Analysing Security Protocols with AVISPA

Laura Takkinen
Helsinki University of Technology
Laura.Takkinen@tkk.fi

Abstract

Usage of the Internet has increased rapidly over the past decade. Increased amount of network users has also brought along a need of network-based services that require security. Developing new security protocols is a difficult task and sometimes too difficult task for human mind. We need an efficient tools to help the development and verification of the protocols. In this paper we introduce the protocol analysis tool called AVISPA. The paper describes the architecture of the tool and syntax for protocol specification language called HLPSL. We give an example of how a real protocol can be specified with the HLPSL language and how the output of the AVISPA Tool is analysed.

KEYWORDS: protocol analyzer, AVISPA

1 Introduction

Usage of the Internet has increased dramatically over the past decade. Increased amount of network users has also brought along a need of network-based services that require security. Developing new security protocols is a difficult task. Designing protocols is error prone and finding possible vulnerabilities from the protocols is sometimes too difficult for human mind. This is a serious problem for standardization organizations such as the Internet Engineering Task Force (IETF), the International Telecommunication Union (ITU) and the World Wide Web Consortium, but also for several companies that whose business depends on rapid standardization and correctness of the standardized security protocols [6].

To make the development of the protocols faster and to improve their security, it is important to have appropriate tools that support the analysis of the protocols and help to find the vulnerabilities in early stages of development. [6].

Security protocols can be seen as a mathematical objects that require tools that use methods of mathematics and logic to perform analysis [4]. In this paper we present a push-button tool for formal analysis of security protocols, Automated Validation of Internet Security Protocols and Applications (AVISPA).

The rest of the paper is organized as follows: Section 2 introduces the AVISPA tool, its architecture and syntax for presenting security protocols. Section 3 shows an example of how real protocol can be analysed with AVISPA. Finally there are conclusions.

2 AVISPA

Automated Validation of Internet Security Protocols and Applications (AVISPA) is a push-button tool for building and analysing security protocols. In this section we introduce the tool and the principals it is based on. AVISPA provides a role-based, expressive formal language for protocol specification and it integrates four different back-ends, which perform the actual analysis of the protocol. We begin the discussion by presenting the architechture of the tool and then show the syntax of the formal language. Finally we briefly show the four back-ends that AVISPA uses.

2.1 Architecture

The archtecture of AVISPA is shown in figure 2.1. First step in using the tool is to present the analyzed protocol in a special language called High Level Protocol Specification Language (HLPSL). We discuss the HLPSL language more detailed in following section 2.2.

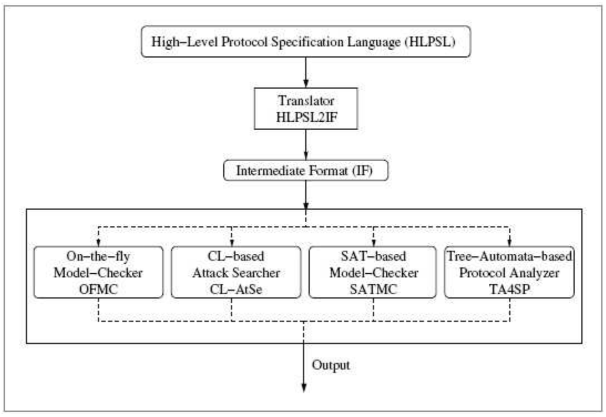


Figure 1: The architecture of the AVISPA tool [6].

The HLPSL presentation of the protocol is translated into the lower level language called Intermediate Format (IF). This translations is performed by the translator called HLPSL2IF. This step is totally transparent to the user.

IF presentation of the protocol is used as an input to the four different back-ends: On-the-fly Model-Checker (OFMC), CL-based Attack Searcher (CL-AtSe), SAT-based Model-Checker (SATMC) and Tree-Automata-based Protocol Analyzer (TA4SP). These back-ends perform the analysis and output the results in precisely defined output format stating whether there are problems in the protocol or not.

2.2 High Lavel Protocol Specification Language

AVISPA uses High Level Protocol Specification Language (HLPSL) to present the analysed protocols. In this section we take a closer look into the structure of HLPSL language according to the AVISPA tutorial [5].

In order to express the protocols in HLPSL language, it is easiest to translate the protocols firts into A-B format, for instance:

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

The notation above illustrates Wide Mouth Frog (WMF) protocol, where endpoints A and B setup a secure session. First A generates a new session key Kab, encrypts it by using a key Kas and transmits the encrypted key to the trusted server S. Kas is a key that is shared between A and S. S decrypts the message, re-encrypts it by using a shared key Kbs and transmits the encrypted message to the B. B can decrypt the message by using the shared secret Kbs and obtains the session key Kab.

HLPSL language is a role-based language, which means that actions of each participant are defined in a separate module, called a *basic role*. In the case of WMF example above, the basic roles are: alice (A), bob (B) and server (S). Basic roles describe what information the corresponding participant has initially (parameters), its initial state and how the state can change (transitions). To continue the WMF example, the role of alice would be expressed in following way:

```
role alice(A,B,S : agent,
           Kas : symmetric_key,
           SND, RCV : channel (dy))
played_by A def=
  local
   State: nat,
   Kab: symmetric_key
  init State := 0
  transition
...
end role
```

The role indicates that *agents* A, B and S are participating to the procol suite, A has a shared key Kas with the agent S and A uses channels SND (send) and RCV (receive) to communication. Dolev-Yao (dy) is the the intruder model that is assumend for the communication channel. Section called local defines the local variables of alice, which are State that is described by a natural number (na) and symmetric key Kab. Initial state of the alice is 0.

Transition section describes received and sended messages and how they affect the state of the role. For instance the role server has following transition called step1:

```
step1. State  = 0 /\ RCV({Kab'}_Kas) =|>
       State':= 2 /\ SND({Kab'}_Kbs)
```

The transition means that if the server's state is 0 and it receives a message from its RCV channel containing a key Kab' that is encrypted with a key Kas, the server changes its state to 2, encrypts the key Kab' with the Kbs and sends the encrypted key to the channel SND.

In addition to basic roles the HLPSL language defines also so called *composition roles* that are used to combine several basic roles. Combining the basic roles means that the roles can execute parallel. Composition roles define the actual protocol sessions. For instance, in the case of WMF protocol there are three basic roles alice, bob and server. Composition role, called session, initiates one instance of each role and thus defines one protocol run. Composition role does not define transitions such as basic roles do, instead they initiate basic roles and defines channels used by the basic roles. Composition role is defined for instance following way:

```
role session(A,B,S   :agent,
             Kas,Kbs :symmetric_key) def=
local SA, RA, SB, RB SS, RS: channel (dy)
composition
   alice (A, B, S, Kas, SA, RA)
/\ bob (B, A, S, Kbs, SB, RB)
/\ server(S, A, B, Kas, Kbs, SS, RS)
end role
```

Finally the HLPSL defines a top level role, called here as environment, that contains global variables and combines several sessions. This top level role can be used to define what information an intruder has and where the intruder can access the protocol. For example, the intruder may play a role of a legitimate user in a protocol run. Following role definition shows how a top level environment can be defined. Letter i in the definition indicates the intruder.

```
role environment()
def=

  const a, b, s       : agent,
kas, kbs, kis : symmetric_key

intruder_knowledge = {a, b, s, kis}

composition
   session(a,b,s,kas,kbs)
/\ session(a,i,s,kas,kis)
/\ session(i,b,s,kis,kbs)

end role
```

Every security protocol has some goals which it is supposed to meet. In order to write the protocol in HLPSL format, we must know these goals. The analysis is done against the defined security goals and results indicate whether the protocol meets the goals or not.

Security goals of the protocol are presented in HLPSL language in section called *goals*. Security goals are actually defined in transition sections of basic roles. The definitions of security goals in transition section are called *goal facts*. The goals section simply describes which combinations of these goal facts indicate an attack. [6].

Below there is an example of a goal fact textttsecret. The notation means that bob allows that the key K1 can be shared with alice, but it must remain secret between the two. The second argument of the `secret` fact is called *protocol id* and it simply names the secret fact and distinguish the different secrurity goals from each other.

```
role bob {
...
   local
     State : nat,
     Nb,Na : text,
     K1 : message

   init
     State := 1

   transition
     1. State  = 1 /\ RCV({Na'}_K) =|>
        State':= 3 /\ Nb'  := new()
                   /\ SND({Nb'}_K)
                   /\ K1':= Hash(Na'.Nb')
                   /\ secret(K1',k1,{A,B})
...
end role
```

A goal section of the protocol definition can be as follows:

```
goal
   secrecy_of k1
   authentication_on bob_alice_nb
end goal
```

The first statement describer the goal fact above and the second statement describes another goal fact that was not included in the example. We do not show the syntax in transition section for this secrurity goal. However, this statement is used to indicate the authentication. Notation `bob_alice_nb` is simply used to name the corresponding goal facts in transition sections of basic roles.

2.3 Back-ends

As figure 2.1 shows AVISPA integrates four different back-ends. Here the word back-end means an entity that inputs a sequence of IF language, does analysis and produces the analysis output.

The four different back-ends used in AVISPA, OFMC, CL-AtSe, SATMC and TA4SP, are complementary rather that equivalent. Thus, the output of the back-ends may differ.

All back-ends assume *perfect cryptography*, which means that attacker cannot solve ecryption without the knowledge of the whole key. Also, the tranmission channel is assumed to be controlled by a Dolev-Yao attacker. This means, that the attacker has basically full control over the channel. [6]

3 Case Study: IKEv2

The AVISPA project [3] provides a library of known protocols. The AVISPA tool has been evaluated by running it against the library of 33 known protocols. During the evaluation, the tool detected a number of previously unknown attacks on some of these protocols. One of the protocols was Internet Key Exchange Protocol version 2 (IKEv2) with digital signatures [2].

In this section we give an example of how the protocol is build and analyzed with the tool. We use the IKEv2 protocol in this example.

3.1 IKEv2 with digital signatures

The IKEv2 protocol variat we are using here proceeds in two so-called exchanges. These exchanges are called `IKE_SA_INIT` and `IKE_SA_AUTH`. These two exchanges consist of four messages. The first pair of messages (`IKE_SA_INIT`) negotiate cryptographic algorithms, exchange nonces, and do a Diffie-Hellman exchange. The second pair (`IKE_SA_AUTH`) authenticate the first two messages, exchange identities and certificates, finally establish the first so-called "child security association" or `CHILD_SA`. This association is the one that will be used to secure the subsequent IPsec tunnel. [2]

Parts of the `IKE_SA_AUTH` messages are encrypted and integrity protected with keys established through the `IKE_SA_INIT` exchange, so the identities are hidden from eavesdroppers and all fields in all the messages are authenticated.

Below is shown the A-B notation of the IKEv2 authentication protocol [3]:

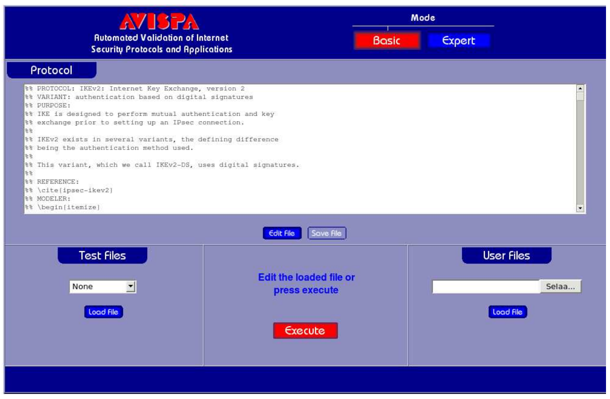
```
IKE_SA_INIT
1. A -> B: SAa1, KEa, Na
2. B -> A: SAb1, KEb, Nb
IKE_SA_AUTH
3. A -> B: {A, AUTHa, SAa2}K
   where K = H(Na.Nb.SAa1.g^KEa^KEb) and
     AUTHa = {SAa1.g^KEa.Na.Nb}inv(Ka)
4. B -> A: {B, AUTHb, SAb2}K
   where
     AUTHb = {SAb1.g^KEb.Na.Nb}inv(Kb)
```

In `IKE_SA_INIT` phase peers exchange nonce values Na and Nb, Diffie-Hellman keys KEa and KEb. In addition, SAa1 contains A's cryptosuite offers and SAb1 B's preference for the establishment of the `IKE_SA`. Similarly SAa2 and SAb2 for the establishment of the `CHILD_SA` [3].

After the A-B notation, the protocol is more easier to convert to HLPSL format. Appendix 1 shows the HLPSL specification as a whole for the protocol. In this example we use AVISPA Web Tool [1] to perform analysis remotely. Figure 3.1 shows a front page of the Web Tool. Local HLPSL files can easily be imported to the Web Tool. In this figure IKEv2 HLPSL file has been imported and the tool is ready for analysis.

Analysis results show that the protocol security goals are not met. Output of the OFMC back-end includes following attack trace:

```
ATTACK TRACE
i -> (a,6): start
(a,6) -> i: SA1(1).exp(g,DHX(1)).Ni(1)
i -> (b,3): SA1(1).exp(g,DHX(1)).Ni(1)
(b,3) -> i: SA1(1).exp(g,DHY(2)).Nr(2)
i -> (a,6): SA1(1).exp(g,DHY(2)).Nr(2)
```


AVISPA
Automated Validation of Internet Security Protocols and Applications
Mode
Basic
Expert
Protocol
Edit File
Save File
Test Files
None
Load File
Edit the loaded file or press execute
Execute
User Files
Selaa...
Load File

```
%% PROTOCOL: IKEv2: Internet Key Exchange, version 2
%% VARIANT: authentication based on digital signatures
%% PURPOSE:
%% IKE is designed to perform mutual authentication and key
%% exchange prior to setting up an IPsec connection.
%%
%% IKEv2 exists in several variants, the defining difference
%% being the authentication method used.
%%
%% This variant, which we call IKEv2-DS, uses digital signatures.
%%
%% REFERENCE:
%% \cite{ipsec-ikev2}
%% MODELER:
%% \begin{itemize}
```

Figure 2: AVISPA Web Tool

```
(a,6) -> i: {a.{SA1(1).exp(g,DHX(1))
     .Ni(1).Nr(2)}_inv(ka).SA2(3)}_
     (f(Ni(1).Nr(2).SA1(1)
     .exp(exp(g,DHY(2)),DHX(1))))
i -> (b,3): {a.{SA1(1).exp(g,DHX(1))
     .Ni(1).Nr(2)}_inv(ka).SA2(3)}_
     (f(Ni(1).Nr(2).SA1(1)
     .exp(exp(g,DHX(1)),DHY(2))))
(b,3) -> i: {b.{SA1(1).exp(g,DHY(2))
     .Nr(2).Ni(1)}_inv(kb).SA2(3)}_
     (f(Ni(1).Nr(2).SA1(1)
     .exp(exp(g,DHX(1)),DHY(2))))
```

In the attack trace `i` stands for intruder and `a` and `b` correspond alice and bob endpoints. The number in notation `(a,6)` is a session number related to the internal workings of the HLPSL2IF translator and the AVISPA back-ends.

The attack trace describes a man-in-the-middle attack where intruder initiates an association with alice but forwards the messages to the bob. Endpoint alice thinks that it has an assiociation with intruder and does not know that she is actually participating an association with bob.

Attack traces can also be viewed more readable format, Message Sequence Chart (MSC) format. Figure 3.1 shows a same attack trace presented in MSC chart.

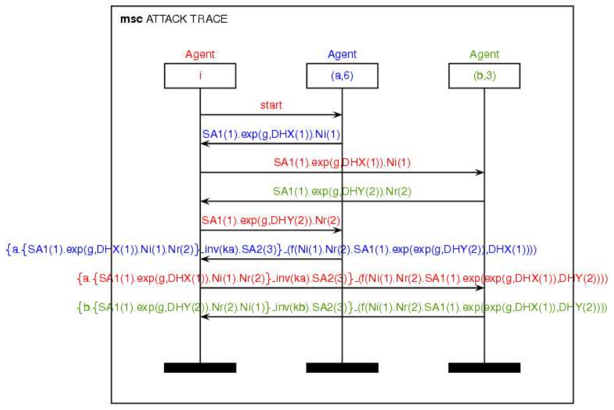

msc ATTACK TRACE
Agent
i
Agent
(a,6)
Agent
(b,3)
start
SA1(1).exp(g,DHX(1)).Ni(1)
SA1(1).exp(g,DHX(1)).Ni(1)
SA1(1).exp(g,DHY(2)).Nr(2)
SA1(1).exp(g,DHY(2)).Nr(2)
{a.{SA1(1).exp(g,DHX(1)).Ni(1).Nr(2)}_inv(ka).SA2(3)}_(f(Ni(1).Nr(2).SA1(1).exp(exp(g,DHY(2)),DHX(1))))
{a.{SA1(1).exp(g,DHX(1)).Ni(1).Nr(2)}_inv(ka).SA2(3)}_(f(Ni(1).Nr(2).SA1(1).exp(exp(g,DHX(1)),DHY(2))))
{b.{SA1(1).exp(g,DHY(2)).Nr(2).Ni(1)}_inv(kb).SA2(3)}_(f(Ni(1).Nr(2).SA1(1).exp(exp(g,DHX(1)),DHY(2))))

Figure 3: Man-in-the-middle attack presented in MCS chart. [1]

4 Conclusions

This paper introduces the AVISPA protocol analysis tool with the help of an use case of IKEv2. AVISPA provides a powerfull specification language, HLPSL, for protocols and integrates four different back-ends that perform the actual protocol analysis.

The AVISPA Tool is quite promising tool for formal protocol analysis. Although it requires deep knowledge of the analyzed protocols, afterall mistakes in the HLPSL specification of the protocol may cause that the back-ends cannot analyze the protocol or the output may give wrong results. The tool is also quite diffucult to use. Basically a user has to learn a new programming language in order to use the tool. Also the analysing of the output and possible attack trace is time consuming.

References

[1] AVISPA Project. AVISPA Web Tool. At `http://www.avispa-project.org/web-interface/`.

[2] C. Kaufman, Ed. Internet Key Exchange (IKEv2) Protocol. RFC 4306, Internet Engineering Task Force, December 2005.

[3] A. Project. AVISPA Automated Validation of Internet Security Protocols and Applications. At `http://www.avispa-project.org`.

[4] Sebastian Mödersheim and Luca Vigano and David von Oheimb. Automated Validation of Security Protocols (AVASP). Lecture slides, April 2005.

[5] The AVISPA team. HLPSL Tutorial The Beginner's Guide to Modelling and Analysing Internet Security Protocols. Technical report, AVISPA project, June 2006.

[6] L. Vigano. Automated Security Protocol Analysis With the AVISPA Tool. *Electronic Notes in Theoretical Computer Science*, (155):62–86, 2006.

A HLPSL Specification for IKEv2 Protocol

```
role alice(A,B:agent,
           G: text,
           F: hash_func,
           Ka,Kb: public_key,
           SND_B, RCV_B: channel (dy))
played_by A
def=

  local Ni, SA1, SA2, DHX: text,
        Nr: text,
        KEr: message, %% more specific: exp(text,text)
        SK: hash(text.text.text.message),
        State: nat

  const sec_a_SK : protocol_id

  init  State := 0

  transition

  %% The IKE_SA_INIT exchange:
  %% We have abstracted away from the negotiation of cryptographic
  %% parameters.  Alice sends a nonce SAi1, which is meant to
  %% model Alice sending only a single crypto-suite offer.  Bob must
  %% then respond with the same nonce.
  1. State = 0  /\ RCV_B(start) =|>
     State':= 2 /\ SA1' := new()
                /\ DHX' := new()
                /\ Ni' := new()
                /\ SND_B( SA1'.exp(G,DHX').Ni' )

  %% Alice receives message 2 of IKE_SA_INIT, checks that Bob has
  %% indeed sent the same nonce in SAr1, and then sends the first
  %% message of IKE_AUTH.
  %% As authentication Data, she signs her first message and Bob's nonce.
  2. State = 2  /\ RCV_B(SA1.KEr'.Nr') =|>
     State':= 4 /\ SA2' := new()
                /\ SK' := F(Ni.Nr'.SA1.exp(KEr',DHX))
                /\ SND_B( {A.{SA1.exp(G,DHX).Ni.Nr'}_(inv(Ka)).SA2'}_SK' )
                /\ witness(A,B,sk2,F(Ni.Nr'.SA1.exp(KEr',DHX)))





  3. State = 4  /\ RCV_B({B.{SA1.KEr.Nr.Ni}_(inv(Kb)).SA2}_SK) =|>
     State':= 9 /\ secret(SK,sec_a_SK,{A,B})
                /\ request(A,B,sk1,SK)

end role



role bob (B,A:agent,
          G: text,
          F: hash_func,
          Kb, Ka: public_key,
          SND_A, RCV_A: channel (dy))
```

```
played_by B
def=

  local Ni, SA1, SA2: text,
        Nr, DHY: text,
        SK: hash(text.text.text.message),
        KEi: message,
        State: nat

  const sec_b_SK : protocol_id

  init  State := 1

  transition

  1. State = 1  /\ RCV_A( SA1'.KEi'.Ni' ) =|>
     State':= 3 /\ DHY' := new()
                /\ Nr' := new()
                /\ SND_A(SA1'.exp(G,DHY').Nr')
                /\ SK' := F(Ni'.Nr'.SA1'.exp(KEi',DHY'))
                /\ witness(B,A,sk1,F(Ni'.Nr'.SA1'.exp(KEi',DHY')))

  2. State = 3  /\ RCV_A( {A.{SA1.KEi.Ni.Nr}_(inv(Ka)).SA2'}_SK ) =|>
     State':= 9 /\ SND_A( {B.{SA1.exp(G,DHY).Nr.Ni}_(inv(Kb)).SA2'}_SK )
                /\ secret(SK,sec_b_SK,{A,B})
                /\ request(B,A,sk2,SK)

end role



role session(A, B: agent,
             Ka, Kb: public_key,
             G: text,
             F: hash_func)
def=

  local SA, RA, SB, RB: channel (dy)

  composition
           alice(A,B,G,F,Ka,Kb,SA,RA)
        /\ bob(B,A,G,F,Kb,Ka,SB,RB)

end role



role environment()
def=

  const sk1,sk2     : protocol_id,
        a, b        : agent,
        ka, kb, ki  : public_key,
        g           : text,
        f           : hash_func

  intruder_knowledge = {g,f,a,b,ka,kb,i,ki,inv(ki)

                       }
  composition

        session(a,b,ka,kb,g,f)
     /\ session(a,i,ka,ki,g,f)
     /\ session(i,b,ki,kb,g,f)

end role



goal

  %secrecy_of SK
  secrecy_of sec_a_SK, sec_b_SK % Addresses G9

  %Alice authenticates Bob on sk1
  authentication_on sk1 % Addresses G1, G2, G3, G7, G10
  %Bob authenticates Alice on sk2
  authentication_on sk2 % Addresses G1, G2, G3, G7, G10

end goal


environment()
```