeder Protocol

Recall the protocol given in Section 3. Here are the basic data of the protocol:

- K_A and K_B are two publics keys
- $inv(K_A)$ and $inv(K_B)$ are two privates keys
- N_A and N_B are nonces

The initial knowledge of the intruder contains all the public keys as well as its own public and private key (see Figure 8). The protocol can be attacked if Alice wants to start a protocol session with the intruder.

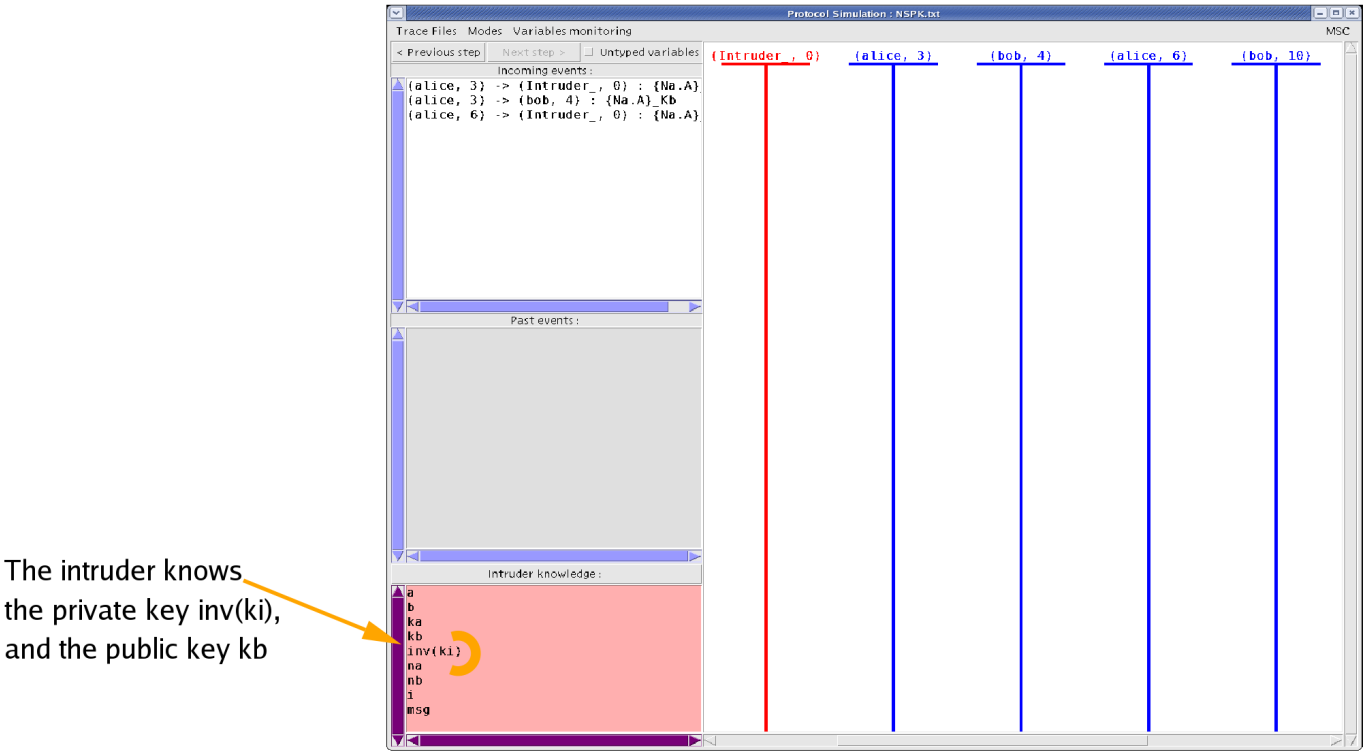


Figure 8: The initial intruder's knowledge

Alice starts the protocol with the Intruder and thus use its public key, K_I, to cipher the message:

$$A \hookrightarrow I : \{N_A, A\}_{K_I}$$

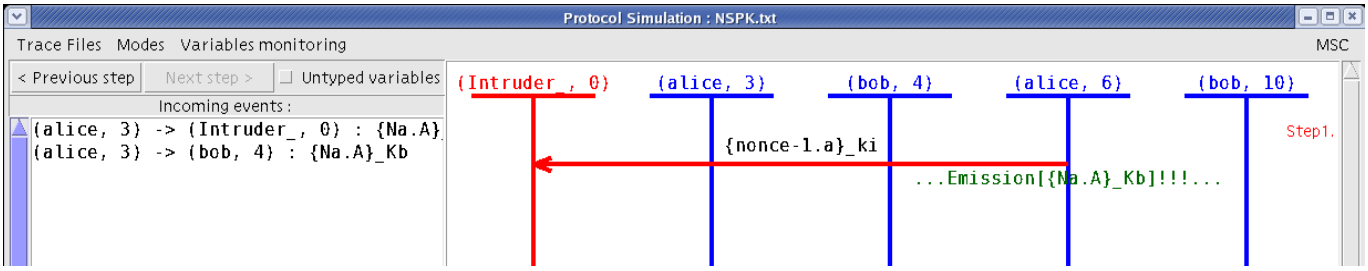


Figure 9: Alice starts the protocol with the intruder instead of Bob

This first step can be simulated in SPAN (see Figure 9).

Then the intruder I can decrypt $\{N_A, A\}_{K_I}$ with $inv(K_I)$, thus he automatically deduces N_A and A (see Figure 10, where in the current session the value of the nonce N_A is `nonce-1`). Since K_B is also known by the intruder, using the message construction interface, the user can build the message $\{N_A, A\}_{K_B}$ whose value in the current session is {`nonce-1.a`}_{`kb`} (see Figure 10).

$$I(A) \hookrightarrow B : \{N_A, A\}_{K_B}$$

The intruder I sends $\{N_A, A\}_{K_B}$ to Bob (B). Thus, Bob thinks that the message comes from Alice though it comes from the intruder. Then, Bob continues to execute the protocol with A using the nonce N_A that A has created for initiating a communication with the intruder. Alices continues to execute the protocol with the intruder (see Figure 12).

Thus, we can build the original attack found by G.Lowe [Low96]:

$$\begin{aligned} A &\hookrightarrow I : \{N_A, A\}_{K_I} \\ I(A) &\hookrightarrow B : \{N_A, A\}_{K_B} \\ B &\hookrightarrow A : \{N_A, N_B\}_{K_A} \\ A &\hookrightarrow I : \{N_B\}_{K_I} \\ I(A) &\hookrightarrow B : \{N_B\}_{K_B} \end{aligned}$$

4.6 Load/Save, Print/Export and Specific display modes

Save/Load For a given protocol, a constructed MSC can be saved, reloaded and replayed in SPAN (See Figure 13).

Print/Export The MSCs can be exported in postscript format or PDF format to be printed or integrated in other documents. (See Figure 14)

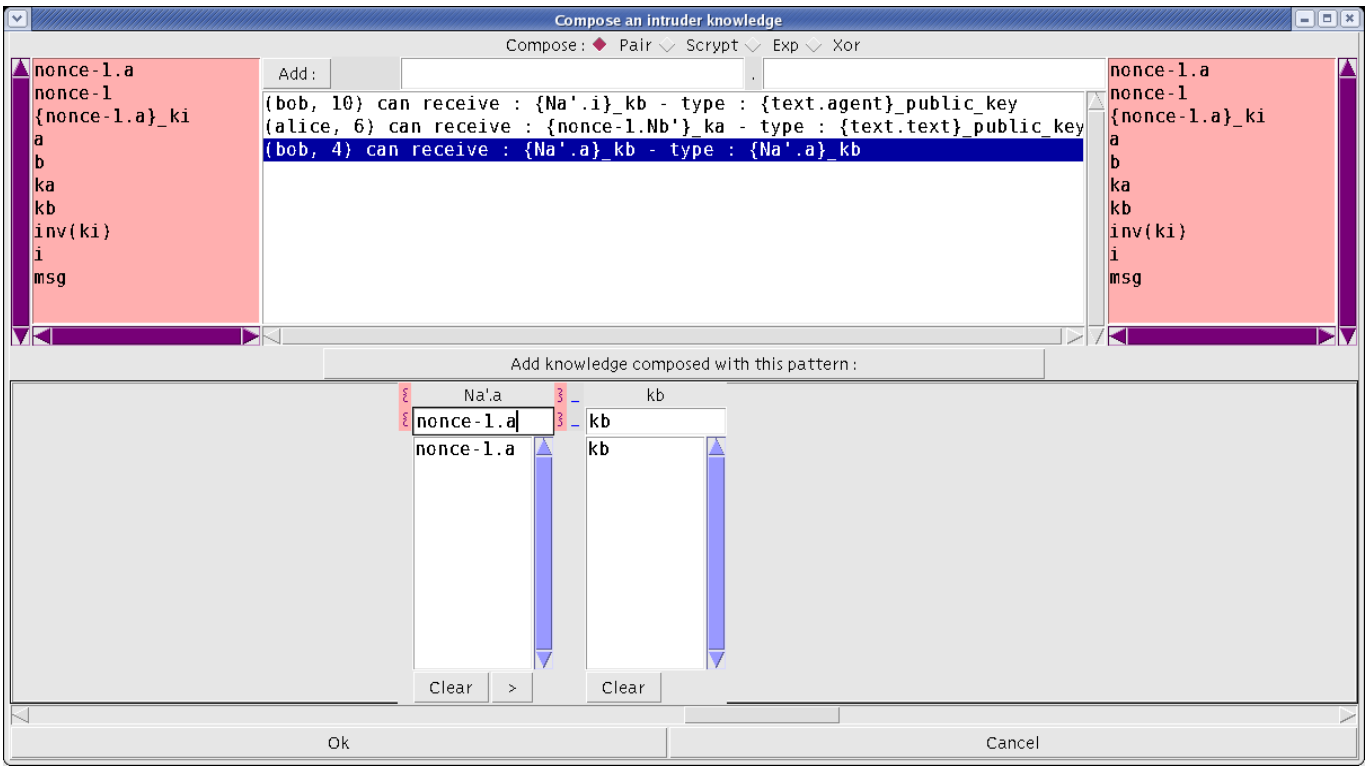


Figure 10: The intruder compose a message for Bob

Display mode Using the options of the *Modes* menu it is possible to display additionnal informations on the MSC: the message pattern of the sender, the message pattern of the receiver, all the message sendings on the network (useful when several messages are sent at once). Each of these elements can be drawn (or not) on the MSC. (See Figure 15)

List of Figures

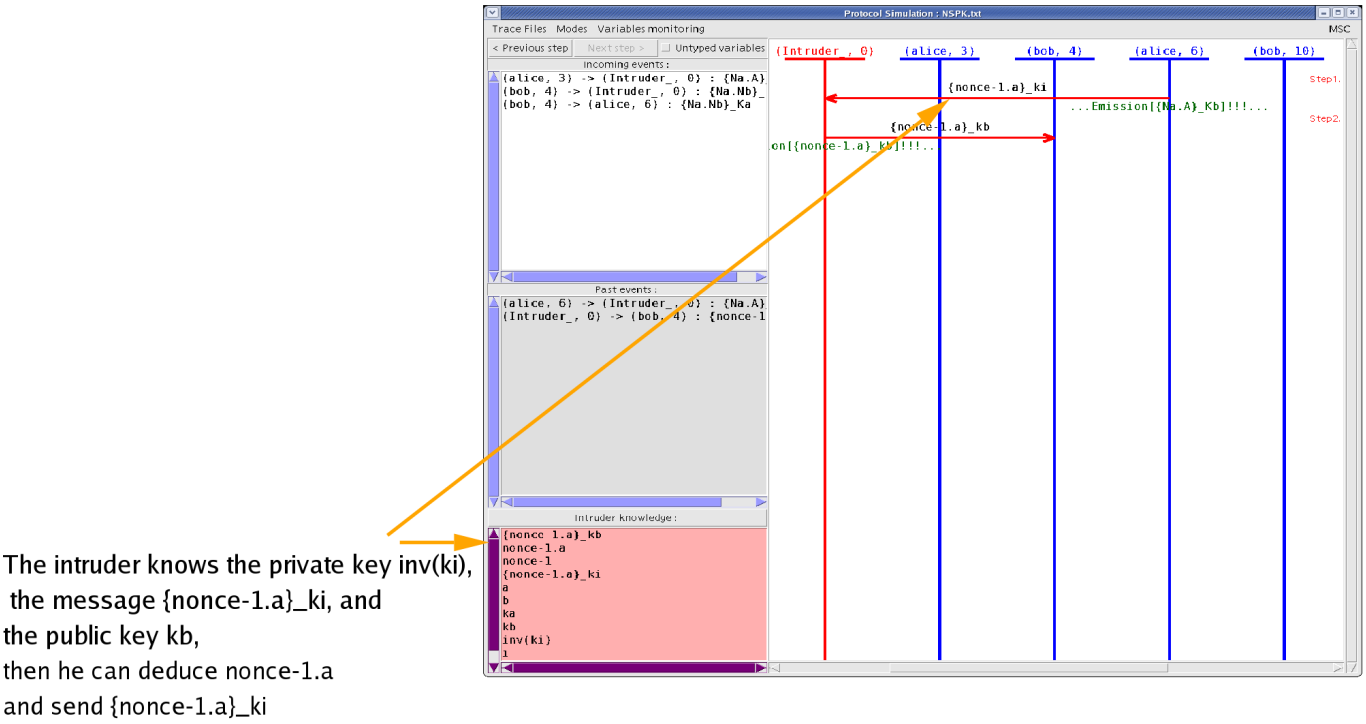


Figure 11: Bob receives a message from the intruder instead of Alice

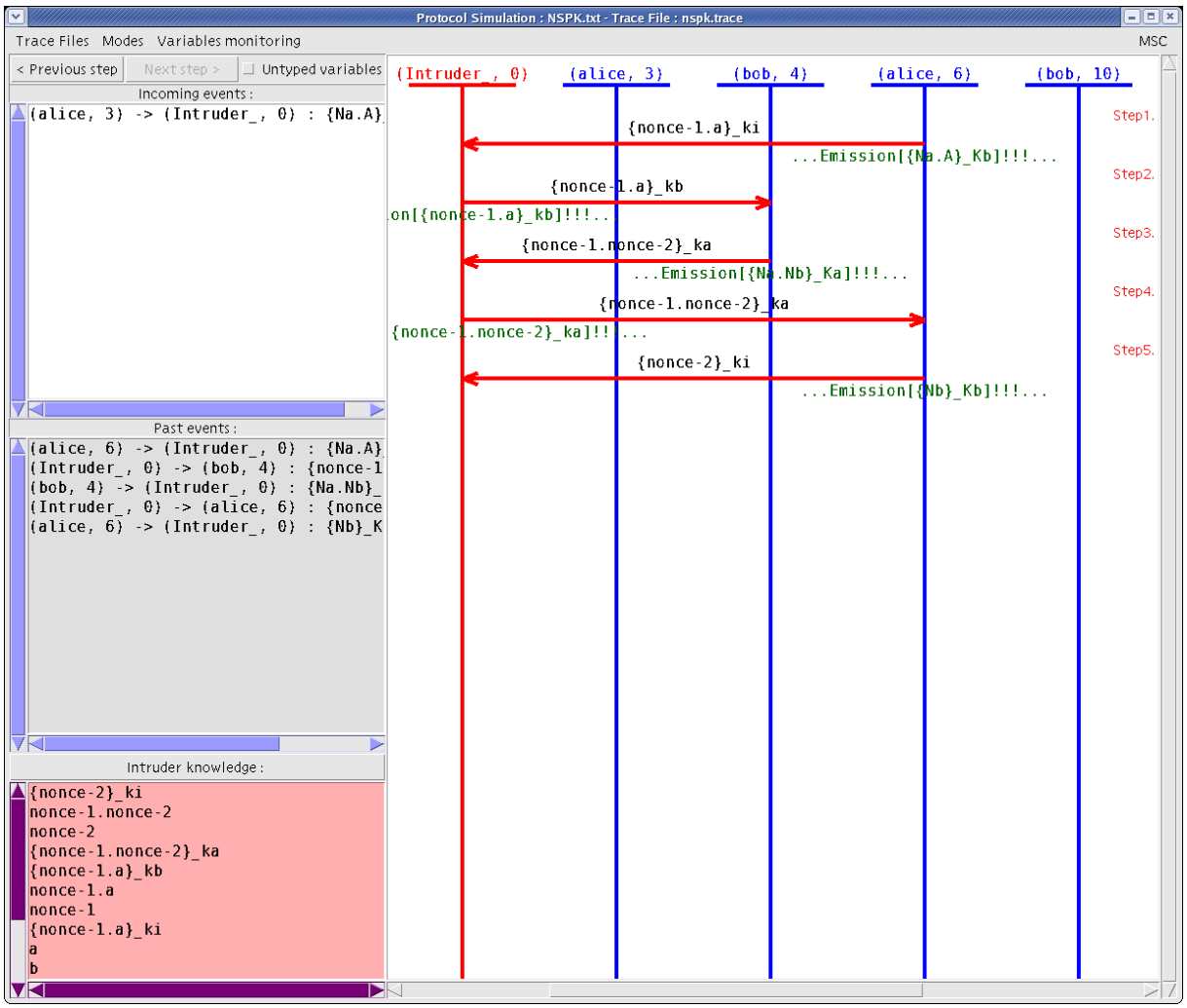


Figure 12: The man in the middle attack continued: the intruder is between Alice and Bob

References

[ABB+05] A. Armando, D. Basin, Y. Boichut, Y. Chevalier, L. Compagna, J. Cuellar, P. Hankes Drielsma, P.-C. Héam, O. Kouchnarenko, J. Mantovani, S. Mödersheim, D. von Oheimb, M. Rusinowitch, J. Santos Santiago, M. Turuani, L. Viganò, and L. Vigneron. The AVISPA Tool for the automated validation of internet security protocols and applications. In K. Etessami and S. Rajamani, editors, *17th International Conference on Computer Aided Ver-*

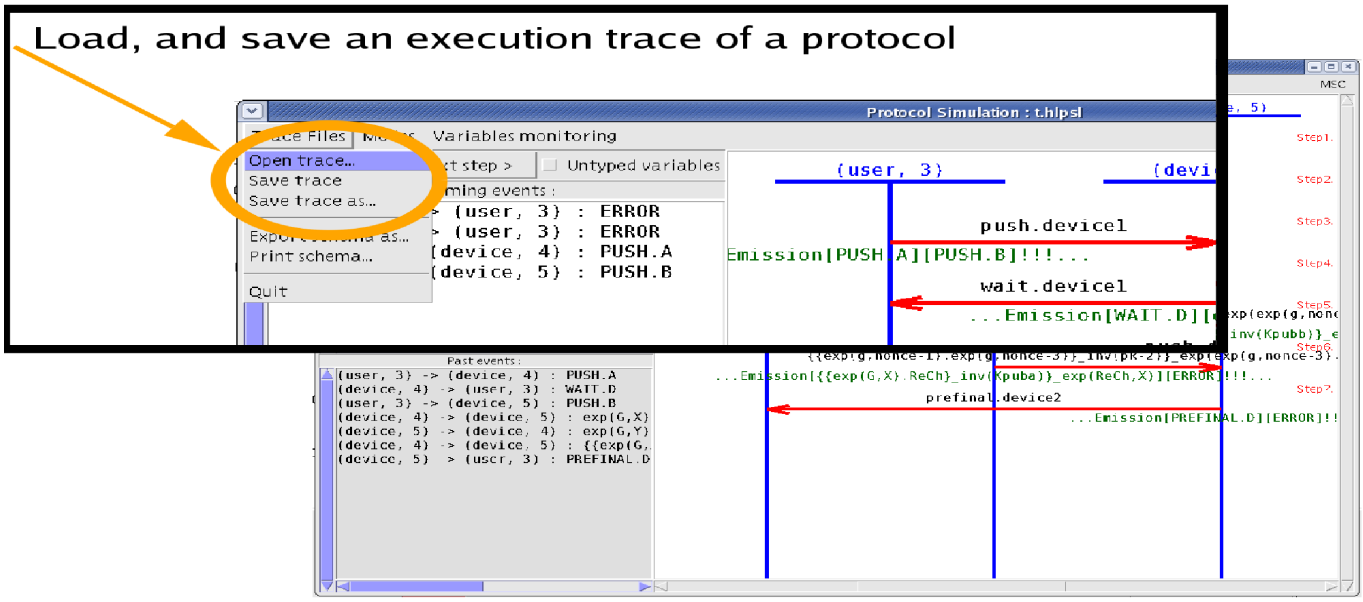


Figure 13: Load and save an execution trace of a protocol

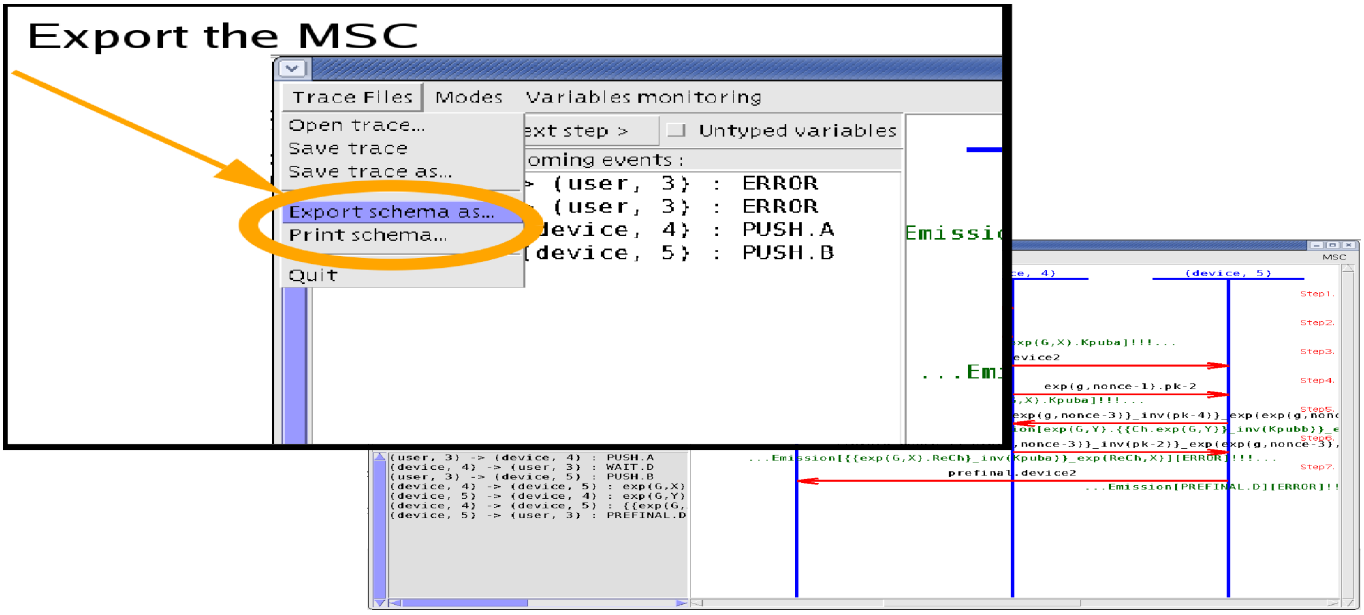


Figure 14: Export a MSC

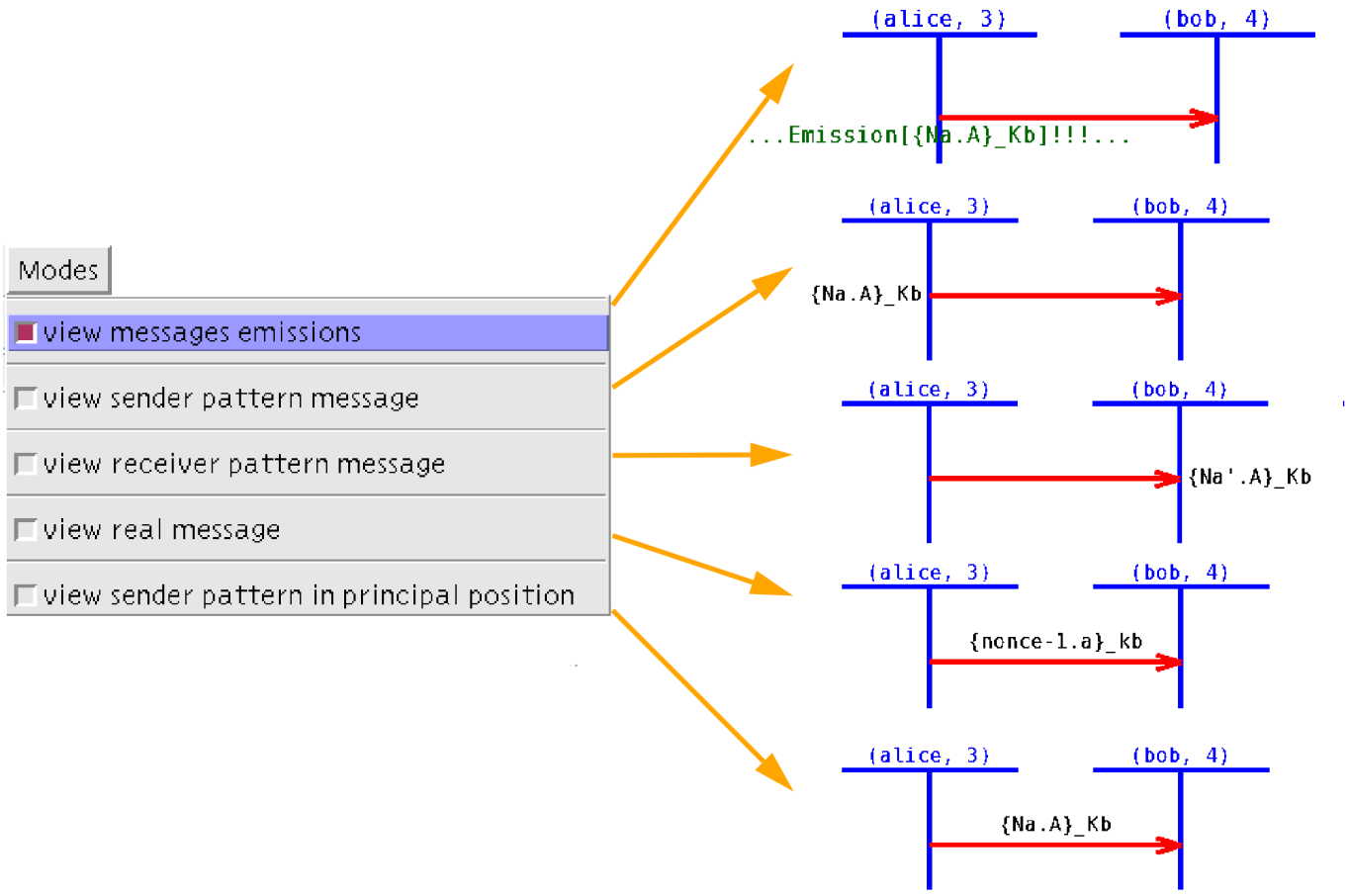


Figure 15: Display modes for messages

ification, CAV'2005, volume 3576 of *Lecture Notes in Computer Science*, pages 281–285, Edinburgh, Scotland, 2005. Springer.

[AC05] Alessandro Armando and Luca Compagna. An optimized intruder model for sat-based model-checking of security protocols. *Electr. Notes Theor. Comput. Sci.*, 125(1):91–108, 2005.

[avi] Avispa – a tool for Automated Validation of Internet Security Protocols. `http://www.avispa-project.org`.

[BHK04] Y. Boichut, P.-C. Héam, and O. Kouchnarenko. Automatic Approximation for the Verification of Cryptographic Protocols. In *Proc. AVIS'2004, joint to ETAPS'04, Barcelona (Spain)*, 2004.

[BMV05] David A. Basin, Sebastian Mödersheim, and Luca Viganò. Ofmc: A symbolic model checker for security protocols. *Int. J. Inf. Sec.*, 4(3):181–208, 2005.

[GGHC06] Y. Glouche, T. Genet, O. Heen, and O. Courtay. A Security Protocol Animator Tool for AVISPA. In *ARTIST-2 workshop on security of embedded systems, Pisa (Italy)*, 2006.

[HT03] D. Harel and P. S. Thiagarajan. Message sequence charts. *UML for Real: Design of Embedded Real-time Systems*, 2003.

[Low96] G. Lowe. Some New Attacks upon Security Protocols. In *9th Computer Security Foundations Workshop*. IEEE Computer Society Press, 1996.

[NS78] R. M. Needham and M. D. Schroeder. Using Encryption for Authentication in Large Networks of Computers. *CACM*, 21(12):993–999, 1978.

[Ohe05] D.von Oheimb. Specification language hlpsl developed in the eu project avispa. In *APPSEM*, 2005.

[Tur06] Mathieu Turuani. The cl-atse protocol analyser. In Frank Pfenning, editor, *RTA*, volume 4098 of *Lecture Notes in Computer Science*, pages 277–286. Springer, 2006.