Available online at www.sciencedirect.com

SCIENCE DIRECT®

Electronic Notes in Theoretical Computer Science 155 (2006) 61–86

Electronic Notes in Theoretical Computer Science

www.elsevier.com/locate/entcs

Automated Security Protocol Analysis With the AVISPA Tool[1]

Luca Viganò[2]

Information Security Group
Department of Computer Science
ETH Zurich
Switzerland

Abstract

The AVISPA Tool is a push-button tool for the Automated Validation of Internet Security Protocols and Applications. It provides a modular and expressive formal language for specifying protocols and their security properties, and integrates different back-ends that implement a variety of automatic protocol analysis techniques. Experimental results, carried out on a large library of Internet security protocols, indicate that the AVISPA Tool is a state-of-the-art tool for Internet security protocol analysis as, to our knowledge, no other tool exhibits the same level of scope and robustness while enjoying the same performance and scalability.

Keywords: Security protocols, protocol models, automated protocol validation.

1 Introduction

With the spread of the Internet and network-based services, and the development of new technological possibilities, the number and scale of new security protocols under development is out-pacing the human ability to rigorously

[1] The work presented in this paper was supported by the FET Open Project IST-2001-39252 and the BBW Project 02.0431, "AVISPA: Automated Validation of Internet Security Protocols and Applications" (www.avispa-project.org). I thank all the people who participated to the AVISPA project and contributed to the development of the AVISPA Tool, in particular: A. Armando, D. Basin, Y. Boichut, Y. Chevalier, L. Compagna, J. Cuellar, P. Hankes Drielsma, P.C. Heám, O. Kouchnarenko, J. Mantovani, S. Mödersheim, D. von Oheimb, M. Rusinowitch, J. Santiago, M. Turuani, and L. Vigneron.

[2] Email: vigano@inf.ethz.ch

doi:10.1016/j.entcs.2005.11.052

analyze and validate them. This is an increasingly serious problem for standardization organizations like the Internet Engineering Task Force (IETF), the International Telecommunication Union (ITU), and the World Wide Web Consortium (W3C), as well as for companies whose products and/or services depend on the rapid standardization and correct functioning of these protocols. It is also a problem for users of these technologies whose rights and freedoms, such as the right to privacy of personal data, depend on a secure infrastructure.

Designing secure protocols is a hard problem. In open networks, such as the Internet, protocols should work even under worst-case assumptions, e.g., that messages may be eavesdropped or tampered with by an *intruder* (often also called the *attacker* or the *adversary*). Severe attacks can be conducted even without breaking cryptography, but by exploiting weaknesses in the protocols themselves; for instance, by carrying out "masquerading attacks", where an attacker impersonates an honest agent, or "replay attacks", where messages from one protocol session (i.e., one execution of the protocol) are used in another session. The possibility of these attacks sometimes stems from subtle misconceptions in the design of the protocols. Typically, these attacks are simply overlooked, as it is difficult for humans, even by careful protocol inspection, to determine all the complex ways that different protocol sessions could be interleaved together, with the possible interference of a malicious intruder.

To speed up the development of the next generation of security protocols, and to improve their security, it is thus of the utmost importance to have tools that support the rigorous analysis of security protocols by either finding flaws or establishing their correctness. Optimally, these tools should be completely automated, robust, expressive, and easily usable, so that they can be integrated into the protocol development and standardization processes.

In the last decade, we have thus witnessed the development of a large number of new techniques for the formal analysis of security protocols. Many (semi-)automated security protocol analysis tools have been proposed (e.g., [3,21,23,26,27,28,41,42,43,61,64,66,69,71,73,76]), which can analyze small and medium-scale protocols such as those in the Clark/Jacob library [39]. However, scaling up to large, industrial-scale security protocols is both a scientific and a technological challenge. The *AVISPA Tool*[3] is a push-button tool

[3] The AVISPA Tool has been developed jointly by the four partners of the project "AVISPA: Automated Validation of Internet Security Protocols and Applications" [10]: the research team lead by A. Armando at the University of Genova, Italy; the team lead by M. Rusinowitch at INRIA-Lorraine, Nancy, France; the team lead by D. Basin at the ETH Zurich, Switzerland; and the team lead by J. Cuellar at SIEMENS AG, Munich, Germany. The AVISPA Tool significantly extends the scope, effectiveness, and performance of its predeces-

for the *A*utomated *V*alidation of *I*nternet *S*ecurity-sensitive *P*rotocols and *A*pplications which rises to this challenge in a systematic way by

(i) providing a modular and expressive formal language for specifying security protocols and properties, the *High-Level Protocol Specification Language HLPSL*, and

(ii) integrating different back-ends that implement a variety of automatic analysis techniques ranging from *protocol falsification* (by finding an attack on the input protocol) to *abstraction-based verification* methods for infinite numbers of sessions.

In order to assess the strength of the AVISPA Tool, and to demonstrate proof-of-concept on a large collection of practically relevant, industrial protocols, we have given HLPSL specifications of a substantial set of security protocols currently being drafted or standardized by organizations like the IETF, along with the security properties these protocols are expected to enjoy. The result of this specification effort is the *AVISPA Library* (also publicly available at the AVISPA web-site [10]), which currently comprises 215 security problems derived from 33 industrial-scale security protocols. We have assessed the AVISPA Tool by running it against the AVISPA Library, and the results indicate that, to the best of our knowledge, no other tool exhibits the same scope and robustness while enjoying the same performance and scalability. In particular, the AVISPA Tool has detected a number of previously unknown attacks on some of the protocols analyzed, e.g., on some protocols of the ISO-PK family, on the IKEv2 protocol with digital signatures, on the SET protocol, on the ASW protocol, and on the H.530 protocol, which will be our running example throughout the paper.

The remainder of this paper is organized as follows. Section 2 is devoted to the description of the AVISPA Tool, presenting the architecture and the web-based graphical interface of the tool (§2.1), and discussing the main characteristics of the high-level and the low-level specification languages (§2.2) and of the four back-ends of the tool (§2.3). Section 3 discusses the results of the experiments that we have carried out on applying the tool for the validation of the protocols in the AVISPA Library. Section 4 concludes the paper by giving an outlook on possible future extensions of the AVISPA Tool.

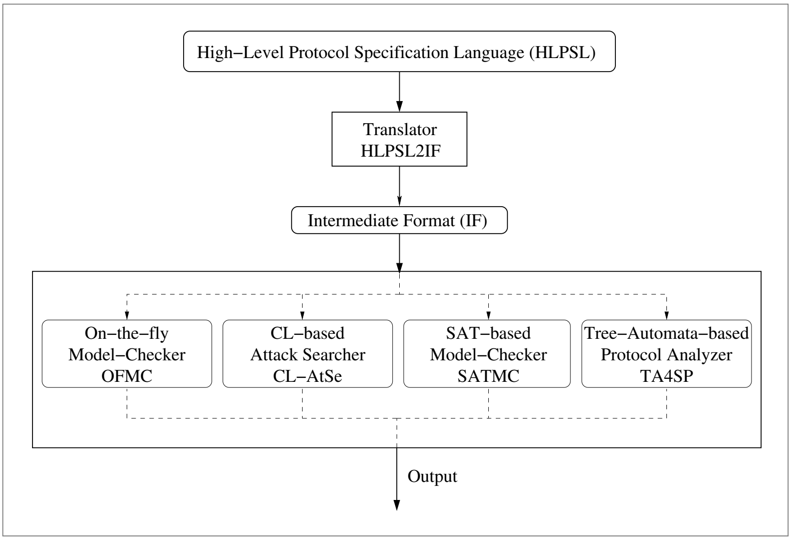


Fig. 1. The architecture of the AVISPA Tool.

2 The AVISPA Tool

2.1 The Architecture of the AVISPA Tool

The architecture of the AVISPA Tool is depicted in Fig. 1. A user interacts with the tool by specifying a *security problem* (a protocol paired with a security property that it is expected to achieve) in the High-Level Protocol Specification Language HLPSL. The HLPSL is an expressive, modular, role-based, formal language that allows for the specification of control-flow patterns, data structures, different cryptographic operators and their algebraic properties, alternative adversary models, as well as complex security properties. These features allow one to specify protocols in HLPSL without resorting to specific techniques to simplify the protocols first, as is often required in weaker approaches. The AVISPA Tool automatically translates (via the *HLPSL2IF Translator*) a user-defined security problem into an equivalent specification written in the rewrite-based formalism *Intermediate Format IF*. An IF specification describes an infinite-state transition system amenable to formal analysis: IF specifications are automatically input to the back-ends of the AVISPA Tool, which implement different techniques to search the corresponding infinite-state transition system for states that represent attacks on the intended properties of the protocols.

The current version of the tool integrates four back-ends: the *On-the-fly*

sor tool, the AVISS Tool [3], which automated the analysis of security protocols like those in the Clark/Jacob library.

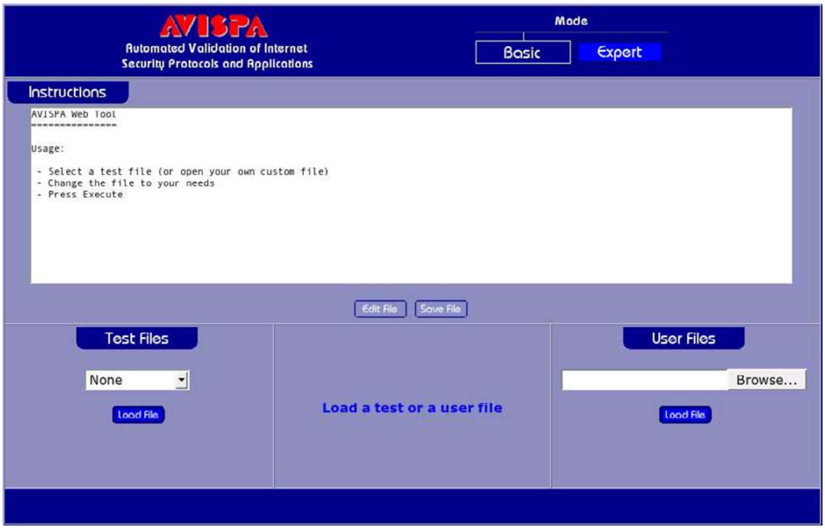
Model-Checker OFMC, the *Constraint-Logic-based Attack Searcher CL-AtSe*, the *SAT-based Model-Checker SATMC*, and the *TA4SP protocol analyzer*, which verifies protocols by implementing tree automata based on automatic approximations. All the back-ends of the tool analyze protocols under the assumptions of *perfect cryptography* and that the protocol messages are exchanged over a network that is under the control of a *Dolev-Yao intruder* [44]. That is, the back-ends analyze protocols by considering the standard protocol-independent, asynchronous model of an active intruder who controls the network but cannot break cryptography; in particular, the intruder can intercept messages and analyze them if he possesses the corresponding keys for decryption, and he can generate messages from his knowledge and send them under any party name.

Upon termination, each back-end of the AVISPA Tool outputs the result of its analysis using a common and precisely defined *output format* stating whether the input problem was solved (giving a description of the considered protocol goal or, in case it was violated, the related attack trace), some of the system resources were exhausted, or the problem was not tackled by the required back-end for some reason.

The AVISPA Tool has been designed to be easily usable by IT professionals, engineers, and protocol designers working in industry or standardization organizations. We thus deployed the AVISPA Tool as a downloadable single "package" to be installed on the users' local machines, as well as a remote tool that can be employed by external users thanks to a web-based graphical user interface that supports the editing of protocol specifications and allows the user to select, configure, and execute the different back-ends of the tool.[4]

Using the interface, the user can easily load a protocol specification among the ones provided in the AVISPA Library, or write a specification on his own and invoke one or all of the back-ends. An XEmacs mode for editing protocol specifications is available as well. In case an attack is found, the attack trace is output in ASCII as well as in a graphical format, using Message Sequence Charts, which can be displayed in a new window or output to a postscript file. The interface features specialized menus for both novice users (*basic mode*) and expert users (*expert mode*), as displayed in Fig. 2 and Fig. 3, respectively. In particular, Fig. 3 shows part of the specification of the H.530 protocol (in the main window of the interface and in an XEmacs window) and the message sequence chart of the attack trace that the AVISPA Tool found when analyzing the protocol (in the bottom window), to which we will return below.

[4] Both the package and the web-based graphical user interface are available at the project's web-site [10].



```
AVISPA Web Tool
===============

Usage:

 - Select a test file (or open your own custom file)
 - Change the file to your needs
 - Press Execute
```

Fig. 2. A screen-shot of the AVISPA Tool in basic mode.

2.2 The Specification Languages HLPSL and IF

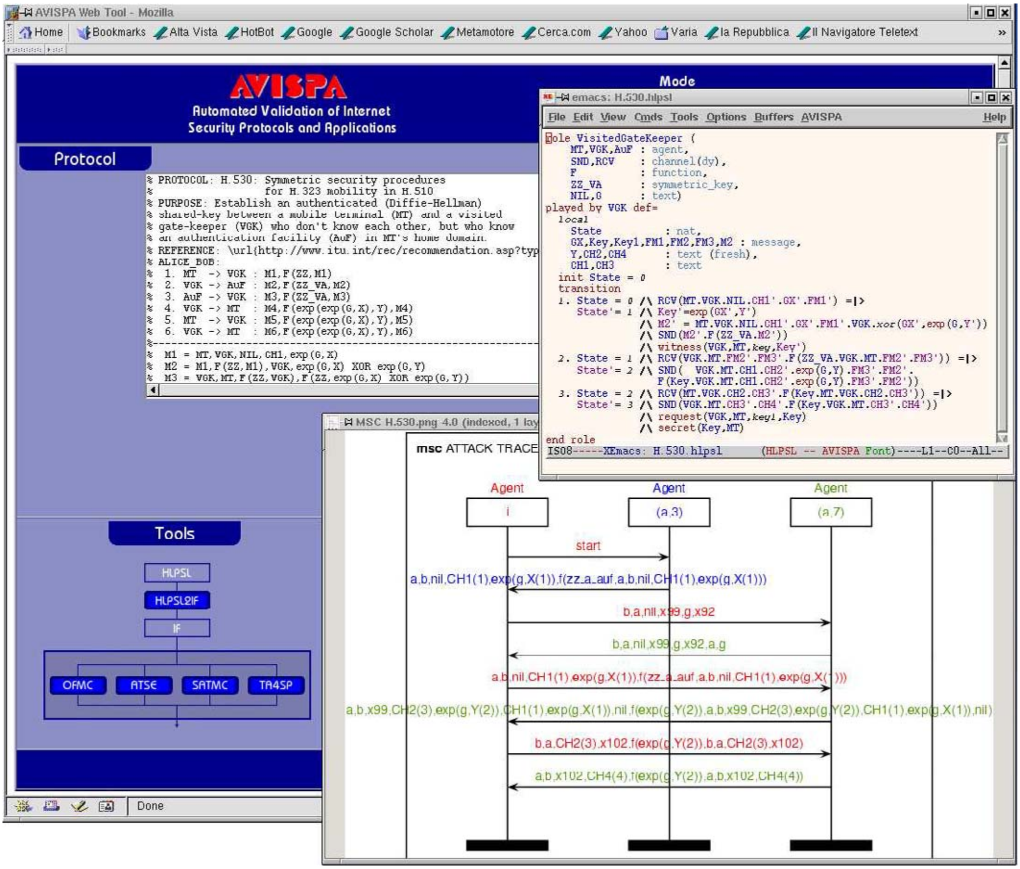
2.2.1 The HLPSL

The so-called *Alice&Bob notation* is commonly used to describe security protocols as sequences of message exchange steps of the form

$$A \rightarrow B \;:\; M\,.$$

This notation is quite convenient as it gives an illustration of the messages exchanged in a normal, successful run of a given protocol, and several protocol specification languages (e.g., [23,55,64,73] as well as an older version of HLPSL [3]) are based on (a formalization of the) Alice&Bob notation. However, this notation, while intuitive and compact, is also informal and not expressive enough to capture the sequence of events that need to be specified when considering large-scale Internet protocols. For instance, such protocols often call for control-flow constructs such as if-then-else branches, looping and other features. The Alice&Bob notation, which shows only message exchanges, is too high level to capture such constructs, which require defining the sequence of actions taken by each honest principal participating in a protocol run. That's why we need a more expressive language like HLPSL, which, as we remarked above, enjoys a number of features that make it well suited for specifying modern, industrial-scale protocols.

In order to discuss the main features of the HLPSL and of the AVISPA Tool, let us consider, as a running example, the H.530 protocol of the ITU [53], which has been developed by Siemens in order to provide mutual authentica-


AVISPA Web Tool - Mozilla
Home
Bookmarks
Alta Vista
HotBot
Google
Google Scholar
Metamotore
Cerca.com
Yahoo
Varia
la Repubblica
Il Navigatore Teletext
AVISPA
Automated Validation of Internet
Security Protocols and Applications
Protocol
Mode
Tools
HLPSL
HLPSL2IF
IF
OFMC
ATSE
SATMC
TA4SP
Done
emacs: H.530.hlpsl
File
Edit
View
Cmds
Tools
Options
Buffers
AVISPA
Help
MSC H.530.png 4.0 (indexed, 1 lay
msc ATTACK TRACE
Agent
i
Agent
(a,3)
Agent
(a,7)
start
a,b,nil,CH1(1),exp(g,X(1)),f(zz_a_auf,a,b,nil,CH1(1),exp(g,X(1)))
b,a,nil,x99,g,x92
b,a,nil,x99,g,x92,a,g
a,b,nil,CH1(1),exp(g,X(1)),f(zz_a_auf,a,b,nil,CH1(1),exp(g,X(1)))
a,b,x99,CH2(3),exp(g,Y(2)),CH1(1),exp(g,X(1)),nil,f(exp(g,Y(2)),a,b,x99,CH2(3),exp(g,Y(2)),CH1(1),exp(g,X(1)),nil)
b,a,CH2(3),x102,f(exp(g,Y(2)),b,a,CH2(3),x102)
a,b,x102,CH4(4),f(exp(g,Y(2)),a,b,x102,CH4(4))

```
% PROTOCOL: H.530: Symmetric security procedures
%                  for H.323 mobility in H.510
% PURPOSE: Establish an authenticated (Diffie-Hellman)
% shared-key between a mobile terminal (MT) and a visited
% gate-keeper (VGK) who don't know each other, but who know
% an authentication facility (AuF) in MT's home domain.
% REFERENCE: \url{http://www.itu.int/rec/recommendation.asp?typ
% ALICE_BOB:
%  1. MT  -> VGK : M1,F(ZZ,M1)
%  2. VGK -> AuF : M2,F(ZZ_VA,M2)
%  3. AuF -> VGK : M3,F(ZZ_VA,M3)
%  4. VGK -> MT  : M4,F(exp(exp(G,X),Y),M4)
%  5. MT  -> VGK : M5,F(exp(exp(G,X),Y),M5)
%  6. VGK -> MT  : M6,F(exp(exp(G,X),Y),M6)
%------------------------------------------------------------
%  M1 = MT,VGK,NIL,CH1,exp(G,X)
%  M2 = M1,F(ZZ,M1),VGK,exp(G,X) XOR exp(G,Y)
%  M3 = VGK,MT,F(ZZ,VGK),F(ZZ,exp(G,X) XOR exp(G,Y))
```

```
role VisitedGateKeeper (
    MT,VGK,AuF  : agent,
    SND,RCV     : channel(dy),
    F           : function,
    ZZ_VA       : symmetric_key,
    NIL,G       : text)
played by VGK def=
  local
    State             : nat,
    GX,Key,Key1,FM1,FM2,FM3,M2 : message,
    Y,CH2,CH4         : text (fresh),
    CH1,CH3           : text
  init State = 0
  transition
  1. State = 0 /\ RCV(MT.VGK.NIL.CH1'.GX'.FM1') =|>
     State'= 1 /\ Key'=exp(GX',Y')
               /\ M2' = MT.VGK.NIL.CH1'.GX'.FM1'.VGK.xor(GX',exp(G,Y'))
               /\ SND(M2'.F(ZZ_VA.M2'))
               /\ witness(VGK,MT,key,Key')
  2. State = 1 /\ RCV(VGK.MT.FM2'.FM3'.F(ZZ_VA.VGK.MT.FM2'.FM3')) =|>
     State'= 2 /\ SND(  VGK.MT.CH1.CH2'.exp(G,Y).FM3'.FM2'.
                 F(Key.VGK.MT.CH1.CH2'.exp(G,Y).FM3'.FM2'))
  3. State = 2 /\ RCV(MT.VGK.CH2.CH3'.F(Key.MT.VGK.CH2.CH3')) =|>
     State'= 3 /\ SND(VGK.MT.CH3'.CH4'.F(Key.VGK.MT.CH3'.CH4'))
               /\ request(VGK,MT,key1,Key)
               /\ secret(Key,MT)
end role
ISO8-----XEmacs: H.530.hlpsl      (HLPSL -- AVISPA Font)----L1--C0--All--
```

Fig. 3. A screen-shot of the AVISPA Tool in expert mode.

tion and key agreement in mobile roaming scenarios in multimedia communication. The interested reader will find more details about the HLPSL in the paper [31], in the HLPSL Tutorial [11], and in the AVISPA User Manual [12], which are all available at the AVISPA web-site [10]. There, the reader will also find the AVISPA Library, which contains example specifications of Internet protocols that are even more complex than the H.530 protocol.

2.2.2 The H.530 Protocol in HLPSL

The H.530 protocol is deployed as shown in the left part of Fig. 4: a mobile terminal (MT) wants to establish a secure connection and negotiate a Diffie-Hellman key with the gatekeeper (VGK) of a visited domain. As they do not know each other in advance, the authentication is performed using an authentication facility AuF within the home domain of the MT. Both MT and VGK initially have shared keys with AuF. The right part of Fig. 4 shows the messages exchanged: first, both MT and VGK create Diffie-Hellman half-keys,

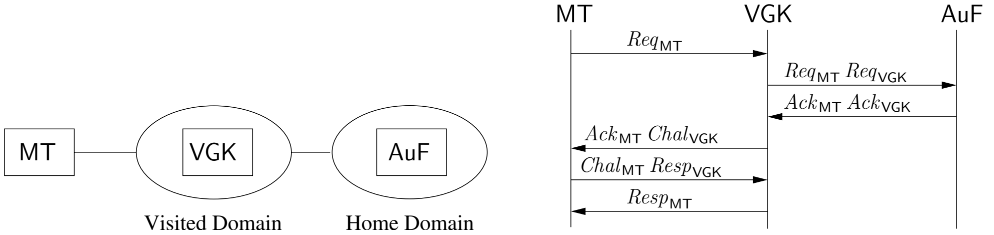


Fig. 4. The H.530 protocol (simplified). The deployment of the protocol is on the left, the messages exchanged between the participants are summarized on the right.

along with hashes that are encrypted for AuF (denoted by the messages Req_{MT} and Req_{VGK}, respectively). After checking these messages, AuF replies with appropriate acknowledgment messages Ack_{MT} and Ack_{VGK} that also contain encrypted hashes for the respective recipients. Finally, MT and VGK perform a mutual challenge-response using the new Diffie-Hellman key that was authenticated by AuF (directly or over a chain of trustworthy servers).

HLPSL is a role-based language, meaning that we first specify the sequence of actions of each kind of protocol participant in a module, which is called a *basic role*. This specification can later be instantiated by one or more agents playing the given role, and we further specify how the resulting participants interact with one another by "gluing" multiple basic roles together into a *composed role*. In the case of the H.530 protocol, for instance, there are three basic roles, which we call `mobileTerminal`, `visitedGateKeeper`, and `authenticationFacilityserver`. Note that role names begin with lowercase letters. We use, for instance, the name `mobileTerminal` to denote the role itself, while the name of the agent playing the role will be called `MT`, as in Fig. 4.

Each basic role describes what information the participant can use initially (*parameters*), its *initial state*, and ways in which the state can change (*transitions*). Fig. 4 already displayed part of the HLPSL-specification of the `visitedGateKeeper` role (as well as part of the message exchanges of the protocol in the Alice&Bob notation), and the following is the initial part of the role `mobileTerminal`:

```
role mobileTerminal (MT,VGK,AuF : agent,
                     SND,RCV    : channel(dy),
                     F          : hash_func,
                     ZZ         : symmetric_key,
                     NIL,G      : text)
played_by MT def=
  local  State             : nat,
         X,CH1,CH2,CH3,CH4 : text,
          GY,Key              : message
   init  State := 0
   transition
       ...
 end role
```

The role has a number of parameters. MT, VGK, and AuF are of type agent, F is of type hash function, ZZ is of type symmetric_key, and NIL and G are of type text and will represent nonces created in the course of the protocol execution. The SND and RCV parameters are of type channel, indicating that these are channels through which the agent playing role mobileTerminal will communicate. The attribute to the channel type, in this case (dy) for Dolev-Yao, denotes the intruder model to be considered for this channel (different intruder models and their specification in HLPSL, as well as other details such as the different types available in HLPSL, are discussed in [12], which describes the full syntax and semantics of the language). All variables in HLPSL begin with a capital letter, and all constants begin with a lower-case letter. Moreover, all variables and constants are typed.

The parameter MT appears in the played_by section, which means, intuitively, that MT denotes the name of the agent who plays the mobileTerminal role. The local section declares local variables of mobileTerminal, such as the local variable called State, which is a natural number and is initialized to 0 in the init section.

The transition section of a HLPSL specification contains a set of transitions. Generally, each transition represents the receipt of a message and the sending of a reply message. A transition consists of a trigger, or precondition, and an action to be performed when the triggering event occurs. For instance, the role mobileTerminal in our running example contains the following transition (where some details have been omitted for simplicity):

```
 step2.
   State  = 1 /\ RCV(VGK.MT.CH1.CH2'.GY'.
                      F(ZZ.xor(exp(G,X),GY')).F(ZZ.VGK).
                      F(exp(GY',X).VGK.MT.CH1.CH2'.GY'.
                        F(ZZ.xor(exp(G,X),GY')).F(ZZ.VGK)))
   =|>
   State':= 2 /\ CH3' := new()
              /\ Key' := exp(GY',X)
              /\ SND(MT.VGK.CH2'.CH3'.F(Key'.MT.VGK.CH2'.CH3'))
```

This is a transition called step2, but note that the names of the transitions serve merely to distinguish them from one another. It specifies that if the value of `State` is equal to `1` and the message received on channel `RCV` has the expected structure shown above, then a transition fires which, among other things, sets the new value of `State` to 2 and generates and sends the key `Key'` (hashed and paired together with other data) on channel `SND`. Here we see an example of *priming*; `X'` means: the new value of the variable `X`. The notation stems from the *Temporal Logic of Action TLA* [59,60], upon which HLPSL is based. It is important to realize that the value of the variable will not be changed until the current transition is complete. So, the right-hand side tells us that the value of the `State` variable, after transition `step2` fires, will be `2`. The variables that occur within the `RCV` behave as follows: a variable that is not primed indicates that the received message must have the same value as the current value of the variable, while primed variables are bound in the next step to whatever is received. This is how one may model the way in which the information available to a role may change.

The other transitions of the `mobileTerminal` role and those of the other two basic roles are specified similarly, and the basic roles are then composed together in a composed role. Composed roles have no `transition` section, but rather they instantiate one or more basic roles, "gluing" them together so they execute together, usually in parallel (with an interleaving semantics) by means of the operator `/\`. That is, composed roles describe sessions of the protocol. In our example, we can define the following composed role `session` which instantiates one instance of each basic role and thus describes one whole protocol session:

```
role session(
  MT,VGK,AuF : agent,
  F          : hash_func,
  ZZ,ZZ_VA   : symmetric_key,
  NIL,G      : text)
def=
  local SND,RCV : channel (dy)
  composition
        mobileTerminal(MT,VGK,AuF,SND,RCV,F,ZZ,NIL,G)
     /\ authenticationFacility(MT,VGK,AuF,SND,RCV,F,ZZ,
                                ZZ_VA,NIL,G)
     /\ visitedGateKeeper(MT,VGK,AuF,SND,RCV,F,ZZ_VA,NIL,G)
end role
```

The last role to be defined in an HLPSL protocol specification is a top-level role that contains global constants and a composition of one or more sessions, where the intruder may play some roles as a legitimate user. There is also a statement which describes the initial knowledge of the intruder. Typically, this includes the names of all agents, all public keys, the intruder's own private key, any keys he shares with others, and all publicly-known functions. Note that the constant `i` is used to refer to the intruder. In our example, we could include the following in our HLPSL specification:

```
role environment()
def=
  const
    a,b,auf                     : agent,
    f                           : hash_func,
    sec_m_Key,key,key1          : protocol_id,
    zz_a_auf,zz_b_auf,zz_i_auf  : symmetric_key,
    nil,g                       : text
  intruder_knowledge = {a,b,auf,f,g,nil,zz_i_auf}
  composition
       session(a,b,auf,f,zz_a_auf,zz_b_auf,nil,g)
    /\ session(a,b,auf,f,zz_a_auf,zz_b_auf,nil,g)
end role
```

The sessions specified allow the AVISPA Tool to perform bounded session verification, but note that the associated search space is still infinite as the messages are not bounded. As we discuss in [17], we also have a preliminary implementation of a technique that we call *symbolic session generation*, which can be used to exploit the symbolic representation of the intruder provided by OFMC and CL-AtSe, and thereby avoid enumerating all possible session instances associated with a bounded number of sessions. The TA4SP back-end, which performs unbounded protocol verification, simply ignores the session declarations in the specification.

Finally, we declare the goal(s) of the protocol, which in the case of the H.530 protocol are:

```
goal
  authentication_on key
  authentication_on key1
  secrecy_of sec_m_Key, sec_v_Key
end goal
```

Security goals are specified in HLPSL by augmenting the transitions of basic roles with so-called *goal facts* and by then assigning them a meaning by describing, in the HLPSL `goal` section, what conditions — that is, what combination of such facts — indicate an attack. In other words, we model the goals of the protocol by labeling several transitions in the HLPSL specification with special events that express the meaning of the transition with respect to the protocol goals.

For instance, for *secrecy*, the goal facts assert which values should be secret between whom, and the goal declaration in the `goal` section (e.g., `secrecy_of sec_m_Key, sec_v_Key`) specifies that if the intruder learns a secret value that is not explicitly a secret between him and someone else, then the intruder has successfully attacked the protocol. Similarly, HLPSL provides for the specification of goal facts related to *authentication* (e.g., `authentication_on key` and `authentication_on key1`), which are for instance used to check that a principal is right in believing that his intended peer is present in the current session, has reached a certain state, and agrees on a certain value, which typically is fresh.

Internally, the attack conditions are specified in terms of temporal logic (as safety properties), but macros are provided for the most frequently used security goals, i.e., secrecy and different forms of authentication (cf. the notions of authentication discussed in [63]).

2.2.3 The IF

The HLPSL enjoys both a declarative semantics based on a fragment of the TLA [59,60] and an operational semantics based on the translation into the rewrite-based formalism Intermediate Format IF, which is performed automatically by the HLPSL2IF translator.[5] An IF specification describes a protocol in terms of rewrite rules describing an infinite-state transition system with an initial state, transition rules, and a state-based safety property, namely a goal (attack) predicate that defines whether a given state is an attack state or not. (An *attack trace* is then a path that leads from the initial state to an attack state.) IF specifications can be generated both in an untyped variant and in a typed one, which abstracts away type-flaw attacks (if any) from the protocol; this is useful as in many cases type-flaws can be prevented in the actual implementation of a protocol [51].

The Intermediate Format thus provides low-level descriptions of protocols and their properties that are suitable for automatic analysis (rather than being abstract and fairly easy to read for human users like the HLPSL), and yet this format is independent from the analysis methods employed by the various back-ends of the tool. This decoupling allows for the independent development of the high-level language and of the back-ends, and also provides an interface for the future connection of other tools to the AVISPA Tool.[6]

[5] Similar translations from a high-level language into a low-level one have been developed for CAPSL/CIL [29] and for CASRUL [55], as well as from an older version of HLPSL based on Alice&Bob notation to IF [3], and from Alice&Bob-style protocol notation to a process algebra that is similar to the spi-calculus [23].

[6] A first example of such a connection is the work of Gotsman, Massacci, and Pis-

2.3 The Back-Ends of the AVISPA Tool

The **Constraint-Logic-based Attack Searcher (CL-AtSe)** applies constraint solving to perform both protocol falsification and verification for bounded numbers of sessions [30,32,33,34,35,36,37,38,55,72,77]. The protocol messages can be typed or untyped, and the pairing can be considered to be associative or not. Several properties of the XOR operator can be handled too, and, more generally, CL-AtSe is built in a modular way and is thus open to extensions for handling algebraic properties of cryptographic operators. CL-AtSe performs several kinds of optimizations to reduce, and often eliminate, redundancies or useless branches in the protocol symbolic execution. For instance, the *lazy intruder technique* represents terms symbolically to avoid explicitly enumerating the possible messages the Dolev-Yao intruder can generate, and thus significantly reduces the search space without excluding any attacks nor introducing new ones. This is achieved by representing intruder messages using terms with variables, and storing and manipulating constraints about what terms must be generated and which terms may be used to generate them.

The **On-the-fly Model-Checker (OFMC)** [13,14,15,16,17,49,50] performs both protocol falsification and bounded session verification, by exploring the transition system described by an IF specification in a demand-driven way (i.e., on-the-fly, hence its name). OFMC considers both typed and untyped protocol models. OFMC's effectiveness is due to the integration of a number of symbolic, constraint-based techniques, which are correct and complete, in the sense that no attacks are lost nor new ones are introduced by them. One such technique is the lazy intruder technique. As another significant example, the *constraint differentiation technique* is a general search reduction technique that integrates the lazy intruder with ideas from partial-order reduction, and which can be formally proved to terminate and to be correct and complete, thereby reducing search time by a factor of two to several orders of magnitude. Moreover, OFMC also implements a number of efficient search heuristics. It also provides support for the modeling an intruder who is capable of performing guessing attacks on weak passwords, and for the specification of algebraic properties of cryptographic operators.

The **SAT-based Model-Checker (SATMC)** [4,5,6,7,8,40] considers the typed protocol model and performs both protocol falsification and bounded session verification by reducing the input problem to a sequence of invoca-

tore [48], who have formalized a translation procedure from protocol descriptions in HLPSL to descriptions in the applied pi calculus, which allowed them to apply the ProVerif tool [1,2,21,22] to some of our HLPSL protocol specifications. It will be interesting to compare similar translations from IF into the applied pi calculus or other protocol specification formalisms.

tions to state-of-the-art SAT solvers. More specifically, SATMC first builds a propositional formula encoding a bounded unrolling of the transition relation specified by the IF, the initial state, and the set of states representing a violation of the security properties. The propositional formula is then fed to a SAT solver and any model found is translated back into an attack. The interface between the SATMC and the SAT solver complies with the DIMACS format (a de facto standard for SAT problems) and therefore SATMC can easily incorporate and exploit new SAT solvers as soon as they become available.

The **TA4SP (Tree Automata based on Automatic Approximations for the Analysis of Security Protocols)** back-end [24,25,70] performs unbounded protocol verification by approximating the intruder knowledge by using regular tree languages and rewriting. Its starting point is an extension of an approximation method based on tree automata, introduced by Genet and Klay [46,47] for verifying security protocols. The previous procedures of this kind required the presence of an expert to transform by hand a security protocol into a term-rewriting system and compute an ad hoc approximation function. Our result allows one to compute an approximation function automatically. For secrecy properties in the typed model, TA4SP can show whether a protocol is flawed (by under-approximation) or whether it is safe for any number of sessions (by over-approximation).

3 Experimental Results

We have assessed the AVISPA Tool by running it against the AVISPA Library, which at present comprises HLPSL-specifications of 215 security problems derived from 33 protocols (or 147 problems if all the secrecy properties specified for a protocol are checked in one single security problem). The tool successfully analyzes all problems in the library in less than 24 minutes of CPU time per problem (globally, the entire library of 215 problems requires 87 minutes of CPU time to be analyzed). In more detail, the majority of the problems (namely, 206 problems) require less than 1 second of CPU time each, and 211 problems require less than 10 seconds of CPU time each to be analyzed. Hence, the time required by the AVISPA Tool for analyzing most of the problems is very low and thus acceptable for a modeler involved in security protocol design. Note that the time spent by the AVISPA Tool for compiling HLPSL into IF is always negligible (a few milliseconds), and therefore we do not report it in the tables of results.

To the best of our knowledge, no other tool exhibits the same scope and robustness while enjoying the same performance and scalability. Moreover, the AVISPA Tool has detected a number of previously unknown attacks on some of the protocols analyzed, e.g., on some protocols of the ISO-PK family, on the IKEv2 protocol with digital signatures, on the SET protocol, on the ASW protocol, and on the H.530 protocol.

We will discuss these new attacks found by the AVISPA Tool in §3.2, after a detailed presentation of the results of the experimental evaluation of the effectiveness and the performance of the tool in §3.1.

3.1 Effectiveness and Performance of the Tool

The protocol specifications in the IF are equipped with a signature section describing the types of the messages exchanged among the participating agents. This section may be neglected by the back-ends in order to search for type-flaw attacks; when this is the case, we say that the back-end considers the *untyped model* of the security problem. If the signature section is taken into account, then type-flaw attacks are abstracted away from the analysis, and we say that the back-end considers the *typed model* of the security problem.

It is fundamental that both models are considered during analysis as, on the one hand, it is important to be able to detect all possible attacks, but on the other hand, many type-flaw attacks are of little practical significance as actual implementations of security protocols often enforce simple mechanisms that exclude their applicability (see, for instance, [51]). All the four back-ends are able to carry out the analysis with respect to the typed model, and CL-AtSe and OFMC are also able to adopt the untyped model. The AVISPA Tool can thus analyze protocols with respect to both models.

We have run the AVISPA Tool against three classes of problems modeling a typed scenario with a bounded number of protocol sessions (denoted by TY&B), an untyped scenario with a bounded number of protocol sessions (denoted by UNTY&B), and a typed scenario with an unbounded number of protocol sessions (denoted by TY&UNB, but note that experimentation with TY&UNB has started only recently and so the results in this case are preliminary).

By running the back-ends of the AVISPA Tool against all the 215 problems under the TY&B and UNTY&B scenarios, we obtained the results summarized in Table 1 and Table 2, respectively.[7] For each of the protocols, the tables give the number of security problems ("#P"), and for each back-end, the number of problems for which no attacks are detected ("S"), the number of problems for which attacks are detected ("A"), and the (average) time in seconds ("Time") spent by the back-end to find the attacks or to report that no attack exists

[7] Results are obtained by each single back-end with a resource limit of 1 hour CPU time and 1GB memory, on a Pentium IV 2.4GHz under Linux.

Table 1
Effectiveness of the AVISPA Tool on the TY&B scenario.

Problems		**CL-AtSe**			**OFMC**			**SATMC**			**TA4SP**		
Protocol	**#P**	**Time**	**S**	**A**	**Time**	**S**	**A**	**Time**	**S**	**A**	**Time**	**S**	**A**
UMTS_AKA	4	0.01	4	0	0.03	4	0	0.01	4	0	0.56	2	0
ISO1	1	0.02	0	1	0.02	0	1	0.04	0	1	-	0	0
ISO2	1	0.02	1	0	0.07	1	0	0.63	1	0	-	0	0
ISO3	2	0.03	0	$\boxed{2}$	0.03	0	$\boxed{2}$	0.39	0	$\boxed{2}$	-	0	0
ISO4	2	0.03	2	0	0.38	2	0	208.31	2	0	-	0	0
CHAPv2	4	0.02	4	0	0.18	4	0	0.10	4	0	16.29	2	0
EKE	4	0.03	2	2	0.10	2	2	0.09	2	2	2.86	2	0
SRP	4	0.02	4	0	0.07	4	0	-	0	0	-	0	0
EKE2	4	0.03	4	0	0.05	4	0	-	0	0	-	0	0
SPEKE	6	0.07	6	0	1.49	6	0	-	0	0	-	0	0
IKEv2-CHILD	4	0.07	4	0	0.51	4	0	-	0	0	-	0	0
IKEv2-DS	4	0.29	3	$\boxed{1}$	2.38	3	$\boxed{1}$	-	0	0	-	0	0
IKEv2-DSx	4	3.74	4	0	17.28	4	0	-	0	0	-	0	0
IKEv2-MAC	4	0.05	4	0	3.01	4	0	-	0	0	-	0	0
IKEv2-MACx	4	5.27	4	0	15.94	4	0	-	0	0	-	0	0
TLS	4	0.05	4	0	0.29	4	0	1018.28	4	0	TO	0	0
LPD-MSR	2	0.02	0	2	0.02	0	2	0.06	0	2	0.61	0	0
LPD-IMSR	2	0.04	2	0	0.04	2	0	0.10	2	0	3.25	1	0
Kerb-basic	10	0.07	10	0	0.61	10	0	6.24	10	0	TO	0	0
Kerb-Cross-Realm	18	0.52	18	0	2.22	18	0	5.70	18	0	-	0	0
Kerb-Ticket-Cache	11	0.08	11	0	0.60	11	0	28.90	11	0	-	0	0
Kerb-PKINIT	12	0.06	12	0	0.47	12	0	27.21	12	0	-	0	0
Kerb-Forwardable	12	0.16	12	0	7.12	12	0	TO	0	0	-	0	0
Kerb-preauth	12	0.12	12	0	0.39	12	0	20.54	12	0	-	0	0
CRAM-MD5	2	0.04	2	0	0.23	2	0	0.17	2	0	0.97	1	0
PBK	1	0.01	0	1	0.34	0	1	0.25	0	1	-	0	0
PBK-fix	1	0.03	0	1	0.14	0	1	0.09	0	1	-	0	0
PBK-fix-weak-auth	1	0.49	1	0	3.47	1	0	0.33	1	0	-	0	0
hip	2	0.09	2	0	0.23	2	0	-	0	0	-	0	0
DHCP-delayed-auth	2	0.02	2	0	0.06	2	0	0.12	2	0	6.84	1	0
lipkey-spkm-knw-init.	6	0.06	6	0	0.17	6	0	-	0	0	-	0	0
lipkey-spkm-unknw-init.	5	0.12	5	0	4.72	5	0	-	0	0	-	0	0
TSIG	2	0.05	2	0	0.19	2	0	0.38	2	0	-	0	0
ASW	3	0.10	3	0	0.35	3	0	TO	0	0	-	0	0
ASW-abort	4	0.20	3	$\boxed{1}$	1.98	3	$\boxed{1}$	65.75	3	$\boxed{1}$	-	0	0
FairZG	5	0.37	5	0	8.76	5	0	0.28	5	0	-	0	0
SET-purchase	4	23.66	1	$\boxed{2}$	1.24	0	$\boxed{2}$	TO	0	0	-	0	0
SET-p.-hon.-payment-gw	4	0.61	4	0	0.83	4	0	TO	0	0	-	0	0
AAAMobileIP	9	0.03	9	0	0.14	9	0	0.11	9	0	754.11	3	0
h.530	4	TO	0	0	0.70	0	$\boxed{2}$	-	0	0	-	0	0
h.530-fix	4	TO	0	0	1391.66	4	0	-	0	0	-	0	0
Simple	3	100.17	3	0	78.50	3	0	0.50	3	0	-	0	0
CTP-non_predictive-fix	3	0.06	3	0	0.23	3	0	TO	0	0	-	0	0
geopriv	5	0.04	5	0	0.25	5	0	0.08	5	0	-	0	0
pervasive	2	47.67	2	0	30.52	2	0	4.00	2	0	TO	0	0
two_pseudonyms	5	0.06	5	0	0.30	5	0	0.07	5	0	-	0	0
QoS-NSLP	2	32.16	2	0	16.01	2	0	0.21	2	0	-	0	0
sip	1	0.05	1	0	1.86	1	0	810.01	1	0	-	0	0

Legend:

-	:	the problem is not supported by the back-end
TO	:	time-out

in the given (bounded) scenario. For SATMC, we report only the time spent to generate the SAT formula since that spent to solve the formula is always negligible. A "TO" in the Time column indicates that a "time-out" occurred (after 1 hour of CPU time), while "–" indicates that the back-end does not support some of the features required by the problem (in most cases, this regards special properties of cryptographic operators such as exponentiation) and hence that the problem cannot be properly analyzed by the back-end. A boxed number in the "A" column denotes that the AVISPA Tool has found at least one new (previously unknown in literature) attack.

When using the untyped model (see Table 2), CL-AtSe uses the associativity property of pairing, while OFMC does not. This explains why CL-AtSe finds more attacks on some protocols than OFMC. It must be said that the majority of these attacks are not of practical significance, since they can be straightforwardly prevented in actual implementations (in fact, the length of each message field is usually known in advance and can be simply checked).

Table 3 shows the results obtained by running the TA4SP back-end of the AVISPA Tool under the TY&UNB scenario. For each problem, we report whether the absence of any attack has been established in the considered unbounded scenario (a "YES" in the column "Safe") and the time in seconds spent by the TA4SP back-end to analyze the problem (column "TA4SP").

3.2 New Attacks

We briefly discuss some of the new attacks that the AVISPA Tool has found.

The ISO-PK3 protocol

The AVISPA Tool has found a new attack on the ISO-PK3 (also known as "ISO Public Key Two-Pass Mutual Authentication") protocol [52]. It was already known that ISO-PK3 is vulnerable to replay attacks and hence does not provide strong authentication [45]: nothing in the messages ensures the freshness of the messages for the responder role. The analysis with the AVISPA Tool showed that the ISO-PK3 protocol does not even guarantee weak authentication, i.e., after successfully executing the protocol, neither the initiator nor the responder can be sure about the authenticity of the exchanged messages.

The IKEv2 protocol with digital signatures

A man-in-the-middle attack discovered on the IKEv2 protocol with digital signatures (IKEv2-DS [56]) is new[8] although it is similar to a well-known attack on the Station-2-Station protocol [62]. As pointed out in [67], several

[8] Notice that, independently, the same attack has been reported in [65].

Table 2
Effectiveness of the AVISPA Tool on the UNTY&B scenario.

Problems		**CL-AtSe**			**OFMC**		
Protocol	**#P**	**Time**	**S**	**A**	**Time**	**S**	**A**
UMTS_AKA	4	0.02	4	0	0.04	4	0
ISO1	1	0.01	0	1	0.02	0	1
ISO2	1	0.01	0	1	0.07	1	0
ISO3	2	0.02	0	$\boxed{2}$	0.03	0	$\boxed{2}$
ISO4	2	0.04	0	2	0.63	2	0
CHAPv2	4	0.02	4	0	0.27	4	0
EKE	4	0.04	2	2	0.10	2	2
SRP	4	0.03	4	0	0.08	4	0
EKE2	4	0.02	4	0	0.04	4	0
SPEKE	6	0.08	6	0	1.52	6	0
IKEv2-CHILD	4	0.07	4	0	0.43	4	0
IKEv2-DS	4	0.11	0	$\boxed{4}$	2.52	3	$\boxed{1}$
IKEv2-DSx	4	1.19	0	4	23.63	4	0
IKEv2-MAC	4	0.15	2	2	3.25	4	0
IKEv2-MACx	4	4.82	2	2	22.14	4	0
TLS	4	0.07	4	0	0.27	4	0
LPD-MSR	2	0.02	0	2	0.03	0	2
LPD-IMSR	2	0.03	2	0	0.05	2	0
Kerb-basic	10	0.30	8	2	0.60	10	0
Kerb-Cross-Realm	18	7.75	15	3	1.93	18	0
Kerb-Ticket-Cache	11	0.18	0	6	0.53	11	0
Kerb-Forwardable	12	0.97	0	5	9.74	12	0
Kerb-PreAuth	12	0.20	0	6	0.54	12	0
Kerb-PKINIT	12	0.17	11	1	0.39	12	0
CRAM-MD5	2	0.07	2	0	0.83	2	0
PBK	1	0.01	0	1	0.35	0	1
PBK-fix	1	0.03	0	1	0.12	0	1
PBK-fix-weak-auth	1	0.50	1	0	4.26	1	0
hip	2	0.19	2	0	0.63	2	0
DHCP-delayed-auth	2	0.02	2	0	0.07	2	0
lipkey-spkm-knw-init.	6	0.06	6	0	0.14	6	0
lipkey-spkm-unknw-init.	5	0.29	5	0	3.78	5	0
TSIG	2	0.04	2	0	0.17	2	0
ASW	3	1.10	3	0	0.44	3	0
ASW-abort	4	7.76	3	1	4.92	3	1
FairZG	5	0.34	5	0	7.97	5	0
SET-purchase	4	91.17	0	$\boxed{3}$	1.45	0	$\boxed{2}$
SET-p.-hon.-payment-gw	4	TO	0	0	0.94	4	0
AAAMobileIP	9	0.03	7	2	0.13	7	2
h.530	4	93.43	0	2	0.62	0	2
h.530-fix	4	TO	0	0	1291.39	4	0
Simple	3	102.28	3	0	84.24	3	0
CTP-non_predictive-fix	3	0.05	1	2	0.21	3	0
geopriv	5	0.05	4	1	0.29	5	0
pervasive	2	92.19	2	0	67.25	2	0
two_pseudonyms	5	4.07	5	0	0.39	5	0
QoS-NSLP	2	60.55	2	0	48.59	2	0
sip	1	0.07	1	0	4.05	1	0

Legend:

TO : time-out

Table 3
Effectiveness of the AVISPA Tool on the TY&UNB scenario.

Problem	**Safe**	**TA4SP**
UMTS_AKA-secrecy-sseq1	YES	2.16
UMTS_AKA-secrecy-sseq2	YES	2.13
CHAPv2-secrecy-sec_kab1	YES	113.74
CHAPv2-secrecy-sec_kab2	YES	113.70
EKE-secrecy-sec_k1	YES	4.18
EKE-secrecy-sec_k2	YES	4.15
TLS-secrecy-sec_clientk		TO
TLS-secrecy-sec_serverk		TO
LPD-IMSR-secrecy-secx	YES	3.25
CRAM-MD5-secrecy-sec_SK	YES	0.37
DHCP-delayed-auth-secrecy-sec_k	YES	11.72
AAAMobileIP-secrecy-secFAHA		TO
AAAMobileIP-secrecy-secFAMN		TO
AAAMobileIP-secrecy-secMNHA		TO

Legend:

YES	:	the protocol is proved to be secure with respect to secrecy
TO	:	time-out

protocols that were inspired by Station-2-Station (e.g., also the first version of IKE) exhibit the same vulnerability. Also, as described in both [62] and [67], the attack has limited practical impact, since the intruder can confuse agents about whom they are talking to, but he cannot find out the key negotiated in such a run. We were able to formally express what it means that these attacks are "not relevant". More precisely, IKEv2 (and, similarly, the other similar protocols) does provide strong authentication when not viewing the key-negotiation in isolation but in relation with the usage of the key.

The SET protocol

The Secure Electronic Transactions (SET) protocol suite has been designed to allow for a secure e-commerce. The key feature is to hide the customer's credit card details from the merchant, and the customer's purchase details from the payment gateway. The AVISPA tool detects an attack where a dishonest payment gateway forwards payment authorization requests to another payment gateway. This is due to the fact that the part of the message signed by the card-holder (as well as the one signed by the merchant) does not contain the name of the desired payment gateway. This weakness of the protocol was already mentioned in the analysis of the SET protocol by Bella, Massacci, and Paulson using the interactive theorem prover Isabelle [18]. They argue that the attack is not very interesting as a dishonest payment gateway "has more interesting crimes to commit", however we believe that this vulnerability is not uncritical as it may lead to the situation that two payment gateways charge the account of the card-holder and both posses messages that seem to prove that the card-holder authorized the transaction. Like [18], we suggest to include the name of the desired payment gateway into the messages to fix this problem.

The ASW protocol

The ASW protocol [9] is an optimistic fair exchange protocol for contract signing intended to enable two parties to commit themselves to a previously agreed upon contractual text. A trusted third party (T3P) is involved *only* if dispute resolution is required (hence the term *optimistic*). In resolving disputes, the T3P issues either a *replacement contract* asserting that he recognizes the contract in question as valid, or an *abort token* asserting that he has never issued, and will never issue, a replacement contract. An important requirement of the protocol is that the intruder cannot block messages between an honest agent and the T3P forever.

The particular challenge in analyzing this protocol with the AVISPA Tool lies in the formulation of the goals of the protocol. In particular, we cannot directly formulate the main goal, *fair exchange*, which requires that if one party has a valid contract, then the other also has a valid contract or can obtain one from the T3P. This is a liveness property and we thus have approximated the goal by (stronger) safety properties.

The weakness the AVISPA tool has detected on this protocol, described in [49], is related to this approximation of the goal. A first, quite naïve, approximation of the goals implies that it already counts as an attack, if an intruder possesses both a valid contract and an abort token for that contract. This goal is easy to violate: the intruder as initiator can first run a normal exchange with an honest responder (not involving the T3P) and then ask the T3P for an abort. Moreover, after this, the intruder can start another exchange with the same contractual text and abort at any time; when the honest responder asks the T3P for a resolve, it will obtain an abort token. A more appropriate formulation of the goal thus allows the intruder to obtain both a valid contract and an abort token, as long as the other involved party of the contract also possesses a valid contract (with the same contractual text). This goal still implies the desired fair exchange property.

Still, the situation that one party has both a valid contract and an abort token was unexpected for us and it is unclear whether this situation was anticipated by the designers of the protocol. In fact, it is not unrealistic that an intruder can make another agent execute the protocol once more with the same contractual text by a kind of social engineering.[9] The weakness

[9] For instance, after the first exchange, the intruder could tell the contract partner that

can be eliminated by replay protection (logging all commitments used in any exchange and refusing to start a run with commitments that appear in the log). A similar weakness was discovered by [74] on another contract signing protocol, GJM, while for ASW this weakness was not reported previously in the literature. Moreover, as already shown in [75], ASW cannot provide strong authentication, and the AVISPA tool can also detect such attacks. However, these attacks against strong authentication are not very serious since one should assume that the contracts have some kind of unique identifier, e.g., in bank transactions a unique transaction number, so that accepting the same contractual text several times counts just as one time.

The H.530 protocol

The previously unknown attack on the H.530 protocol shown by the message sequence chart in Fig. 3 was discovered when we applied OFMC to automatically analyze this protocol in collaboration with Siemens [14,17]. It is a replay attack based on replaying old messages: the attack is caused by the lack of information in one protocol message and allows the intruder to masquerade as any honest agent. More specifically, the intruder first listens to a session between honest agents `MT` in role `mobileTerminal`, `VGK` in role `visitedGateKeeper`, and `AuF` in role `authenticationFacilityserver`. The intruder then starts a new session impersonating both `MT` and `AuF`. The weakness that makes the replay possible is the lack of fresh information in the message Ack_{VGK}, i.e., the message where `AuF` acknowledges to `visitedGateKeeper` that he is actually talking with `MT`. Replaying the corresponding message from the first session, the intruder impersonating `MT` can negotiate a new Diffie-Hellman key with `visitedGateKeeper`, “hijacking” `MT`’s identity. To perform the attack, the intruder must at least be able to eavesdrop and insert messages both on the connection between `mobileTerminal` and `visitedGateKeeper` and on the connection between `visitedGateKeeper` and `authenticationFacilityserver`.

The attack can be prevented by including the Diffie-Hellman half-key of `mobileTerminal` in the encrypted hash of the message Ack_{VGK}. With this extension, we have not found any further weaknesses in the protocol and Siemens has thus revised the protocol accordingly [54].

his (the intruder’s) computer crashed and he lost the signed contract and therefore asks the contract partner to run the contract signing again.

4 Concluding remarks

Thanks to its advanced methodologies and technologies for formal protocol specification and analysis, the AVISPA Tool is a state-of-the-art tool for the automatic validation of industrial-scale security protocols such as those in the AVISPA Library. Besides for the possible connection of new back-ends with complementary features (such as the preliminary work of [48]), current work is focusing in particular on further scaling up our specification and analysis techniques to classes of security problems that are at present outside of the scope of the AVISPA Tool. To this end, we have begun extending our approach in order to capture

- larger classes of protocols such as group protocols with evolving agent communities (we believe that the extension of CAPSL into MuCAPSL [68] will provide useful hints for the necessary extensions of our HLPSL),
- different network models such as wireless and ad hoc networks,
- composed security services, such as those provided by web services, and
- more complex security properties (including different channel properties, and properties of group and routing protocols).

As a concrete example, work is currently underway on scaling-up different formal analysis methods and tools to web services for security, e.g., CASPER [57,58] and TulaFale/ProVerif [19,20]. We have similarly begun applying the AVISPA Tool for the analysis of web services and the first results are very promising and we will report on them soon.

References

[1] M. Abadi, B. Blanchet, and C. Fournet. Just Fast Keying in the Pi Calculus. In *Proc. ESOP'04*, LNCS 2986, pp. 340–354. Springer, 2004.

[2] X. Allamigeon and B. Blanchet. Reconstruction of Attacks against Cryptographic Protocols. In *Proc. CSFW-18*, pp. 140–154. IEEE Computer Society Press, 2005.

[3] A. Armando, D. Basin, M. Bouallagui, Y. Chevalier, L. Compagna, S. Mödersheim, M. Rusinowitch, M. Turuani, L. Viganò, and L. Vigneron. The AVISS Security Protocol Analysis Tool. In *Proc. CAV'02*, LNCS 2404, pp. 349–354. Springer, 2002.

[4] A. Armando and L. Compagna. Automatic SAT-Compilation of Protocol Insecurity Problems via Reduction to Planning. In *Proc. FORTE 2002*, LNCS 2529, pp. 210–225. Springer, 2002.

[5] A. Armando and L. Compagna. Abstraction-driven SAT-based Analysis of Security Protocols. In *Proc. SAT'03*, LNCS 2919, pp. 257–271. Springer, 2003.

[6] A. Armando and L. Compagna. SATMC: a SAT-based Model Checker for Security Protocols. In *Proc. JELIA'04*, LNAI 3229, pp. 617–627. Springer, 2004.

[7] A. Armando and L. Compagna. An Optimized Intruder Model for SAT-based Model-Checking of Security Protocols. In *Proc. ARSPA'04*. Electronic Notes in Theoretical Computer Science 125(1):91–108, 2005.

[8] A. Armando, L. Compagna, and P. Ganty. SAT-based Model-Checking of Security Protocols using Planning Graph Analysis. In *Proc. FME'2003*, LNCS 2805, pp. 875–893. Springer, 2003.

[9] N. Asokan, V. Shoup, and M. Waidner. Asynchronous protocols for optimistic fair exchange. In *Proc. the 1998 IEEE Symposium on Research in Security and Privacy*, pp. 86–99. IEEE Computer Society Press, 1998.

[10] The AVISPA Project. URL: www.avispa-project.org/.

[11] HLPSL Tutorial: A Beginner's Guide to Modelling and Analysing Internet Security Protocols, 2005. Available at [10].

[12] AVISPA v1.0 User Manual, 2005. Available at [10].

[13] D. Basin. Lazy infinite-state analysis of security protocols. In *Proc. CQRE'99*, LNCS 1740, pp. 30–42. Springer, 1999.

[14] D. Basin, S. Mödersheim, and L. Viganò. An On-The-Fly Model-Checker for Security Protocol Analysis. In *Proc. ESORICS'03*, LNCS 2808, pp. 253–270. Springer, 2003.

[15] D. Basin, S. Mödersheim, and L. Viganò. Constraint Differentiation: A New Reduction Technique for Constraint-Based Analysis of Security Protocols. In *Proc. CCS'03*, pp. 335–344. ACM Press, 2003.

[16] D. Basin, S. Mödersheim, and L. Viganò. Algebraic intruder deductions. In *Proc. LPAR'05*, LNAI 3835, pp. 549–564. Springer, 2005.

[17] D. Basin, S. Mödersheim, and L. Viganò. OFMC: A symbolic model checker for security protocols. *International Journal of Information Security*, 4(3):181–208, 2005.

[18] G. Bella, F. Massacci, and L. C. Paulson. Verifying the SET Purchase Protocols. In *Journal of Automated Reasoning*, to appear.

[19] K. Bhargavan, C. Fournet, and A. Gordon. Verifying Policy-Based Security for Web Services. In *Proc. CCS'04*, pp. 268–277. ACM Press, 2004.

[20] K. Bhargavan, C. Fournet, A. Gordon, and R. Pucella. TulaFale: a security tool for web services. In *Proc. FMCO'03*, LNCS 3188, pp. 197–222. Springer, 2004.

[21] B. Blanchet. An efficient cryptographic protocol verifier based on prolog rules. In *Proc. CSFW'01*, pp. 82–96. IEEE Computer Society Press, 2001.

[22] B. Blanchet. Automatic Proof of Strong Secrecy for Security Protocols. In *Proc. IEEE Symposium on Security and Privacy*, pp. 86–100. IEEE Computer Society Press, 2004.

[23] C. Bodei, M. Buchholtz, P. Degano, F. Nielson, and H. Riis Nielson. Static validation of security protocols. *Journal of Computer Security*, 13(3):347–390, 2005.

[24] Y. Boichut, P.-C. Héam, and O. Kouchnarenko. Automatic verification of security protocols using approximations. Technical Report RR-5727, INRIA, 2005.

[25] Y. Boichut, P.-C. Heam, O. Kouchnarenko, and F. Oehl. Improvements on the Genet and Klay Technique to Automatically Verify Security Protocols. In *Proc. International Workshop on Automated Verification of Infinite-State Systems (AVIS'2004)*, pp. 1–11, 2004.

[26] M. Boreale and M. G. Buscemi. A framework for the analysis of security protocols. In *Proc. CONCUR 2002*, LNCS 2421, pp. 483–498. Springer, 2002.

[27] L. Bozga, Y. Lakhnech, and M. Perin. Hermes: An Automatic Tool for the Verification of Secrecy in Security Protocols. In *Proc. CAV'03*, LNCS 2725, pp. 219–222. Springer, 2003.

[28] L. Bozga, Y. Lakhnech, and M. Perin. Pattern-based abstraction for verifying secrecy in protocols. In *Proc. TACAS 2003*, LNCS 2619, pp. 299–314. Springer, 2003.

[29] The CAPSL Integrated Protocol Environment. URL: www.csl.sri.com/~millen/capsl/.

[30] Y. Chevalier. *Résolution de problèmes d'accessibilité pour la compilation et la validation de protocoles cryptographiques*. Phd, Université Henri Poincaré, Nancy, France, December 2003.

[31] Y. Chevalier, L. Compagna, J. Cuellar, P. Hankes Drielsma, J. Mantovani, S. Mödersheim, and L. Vigneron. A High Level Protocol Specification Language for Industrial Security-Sensitive Protocols. In *Proc. SAPS'04*, pp. 193–205. Austrian Computer Society, 2004.

[32] Y. Chevalier, R. Küsters, M. Rusinowitch, and M. Turuani. Deciding the Security of Protocols with Commuting Public Key Encryption. In *Proc. ARSPA'04*. Electronic Notes in Theoretical Computer Science 125(1):55–66, 2005.

[33] Y. Chevalier, R. Küsters, M. Rusinowitch, M. Turuani, and L. Vigneron. Deciding the Security of Protocols with Diffie-Hellman Exponentiation and Products in Exponents. In *Proc. FST TCS'2003*, LNCS 2914, pp. 124–135. Springer, 2003.

[34] Y. Chevalier, R. Küsters, M. Rusinowitch, M. Turuani, and L. Vigneron. Extending the Dolev-Yao Intruder for Analyzing an Unbounded Number of Sessions. In *Proc. CSL'2003*, LNCS 2803, pp. 128–141. Springer, 2003.

[35] Y. Chevalier, R. Küsters, M. Rusinowitch, and M. Turuani. An NP Decision Procedure for Protocol Insecurity with XOR. Theoretical Computer Science 338(1–3):247–274, 2005.

[36] Y. Chevalier and M. Rusinowitch. Combining Intruder Theories. In *Proc. ICALP 2005*, LNCS 3580, pp. 639–651. Springer, 2005.

[37] Y. Chevalier and L. Vigneron. Automated Unbounded Verification of Security Protocols. In *Proc. CAV'02*, LNCS 2404, pp. 349–354. Springer, 2002.

[38] Y. Chevalier and L. Vigneron. Strategy for Verifying Security Protocols with Unbounded Message Size. *Journal of Automated Software Engineering*, 11(2):141–166, 2004.

[39] J. Clark and J. Jacob. A Survey of Authentication Protocol Literature: V. 1.0, 17. Nov. 1997. URL: www.cs.york.ac.uk/~jac/papers/drareview.ps.gz.

[40] L. Compagna. *SAT-based Model-Checking of Security Protocols*. Phd, Università di Genova, Italy, and University of Edinburgh, U.K., 2005.

[41] R. Corin and S. Etalle. An Improved Constraint-Based System for the Verification of Security Protocols. In *Proc. SAS'02*, LNCS 2477, pp. 326–341. Springer, 2002.

[42] C. Cremers. Scyther. URL: www.win.tue.nl/~ccremers/scyther.

[43] G. Denker, J. Millen, and H. Rueß. The CAPSL Integrated Protocol Environment. Technical Report SRI-CSL-2000-02, SRI International, Menlo Park, CA, October 2000.

[44] D. Dolev and A. Yao. On the Security of Public-Key Protocols. *IEEE Transactions on Information Theory*, 2(29), 1983.

[45] B. Donovan, P. Norris, and G. Lowe. Analyzing a Library of Security Protocols using Casper and FDR. In *Proc. the Workshop on Formal Methods and Security Protocols*, 1999.

[46] T. Genet and F. Klay. Rewriting for cryptographic protocol verification. In *Proc. CADE'00*, LNCS 1831, pp. 271–290. Springer, 2000.

[47] T. Genet and V. Viet Triem Tong. Reachability Analysis of Term Rewriting Systems with Timbuk. In *Proc. LPAR'01*, LNCS 2250, 2001.

[48] A. Gotsman, F. Massacci, and M. Pistore. Towards an Independent Semantics and Verification Technology for the HLPSL Specification Language. In *Proc. ARSPA'05*. Electronic Notes in Theoretical Computer Science 135(1):59–77, 2005.

[49] P. Hankes Drielsma and S. Mödersheim. The ASW Protocol Revisited: A Unified View. In *Proc. ARSPA'04*. Electronic Notes in Theoretical Computer Science 125(1):141–156, 2005.

[50] P. Hankes Drielsma, S. Mödersheim, and L. Viganò. A formalization of off-line guessing for security protocol analysis. In *Proc. LPAR'04*, LNAI 3452, pp. 363–379. Springer, 2005.

[51] J. Heather, G. Lowe, and S. Schneider. How to prevent type flaw attacks on security protocols. In *Proc. CSFW'00*. IEEE Computer Society Press, 2000.

[52] ISO/IEC 9798-3: Information technology - Security techniques - Entity authentication - Part 3: Mechanisms using digital signature techniques, 1997.

[53] ITU-T Recommendation H.530: Symmetric Security Procedures for H.510 (Mobility for H.323 Multimedia Systems and Services), 2002.

[54] ITU-T Recommendation H.530, Corrigendum 1. 2003. Corrected version of [53].

[55] F. Jacquemard, M. Rusinowitch, and L. Vigneron. Compiling and Verifying Security Protocols. In *Proc. LPAR 2000*, LNCS 1955, pp. 131–160. Springer, 2000.

[56] C. Kaufman. Internet Key Exchange (IKEv2) Protocol, Oct. 2003. Work in Progress.

[57] E. Kleiner and A. Roscoe. Web Services Security: a preliminary study using Casper and FDR. Presented at the Workshop on Automated Reasoning for Security Protocol Analysis (ARSPA'04), 2004.

[58] E. Kleiner and A. Roscoe. On the relationship of traditional and Web Services Security protocols. Presented at the Twenty-first Conference on the Mathematical Foundations of Programming Semantics (MFPS'05), 2005.

[59] L. Lamport. The temporal logic of actions. *ACM Transactions on Programming Languages and Systems*, 16(3):872–923, 1994.

[60] L. Lamport. *Specifying Systems*. Addison-Wesley, 2002.

[61] G. Leduc and F. Germeau. Verification of Security Protocols using LOTOS – Method and Application. *Computer Communications*, 23(12):1089–1103, 2000.

[62] G. Lowe. Some new attacks upon security protocols. In *Proc. CSFW'96*. IEEE Computer Society Press, 1996.

[63] G. Lowe. A hierarchy of authentication specifications. In *Proc. the 10th IEEE Computer Security Foundations Workshop (CSFW'97)*, pp. 31–43. IEEE Computer Society Press, 1997.

[64] G. Lowe. Casper: a Compiler for the Analysis of Security Protocols. *Journal of Computer Security*, 6(1):53–84, 1998.

[65] W. Mao and K. G. Paterson. On the plausible deniability feature of internet protocols. 2004.

[66] C. Meadows. The NRL Protocol Analyzer: An Overview. *Journal of Logic Programming*, 26(2):113–131, 1996.

[67] C. Meadows. Analysis of the Internet Key Exchange Protocol Using the NRL Protocol Analyzer. In *Proc. the 1999 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, 1999.

[68] J. Millen and G. Denker. MuCAPSL. In *Proc. DISCEX III, DARPA Information Survivability Conference and Exposition*, pp. 238–249. IEEE Computer Society Press, 2003.

[69] J. C. Mitchell, M. Mitchell, and U. Stern. Automated Analysis of Cryptographic Protocols Using Murphi. In *Proc. the 1997 IEEE Symposium on Security and Privacy*, pp. 141–153. IEEE Computer Society Press, 1997.

[70] F. Oehl, G. Cécé, O. Kouchnarenko, and D. Sinclair. Automatic approximation for the verification of cryptographic protocols. In *Proc. Conference on Formal Aspects of Security*, LNCS 2629, pp. 33–48. Springer, 2003.

[71] L. C. Paulson. The Inductive Approach to Verifying Cryptographic Protocols. *Journal of Computer Security*, 6(1):85–128, 1998.

[72] M. Rusinowitch and M. Turuani. Protocol Insecurity with Finite Number of Sessions and Composed Keys is NP-complete. *Theoretical Computer Science*, 299:451–475, 2003.

[73] P. Ryan, S. Schneider, M. Goldsmith, G. Lowe, and B. Roscoe. *Modelling and Analysis of Security Protocols*. Addison-Wesley, 2000.

[74] V. Shmatikov and J. C. Mitchell. Analysis of a fair exchange protocol. In *Proc. the 1999 FLoC Workshop on Formal Methods and Security Protocols*, Trento, Italy, 1999.

[75] V. Shmatikov and J. C. Mitchell. Finite-state analysis of two contract signing protocols. *Theoretical Computer Science*, 283(2):419–450, 2002.

[76] D. Song. Athena: A new efficient automatic checker for security protocol analysis. In *Proc. CSFW'99*. IEEE Computer Society Press, 1999.

[77] M. Turuani. *Sécurité des Protocoles Cryptographiques: Décidabilité et Complexité*. Phd, Université Henri Poincaré, Nancy, France, December 2003.