A Security Protocol Animator Tool for AVISPA

Yann Glouche[1], Thomas Genet[1], Olivier Heen[2], Olivier Courtay[2]

[1] IRISA-INRIA, Rennes, France
yann.glouche@irisa.fr
thomas.genet@irisa.fr
[2] Thomson R&D France, Security Lab
olivier.heen@thomson.net
olivier.courtay@thomson.net

Abstract. Avispa 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. In this paper, we present a protocol animator designed to help protocol developers in writing AVISPA specifications. This is the result of an ongoing joint experiment with Thomson R&D to use AVISPA at early stages of protocol development.

1 The Need for a protocol animator in AVISPA System

In the AVISPA tool, protocols are specified using the High Level Protocol Specification Language (HLPSL for short [1]). 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 [2], CL Constraint-Logic-based model-checker [3], SATMC SAT-based Model-Checker [4], and TA4SP Tree Automata based on automatic approximations For the analysis of Security Protocols [5]. Figure 1 depicts the overall architecture of the system.

Since HLPSL is a far more expressive language than basic "Alice & Bob" notation, writing 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 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.

In this paper, we present a tool for animating HLPSL specifications, i.e. interactively produce Message Sequence Charts (MSC for short) which can be seen as an "Alice & Bob" trace from an HLPSL specification.

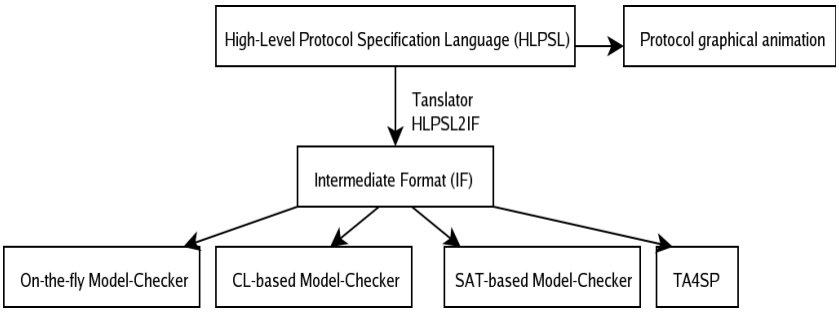

Fig. 1. The overall AVISPA system architecture.

2 The protocol animator

Protocol specifications in HLPSL are divided into roles. The *basic roles*, describe the actions of principals in an execution of the protocol. Other roles, namely composed roles, instantiate several of these basic roles to model sessions of the protocol. Finally, the environment role defines the effective principals and sessions whose execution is to consider.
Here is an example of a basic role decalaration extracted from the HLPSL specification of the Needham-Shroeder protocol with symmetric keys :

```
role a(A : agent, Ka, Kb : symmetric_key,
       SND, RCV : channel(dy))
played_by A def=
local State : nat, Na, Nb : text, B : agent
init State:=0
transition
        step1. State=0 /\ RCV(start)
               =|> State':=1 /\ Na':=new() /\ SND({Na'.A}_Kb)
        step2. State=1 /\ RCV({Na.Nb'}_Ka)
               =|> State':=2 /\ SND({Nb'}_Kb)
end role

role b(B : agent, Ka, Kb : symmetric_key,
       SND, RCV : channel(dy))
played_by B def=
local State : nat, Na,Nb : text, A : agent
init State:=0
transition
        step1. State=0 /\  RCV({Na'.A'}_Kb)
```

```
                =|> State':=1 /\ SND({Na'.Nb'}_Ka)
        step2. State=1 /\ RCV({Nb}_Kb)
                =|> State':=2
end role
```

Two decalarations of composed role :

```
role session(A, B : agent, KaA, KbA, KaB, KbB : symmetric_key)
def=
        local S_A, R_A, S_B, R_B : channel(dy)
        composition
                a(A, KaA, KbB, S_A, R_A)
                /\ b(B, KaB, KbB, S_B, R_B)
end role

role environment()
def=
const alice, bob : agent, ka, kb : symmetric_key
intruder_knowledge={alice, bob}
composition
        session(alice, bob, ka, kb, ka, kb)
        /\ session(alice, bob, ka, kb, ka, kb)
end role
environment()
```

In the example, the composed role *session* describes a single session of the protocol. The composed role *environment* defines two parallel sessions.

Starting from such an HLPSL specification, the protocol animator helps to build one possible MSC corresponding to that specification. The animator 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, the animator 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.

The protocol animator 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.

The tool can save an execution trace corresponding to the execution of the protocol supervised by the user, and it is possible to reload it. The MSC can be exported in postscript format or PDF format.

3 Experiments

We have applied the animator to several protocols : all the protocols of the AVISPA Library, and a new protocol developed by Thomson called User Supervised Device Pairing (USDP for short) [6] for the secure device pairing.

Figure 2 depicts the protocol animator during the execution of Needham Shroeder procotol previously described. The top-left frame contains the incoming event, the bottom-left frame contains the past events, and the right frame contains the MSC.

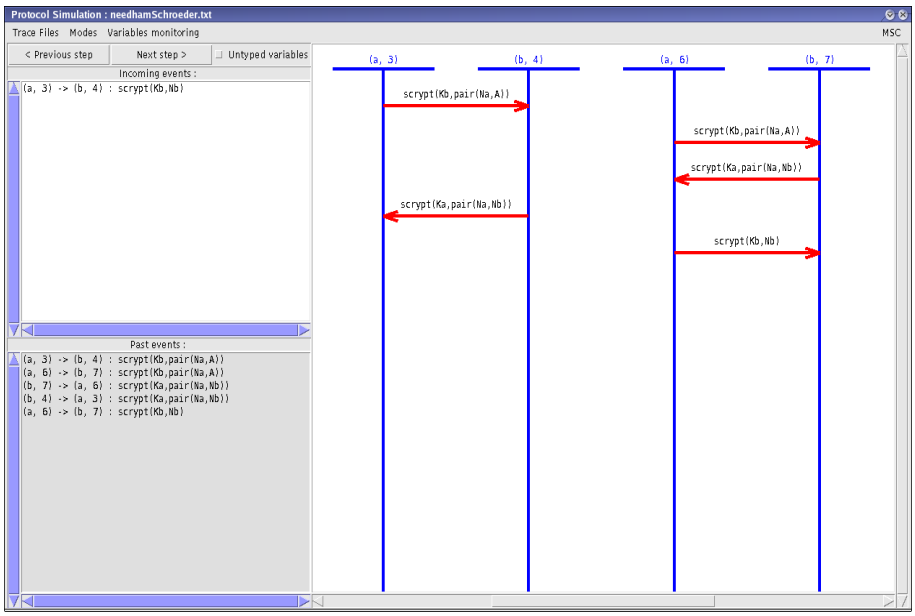

Fig. 2. Graphical interface of the protocol animator.

For all the protocols, the graphical animation corresponds to their specification. The USDP protocol uses three principals : two devices, and a human whose role is to control the execution of the protocol by observing LEDS and by pushing buttons on devices. The devices are two differents instances of a same role specified in HLPSL. This particularity introduce non-determinism, and a lot of choices during the execution for the user of the animator (for example : the possibility to execute an action before another). During the meetings we had with Thomson protocol designers, we used the HLPSL protocol animator to interactively produce MSCs and tune the HLPSL specification to what they expected of the protocol. This permits to quickly reveal and correct misunderstandings remaining in the specification, despite we already agreed on its HLPSL text. We also observed that using the animator gives more confidence to the protocol designers on the final HLPSL specification. Finally, the MSCs commonly used by those engineers in the technical documents and patents on protocols can be produced automatically by the animator from the interactive animation of the HLPSL specification.

Figure 3 depicts the MSC obtained with the animator protocol after the complete execution of the Thomson's USDP protocol corresponding to the following trace.

```
(user, 3)->(device, 4) : pair(PUSH,A)
(device, 4)->(user, 3) : pair(WAIT,D)
(user, 3)->(device, 5) : pair(PUSH,B)
(device, 4)->(device, 5) : pair(exp(G,X),Kpuba)
(device, 5)->(device, 4) : pair(exp(G,Y),
                                pair(scrypt(exp(Ch,Y),
                                     crypt(inv(Kpubb),
                                           pair(Ch,exp(G,Y)))
                                      ), Kpubb))
(device, 4)->(device, 5) : scrypt(exp(ReCh,X),
                                  crypt(inv(Kpuba),
                                        pair(exp(G,X),ReCh)))
(device, 5)->(user, 3) : pair(PREFINAL,D)
(user, 3)->(device, 5) : pair(PUSH,B)
(user, 3)->(device, 4) : pair(PUSH,A)
```

4 Further Work

Some features of HLPSL are not yet taken into account. In fact, the aggregate types : tuples, list, and sets and all functions relating to this

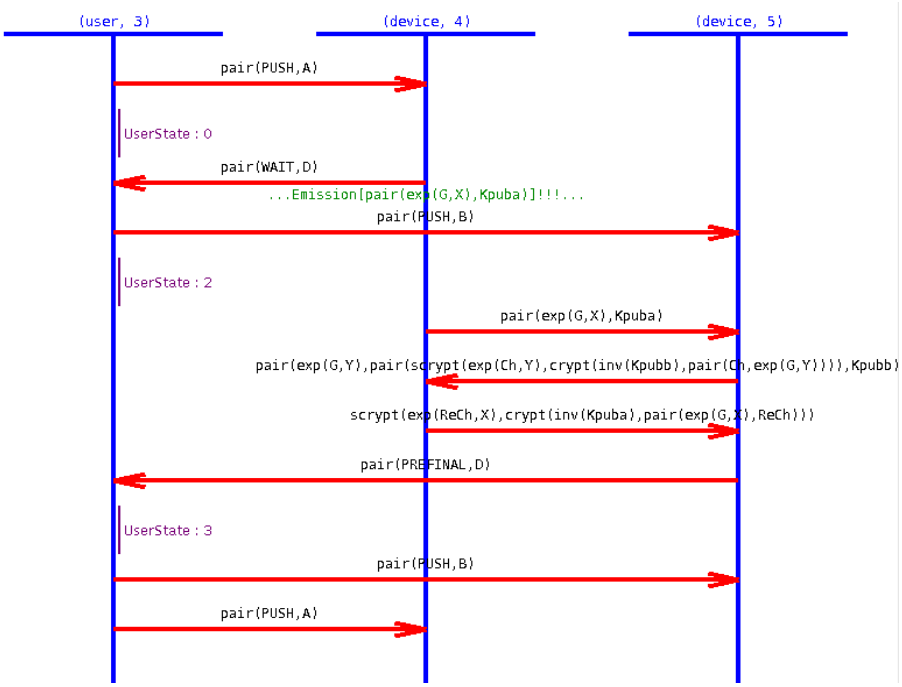

Fig. 3. MSC obtained after the execution of Thomson's USDP protocol

types (*cons*, *in*, *delete*), are not yet treated by the animator. The function *new*() is not fully managed. We hope to solve these problems soon.

Furthermore, to determine which messages can be sent by a principal and received by another, a correct treatment of mathematical functions is necessary to compare the sent message and the received message. This is not yet fully functional when messages include *exp*, *xor*.

As soon as possible we will integrate a mode to replay interactively the attacks. For this, we want to refine the animator in order to be able to execute an intruder role who can receive, replay, and treat all messages sent by an agent to another.

References

1. **D.von Oheimb**. *The High-Level Protocol Specification Language HLPSL developed in the EU project AVISPA*. In Proceedins os APPSEM 2005 Workshop, September 13, 2005.
2. **D.Basin**,**S.Mödersheim**,**L.Vigano**. *A symbolic model checker for security portocols*. International Journal of Information Security 4(3):181:208, 2005.

3. **M.Turuani**. *Securite des Protocoles Cryptographiques : Decidabilite et Complexite*. Phd, Universite Henri Poincare, Nancy, December 2003.
4. **A.Armando**. *SAT-based Model-Checking of Security Protocols*. PhD Thesis, Universita degli Studi di Genova and the University of Edinburg, September2005.
5. **Y.Boichut, P-C.O.Kouchnarenko, F.Oehl**. *Improvements ont the Genet and Klay Technic to Automatically Verify Security Protocols*.
In Proc.AVIS'04, ENTCS.
6. **O.Courtay, O.Heen, M.Karroumi, A.Durand**. *Secure device pairing under realistic conditions*. Applied Cryptography and Network Security (ACNS'06).
7. **AVISPA**. *Deliverable 2.3 : The Intermediate Format*. `http://avispa-project.org/publications.html`, 2003.
8. **AVISPA**. *The AVISPA User Manual*. `http://avispa-project.org/publications.html`, 2005.