AVISS — Automated Verification of Infinite State Systems

(IST-2000-26410)

Deliverable D3.5
Final Definition, Implementation and Experimentation with the Constraint-Logic Theorem-Prover.

1 Introduction

The AVISS prototype verification tool for security protocols has the architecture shown in Figure 1. Protocols are specified in the high-level protocol specification language HLPSL, and the HLPSL2IF translator developed in WP2 translates these specifications into the low-level but tool-independent Intermediate Format, IF. The design and implementation of the three approaches that work on the IF is the task of the third work-package WP3, which consists of three steps:

- *Encoding.* Specifications in the Intermediate Format are translated into tool-specific encodings that fall into the scope of application of our three model-checkers.
- *Experiments.* The model-checkers are applied to the verification problems of the corpus [5] of security protocols.
- *Tuning.* The experiments indicate ways to improve the encodings as well as the inference strategies implemented in the available automated deduction engines.

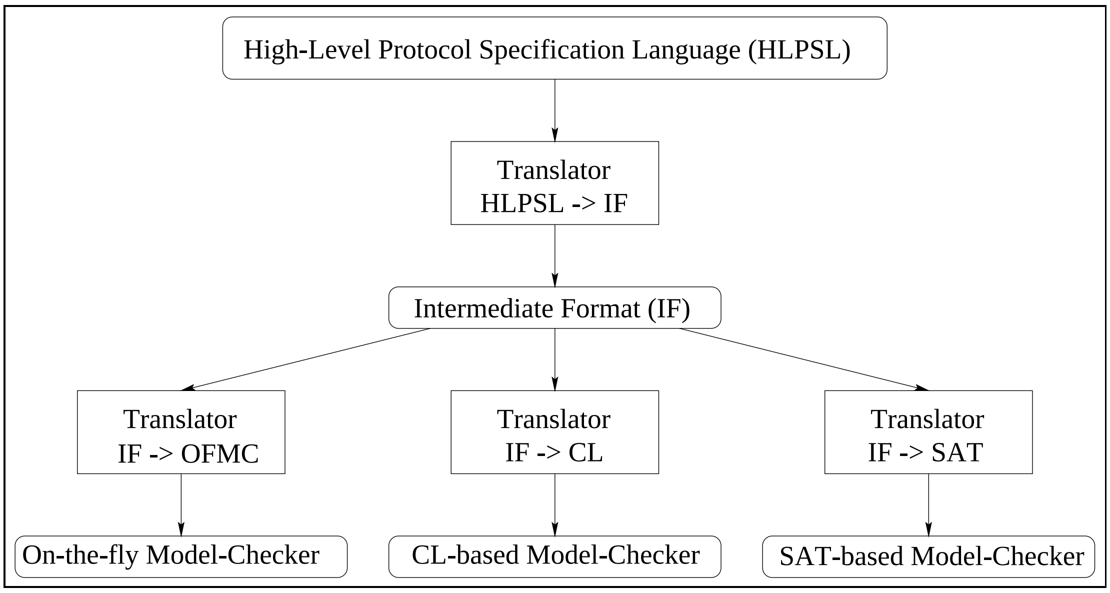

Figure 1: Architecture of the prototype verification tool

This deliverable D3.5 reports on our implementation of Task **T3.2** which includes

1. the translation from the IF format to input files for the CL theorem prover daTac
2. the implementation of a translator (Section 2),
3. the experimentation with problems from the corpus (see Section 4 for a discussion of the results given in Table 1).

In this document, we describe the implementation of the translator from the Intermediate Format to the input format of the theorem prover daTac (Section 2), we give a short description of the daTac theorem prover (Section 3), and then we describe the results we have obtained with it (Section 4).

Note that a detailed comparison of the relative performance of the three tools (see [2, 8, 9]) of the AVISS project will be subject of future deliverables. A preliminary comparison on a selection of examples is given in the paper [1], which we have written as part of deliverable D4.2. See also our own paper [4], and the project webpage

`www.informatik.uni-freiburg.de/~softech/research/projects/aviss`

2 Implementation of the translator from IF to CL

2.1 Presentation of the translator

We have implemented the translator from IF to CL in *Perl*, a script language well-suited for strings manipulation. Since *Perl* is also an object oriented language it allows for nice modular programming. This will enable easy extensions of the tool as well as its integration into a user-friendly script.

Currently, we have 3 different modules. The `IFTOCL.pm` module implements the organization of the translation mechanism. The `Writedatacfile.pm` module parses the output of the HLPSL2IF compiler which is related to the protocol and which is needed in our model namely a) the initial state rule, b) the messages rules and c) the goal rules. The last module, `Parallel.pm`, is only concerned with the parsing of the rules when using the parallel intruder model.

In addition to these modules, files describing the intruder's model and his behavior are included as needed.

2.2 Included files

These files are divided in two groups. In the first group, there are two files concerned with the parameters of the CL theorem prover daTac. The second group of files contains a description of the behavior of the lazy intruder.

Among them the file `simplif.dat` contains data which apply to the study of all protocols. This includes the authentication goals and a description about how the intruder handles knowledge constraints. All the included files are protocol-independent, and their inclusion only relies on the IF rules output by HLPSL2IF.

2.3 Modifications of the IF rules

In the translation, the `Writedatacfile.pm` program processes the output of HLPSL2IF. The initial state is left unchanged except for terms that are used for the handling of the lazy intruder.

The protocol rules are modified. First, we give the intruder the abilities to divert all messages and to impersonate any principal since under these assumptions it is possible to get a more efficient tool which is moreover complete with respect to attacks finding. We assume that the intruder diverts all messages that are sent by the principals. Thus, instead of applying a non-deterministic rule to model the diverting of messages by the intruder, we rather transform the rule:

$$\begin{array}{l} h(s(xTime)).w(0, mr(I), mr(a), etc, \ldots, true, xc) \\ \hfill \to h(xTime). \\ \mathbf{m(1, mr(a), mr(a), mr(b), crypt(pk(kb), c(nonce(c(Na, xTime)), mr(a))), xc)}. \\ \hfill w(2, mr(b), mr(a), c(nonce(c(Na, xTime)), etc), \ldots, true, xc) \end{array}$$

into a rule where the message are systematically diverted, and its content is added to the intruder's knowledge:

$$\begin{array}{l} h(s(xTime)).w(0, mr(I), mr(a), etc, \ldots, true, xc) \\ \hfill \to h(xTime).w \\ \hfill (2, mr(b), mr(a), c(nonce(c(Na, xTime)), etc), \ldots, true, xc). \\ \mathbf{i(c(mr(a), c(mr(b), crypt(pk(kb), c(nonce(c(Na, xTime)), mr(a))))))} \end{array}$$

2.4 Lazy intruder algorithm

The lazy intruder mechanism relies on handling constraints generated when the intruder sends messages to the principals (see [11] for more details). These constraints are not built-in in daTac, so we have to extend the protocol state by a term that is not part of the IF format. This term contains for all messages sent by the intruder the following subterm:

$$\texttt{mesg}(\mathit{Constraints}, \mathit{Knowledge})$$

where the *Knowledge* part represents the state of knowledge of the intruder when it sends a message. The *Constraints* part represents the constraints that the intruder solves in order to be able to send the message. The intruder can send another message only if he has reduced all the constraints associated to previous messages to simpler ones.

3 The constraint-based theorem prover

The aim of the daTac generic theorem prover is to do automated deduction in first-order logic with equality. Its specialty is to apply deductions modulo some equational properties of operators, such as commutativity or associativity-commutativity (AC). The deduction techniques implemented, based on resolution, paramodulation and term rewriting, have been proved refutationally complete in [12].

Currently, we use only on a small part of its possibilities, that is only resolution in an empty theory, but for the multi-set constructor dot '.', which has the same meaning as in the Intermediate Format.

4 Experimentation with the constraint-based theorem prover

In this section, we report upon the experimentations done with HLPSL2IF and daTac. This report contains a description of the flaws we have been searching for (part 4.1), as well as a comparison with the results obtained with other existing tools, namely Casper and FDR (experiments reported in [7]), the classical results reported in [5], and the results obtained by Brackin's tool [3].

4.1 Expressing goals

We have focussed on two important security goals: *authentication* and *secrecy* . We explain now how these two properties permit us to find many flaws on protocols.

Authentication goal

In this case, we say that a principal A *authenticates* a principal B on some values $V_1, \dots, V_n$ if whenever A has finished its part of the protocol, he may be confident in the following facts:

1. B has sent these values
2. as they were sent, the intended receiver was A
3. B and A agree on the role of these values
4. these values have not been received by A in a previous run of the protocol

The first condition is a condition on the messages sent by B. We do not assume here that the values $V_1, \dots, V_k$ were *created* by B. This allows the study of challenge/response messages, such as in:

1. $A \rightarrow B : \{Na, A\}Kb$
2. $B \rightarrow A : \{Na\}Ka$

...

Goal A authenticates B on Na;

The second condition is intended to capture the *'man in the middle'* attacks, where a dishonest principal fakes the identity of another principal, as in the following example, where *Bob* cannot authenticate *Alice* on Na:

$$\begin{array}{lll} Alice \rightarrow I & : & \{Na, Alice\}Kint \\ I(Alice) \rightarrow Bob & : & \{Na, A\}Kbob \\ Bob \rightarrow Alice & : & \{Na\}Kalice \end{array}$$

since the nonce Na was indeed sent by A, but the intended receiver was I, a dishonest principal.

The third condition is straightforward, and is used to ensure that there is no interference that leads to the confusion between two different parts of a message.

The last condition ensures the freshness of the values for A and it ensures that B has performed at least as many sessions with A as A has performed with B. A typical violation of this condition occurs when a message from a previous session may be replayed.

Secrecy goal

There is a *secrecy flaw* in a protocol if the knowledge of some data exchanged during this protocol should have been restricted to some trusted principals, but have been disclosed to another untrusted principal (the intruder).

Secrecy attacks follow in most of the cases from an authentication flaw. A common case is when two principals conduct a protocol to agree on the key, and when the intruder is able to substitute this key with a value he knows. There is an authentication failure on this key, in this case, and it will eventually lead to a secrecy flaw when, for example, the two principals will exchange confidential information using this key.

One wants sometimes to express that a value remains secret during some runs of the protocol but not forever. For example some protocols are used to exchange keys that have a limited lifetime. The key is used for some time in several sessions before being renewed.

A risk here is that during the interval the intruder, either using cryptanalysis, or by any other means, gets the secret key. The key exchange protocols ensure that even if this happens, the new key accepted by the principals will not be known by the intruder. Thus, we have introduced a *short term secret*. This is a secret that is kept as long as the protocol is not over, but is disclosed to the intruder afterwards, in order to simulate the above situation.

4.2 Protocols tested

Our approach is complete for finding flaws. Therefore, we focused on protocols that were known to be flawed and, in order to compare with existing tools, we have considered the list of flawed protocols in [3, 5, 7]. Among the protocols described in these references, only two protocols could not be handled:

1. The *Wide Mouthed Frog protocol*: its flaw is directly connected with the notion of Timestamps. Currently, we consider Timestamps as being simple nonces. This simplification leads, in this case, to a trivial flaw.

2. The *Diffie-Hellman Exchange protocol*: this protocol uses the associative and commutative properties of an exponential function, which we are not able to handle automatically yet.

4.3 Flaws

We describe here some of the flaws we have found on the protocols given in Table 1.

‘Man in the middle’ attacks

One of the most famous ‘man in the middle’ attack is the one discovered by Lowe on the *Needham Schroeder Public Key protocol.* If one considers the short version of the protocol (without the messages used by the principals to get the public keys from a server) the attack is:

$$\begin{array}{lcll}
A & \rightarrow & I & : \{Na, A\}Ki \\
I(A) & \rightarrow & B & : \{Na, A\}Kb \\
B & \rightarrow & I(A) & : \{Nb, Na\}Ka \\
I & \rightarrow & A & : \{Nb, Na\}Ka \\
A & \rightarrow & I & : \{Nb\}Ki \\
I(A) & \rightarrow & B & : \{Nb\}Kb
\end{array}$$

In this attack, the intruder I uses the honest principal A as an oracle to decrypt the cipher $\{Nb, Na\}Ka$. In this way he can learn Nb and compose the last message. This example leads to an *authentication flaw*, since B cannot authenticate A when receiving back Nb (i.e. B believes he is talking not to the intruder but to A).

In general, if one thinks that a principal may be used as an oracle, as it is the case here with A, it is possible to declare this principal in the **Parallel** section of the high level description file. This allows HLPSL2IF to compile the advantage that the intruder would get with this oracle. It often speeds up the experimentations with the protocol but may lead to some state-space explosion problems.

Compromised key attacks

We assume now that the intruder, using external information, learns the value of some data such as a key that was exchanged during a previous session of the protocol. This possibility of learning a value, in this case, is not a problem of the protocol. But it may lead to an attack on the protocol if the intruder, using this knowledge, can convince some participant of a protocol session to accept again this piece of data as a new secret. This is what happens in the *Kao and Chow key exchange protocol.* First, a session is executed without any interference from the intruder:

$$\begin{array}{llcll}
1. & A & \rightarrow & S & : A, B, Na \\
2. & S & \rightarrow & B & : \{A, B, Na, Kab\}Kas, \{A, B, Na, Kab\}Kbs \\
3. & B & \rightarrow & A & : \{A, B, Na, Kab\}Kas, \{Na\}Kab, Nb \\
4. & A & \rightarrow & B & : \{Nb\}Kab
\end{array}$$

Suppose now that the intruder has learned the value Kab. He can then lead the principal B to use this compromised key again thanks to the following sequence of messages:

$$\begin{array}{llcll}
2. & I(S) & \rightarrow & B & : \{A, B, Na, Kab\}Kas, \{A, B, Na, Kab\}Kbs \\
3. & B & \rightarrow & I(A) & : \{A, B, Na, Kab\}Kas, \{Na\}Kab, Nb_2 \\
4. & I(A) & \rightarrow & B & : \{Nb_2\}Kab
\end{array}$$

The first message of the intruder is nothing but a replay of the message of the previous session. B replies with a nonce Nb_2, different from the nonce of the previous session Nb. The problem is that B has accepted the old key as his new key. Since the intruder knows the old key he can compose the last answer to B.

This kind of flaw may affect protocols that are executed at periodic times, such as one-time-password protocols or protocols in which the key exchanged has a limited life time.

Type flaws

Type flaws are flaws that depend on the implementation of the protocol. They may not be serious problems and are sometimes considered as artifacts (e.g. in [7]). However it is important to detect them in order to guide the implementation of a protocol. Note also that we do not consider type flaws as being flaws by themselves, but only problems that lead to an actual flaw, such as in the *Otway Rees key exchange protocol* below.

This protocol is subject to a type flaw. By exploiting it, the intruder can send to a principal a key that the intruder knows, whereas the principal believes this key was sent by another principal. This type flaw leads to both a secrecy and an authentication flaw, as shown by the following sequence of messages:

$$\begin{array}{lcl} A \rightarrow I(B) &:& M, A, B, \{Na, M, A, B\}Kas \\ I(B) \rightarrow A &:& M, \{Na, M, A, B\}Kas \quad (\equiv M, \{Na, Kab\}Kas) \end{array}$$

The principal A believes that only he and B know the value of Kab, but the intruder may compose it since he knows M, A, B (secrecy flaw) and B is not even aware that A has tried to communicate with him (authentication flaw)...

There is also a type flaw on the *Needham Schroeder Public Key protocol* when we take into account the the messages sent to the server ([10]).

Freshness attack

This kind of attack is problematic when the number of times the protocol was executed is relevant. An obvious example is a payment order sent to a bank. It is not enough to ensure the identity of the sender and of the receiver, one also wants to be sure that the sum is not debited several times. It is treated as an authentication attack, since in our model, authentication also takes freshness into account. The family of ISO protocols is concerned by this kind of flaw. For example, once the following message has been issued:

$$A \rightarrow B : Text2, \{Na, B, Text1\}Kab$$

it is very easy, for the intruder, to replay it as many times he wants:

$$I(A) \rightarrow B : Text2, \{Na, B, Text1\}Kab$$

The problem follows from the specification of the protocol, which states that Na can be a nonce (and thus permits the given attack) or a timestamp (in which case the given attack is detected if B keeps track of the previous messages he has received and whose timestamps are still valid) which depends, quoting the protocol specification, *'on the technical capabilities of [A] and [B] as well as the environment'*.[5]

4.4 Comments on the results of the experiments (see Table 1)

Comments on the failed cases

We failed to find a flaw that was already known in only two cases. The first one is the Yahalom protocol:

Protocol	Our results	Lowe's results	Clark & Jacob's results	Brackin's results	Kind of flaw
Protocols using symmetric encryption					
ISO Symmetric Key One-Pass Unilateral Authentication Protocol	Attack	Attack	No attack	No attack	Authentication
ISO Symmetric Key Two-Pass Mutual Authentication Protocol	Attack	Attack	No attack	No attack	Authentication
Andrew Secure RPC Protocol	Attack	Attack	Attack	Attack	Authentication
Davis Swick Private Key Certificates, Protocol 1	Attack	Attack	Attack	Attack	Authentication
Davis Swick Private Key Certificates, Protocol 2	Attack	Attack	Attack	Attack	Authentication
Davis Swick Private Key Certificates, Protocol 3	Attack	Attack	No attack	No attack	Authentication
Davis Swick Private Key Certificates, Protocol 4	Attack	Attack	No attack	No attack	Authentication
Denning Sacco Symmetric Key Protocol	Attack	No attack	No attack	No attack	Authentication
Wide Mouthed Frog Protocol	Unanalyzed	Attack	Attack	No attack	Authentication
Needham Schroeder Protocol with Conventionnal Key	Attack	No attack	Attack	Attack	Authentication
Otway Rees Key exchange Protocol	Attack	Attack	Attack	No attack	Authentication
Yahalom Protocol	No attack	No attack	Attack	Attack	Authentication
Woo and Lam Authentication Protocol Π_1	Attack	Attack	Attack	No attack	Authentication
Woo and Lam Authentication Protocol Π_2	Attack	Attack	Attack	No attack	Authentication
Woo and Lam Authentication Protocol Π_3	Attack	Attack	Attack	No attack	Authentication
Woo and Lam Authentication Protocol Π	Attack	Attack	Attack	No attack	Authentication
Woo and Lam Mutual Authentication	Attack	Attack	Attack	No attack	Authentication
Needham Schroeder Signature Protocol	Attack	Attack	No attack	Attack	Authentication
Kao Chow Repeated Authentication Protocol	Attack	No attack	Attack	Attack	Authentication
Kehne Langendorfer Schoenewalder	Attack	Attack	Attack	No attack	Authentication
Neumann Stubblebine	Attack	Attack	Attack	No attack	Authentication
Protocols using hash functions					
ISO One-Pass Unilateral Authentication Protocol with CCFs	Attack	Attack	No attack	No attack	Authentication
ISO Two-Pass Mutual Authentication Protocol with CCFs	Attack	Attack	No attack	No attack	Authentication
Protocols using public key encryption					
ISO Public Key One-Pass Unilateral Authentication Protocol	Attack	Attack	No attack	No attack	Authentication
ISO Public Key Two-Pass Mutual Authentication Protocol	Attack	Attack	No attack	No attack	Authentication
Diffie-Hellman Exchange	Unanalyzed	Unanalyzed	No attack	No attack	Authentication
Needham Schroeder Public Key Protocol	Attack	Attack	Attack	No attack	Authentication
SPLICE/AS Authentication Protocol	Attack	Attack	Attack	No attack	Authentication
Hwang and Chen's modified SPLICE/AS Authentication Protocol	Attack	Attack	Attack	No attack	Authentication
Denning Sacco Key Distribution with Public Key	Attack	Attack	Attack	No attack	Authentication
CCITT X.509	No attack	Attack	Attack	No attack	Authentication
Shamir Rivest Adelman Three Pass Protocol	Attack	Attack	Attack	Unanalyzed	Secrecy
Encrypted Key Exchange Protocol	Attack	Attack	Attack	No attack	Authentication
TMN Key Exchange Protocol	Attack	Unanalyzed	Unanalyzed	Unanalyzed	Secrecy

Table 1: Comparison of our approach with Lowe's, Clark&Jacob's and Brackin's results

Protocol `Yahalom`;
Identifiers
A, B, S : *User*;
Kas, Kbs, Kab : *Symmetric_Key*;
Na, Nb : *Number*;
Knowledge
A : B, S, Kas;
B : A, S, Kbs;
S : A, B, Kas, Kbs;
1. $A \rightarrow B$: A, Na
2. $B \rightarrow S$: $B, \{A, Na, Nb\}Kbs$
3. $S \rightarrow A$: $\{B, Kab, Na, Nb\}Kas, \{A, Kab\}Kbs$
4. $A \rightarrow B$: $\{A, Kab\}Kbs, \{Nb\}Kab$

Session_instances
$[A : a; B : b; S : se; Kas : kas; Kbs : kbs]$;
Intruder *Divert*, *Impersonate*;
Intruder_knowledge a, b, se;
Goal *Correspondence_Between A B*;

The problem with this protocol is that it is possible for the intruder to compose the first part:

$$\{A, Kab\}Kbs$$

of the last message using a possible type flaw, which leads to a confusion between this part and the second part of the second message:

$$\{A, Na, Nb\}Kbs$$

By examining the execution trace of daTac we note that the intruder tries to exploit this possible weakness. But since he does not know the value of Nb he will not be able to compose the last message. Lowe has already noted that this *error* was in fact an artifact in [7].

We did not find either any attack on the CCITT X.509 protocol. The attacks given in the literature relate to special properties of the principals. That is, one cannot ensure that the sender of a message knows all the parts of this message. We cannot model this kind of flaw yet, since it may be expressed as 'the principal does *not* know a value', which is not a combination of secrecy or authentication flaws.

About the Davis and Swick family of protocols, one shall note that flaws that are found are already considered in [6], but the authors consider they are not important, since a flawed execution of the protocol will be detected later by the honest principals.

A novel attack on Denning Sacco symmetric key protocol

In all the papers surveyed so far, the *Denning Sacco symmetric key protocol* was considered to be secure. It was conceived as a correction of the *Needham Schroeder symmetric key protocol.* The sequence of messages, in this protocol, is:

$A \rightarrow S$: A, B
$S \rightarrow A$: $\{B, Kab, T, \{A, Kab, T\}Kbs\}Kas$
$A \rightarrow B$: $\{A, Kab, T\}Kbs$

where T is a timestamp. In this protocol, A forwards to B a part of the message sent by the server S. The problem in this protocol is that messages 2 and 3 are of the same global shape. This fact can be exploited in the following sequence of messages:

$$\begin{array}{lcl}
I(B) \rightarrow S & : & B, A \\
S \rightarrow I(B) & : & \{A, Kab, T, \{B, Kab, T\}Kas\}Kbs \\
I(A) \rightarrow B & : & \{A, Kab, T, \{B, Kab, T\}Kas\}Kbs\ (\equiv \{A, Kab, T\}Kbs)
\end{array}$$

After this messages exchange B accepts a new value for the symmetric key he shares with A, whereas A is not aware that a protocol session took place. This attack relies on a type flaw. In this case, one shall note that it is unlikely that when receiving the whole message B considers $T, \{B, Kab, T\}Kas$ to be a timestamp. But the implementation of this protocol may lead to a real flaw in two cases:

1. first, in case there is some padding in the encryption, it is possible that B just does not check the bits following the part of the message he is waiting for. Since the two ciphers begin with the same data type, there is a real possibility for B to accept the message, thus leading to the flaw;
2. second, it may happens that, if a bloc-cipher algorithm is used (such as DES, for example), the second message is split into smaller parts, from which the intruder will be able to construct a message acceptable by B, however cautious B is. This will be the case if a bloc finishes right after T.

Thus, we believe this type flaw should not be regarded as an artifact, and that it should be considered for an actual implementation of this protocol.

5 Conclusion

Although there are protocols that cannot be expressed in the high-level specification language our approach, in comparison with related ones is rather effective. This is justified by two reasons.

Firstly, a quick look at Table 1 shows that we are able to correctly find out whether a protocol is flawed. But the Table does not show that some protocols may have multiples errors, and in this case, our tool permits to find all of them. This happens with the *Otway Rees key exchange protocol*, where Lowe finds an authentication error, and Clark and Jacob report a type flaw (Brackin does not find any flaw). We are able to find both flaws in this case, thanks to the expressiveness of the goal we use, and to the lazy intruder model, which permits to catch type flaws. The point, here, is that we have been able to automatically find a type flaw, whereas another more standard method was unable to do so.

Secondly, one should note that we are safe from finding 'artificial' errors. Moreover the 'artificial' error recorded by Clark and Jacob in the *Yahalom key exchange protocol*, and its similarity with the type flaw of *Otway Rees key exchange protocol*, indicate a suspect behavior of the tools considered by Clark and Jacob since sometimes they reveal real flaws, and sometimes reveal artifacts.

Two new protocol verification techniques were developed in Nancy during the AVISS project. The first one is the lazy intruder technique, which permits us to reduce the search for a flaw to the search in a finite and small state space in the case of a finite number of principals. The second one is the concept of *principals in parallel*, which permits to study in a model-checking framework the simplified case where an unbounded number of principals may act at the same time.

As noted above, a detailed comparison of the relative strengths of the three tools of the AVISS project will be subject of future deliverables (see [4, 8, 9]). A preliminary comparison on a selection of examples is given in [1].

Bibliography

[1] 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. January 2002.

[2] A. Armando and L. Compagna. Automatic SAT-Compilation of Security Problems. January 2002.

[3] S. H. Brackin. Evaluating and Improving Protocol Analysis by Automatic Proof. In *Proc. of the 11th IEEE Computer Security Foundation Workshop*. IEEE Computer Society Press, 1998.

[4] Y. Chevalier and L. Vigneron. Automated Unbounded Verification of Security Protocols. Technical Report 4369, LORIA, Vandoeuvre les Nancy, February 2002.

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

[6] D. Davis and R. Swick. Network Security via Private Key Certificates. In *ACM Operating System Review*, 1990.

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

[8] Freiburg Group. Deliverable 3.4 : Final definition, implementation and experimentation with the on-the-fly model-checker. 2002.

[9] Genoa Group. Deliverable 3.6 : Final definition, implementation and experimentation with the SAT model-checker. 2002.

[10] C. Meadows. Analyzing the Needham-Schroeder Public-Key Protocol: A Comparison of Two Approaches. In *ESORICS: European Symposium on Research in Computer Security*. LNCS, Springer-Verlag, 1996.

[11] Protheo - Loria - Group. Deliverable 2.2: Specification of the Intermediate Format IF. 2001.

[12] L. Vigneron. Positive Deduction modulo Regular Theories. In H. Kleine-Büning, editor, *Proceedings of Computer Science Logic*, LNCS 1092, pages 468–485, Berlin, 1995. Springer-Verlag. URL: `www.loria.fr/equipes/protheo/SOFTWARES/DATAC/`.