AVISPA

www.avispa-project.org

IST-2001-39252

Automated Validation of Internet Security Protocols and Applications

Deliverable 4.3: Heuristics

Abstract

We report on our research on formalizing and employing heuristics in the AVISPA tool for security protocol analysis. More specifically, we present a collection of heuristics, which substantially reduce search and improve the performance of the back-ends of the AVISPA tool.

Deliverable details

Deliverable version: *v1.0*
Date of delivery: *12.12.2003*
Classification: *public*

Person-months required: *5*
Due on: *30.11.2003*
Total pages: *15*

Project details

Start date: *January 1st, 2003*
Duration: *30 months*
Project Coordinator: *Alessandro Armando*
Partners: *Università di Genova, INRIA Lorraine, ETH Zürich, Siemens AG*

Information Society Technologies

Project funded by the European Community under the *Information Society Technologies* Programme (1998-2002)

Contents

1 Introduction

Heuristics are practical techniques typically applied to reduce the time spent searching for solutions to problems. They have found their way into many application domains, ranging from Number Theory (for instance, Pollard's Rho heuristics for integer factoring) to Artificial Intelligence (see the discussion of A* in Section 3.1).

A heuristics refers to a problem-solving procedure about which one cannot make formally verifiable statements regarding completeness and/or running time, but which, in practice, is complete and efficient for many cases. For the purposes of this deliverable, we adopt the generally accepted, though somewhat looser, definition of a heuristics as a "rule of thumb" procedure yielding a correct and efficient solution to a problem. In our case, we are interested in heuristics that speed up the search for attacks on security protocols.

Heuristics are particularly applicable to the falsification problem for security protocols, because the nature of the model-checking problem – namely, search through a transition system – lends itself well to the use of heuristic methods to speed up search. On the verification side, however, applicability of heuristics is limited, as we use different techniques in the search to prove that desired properties of a given protocol hold. In this deliverable, we therefore limit the discussion to heuristics for faster attack detection.

Different classes of heuristics are applicable to the problem of protocol analysis. Some can be applied at compile time, yielding a heuristically-modified IF [2] specification (see, for instance, the discussion of session compilation in Section 2). Others, like the "constraining rule variables" heuristics, described in Section 4, are problem-specific and applicable during the search for attacks. These are particular to the problem of protocol falsification and may also depend on the model used in a specific back-end to the AVISPA tool. In still other cases, we can apply generic search heuristics by adapting them to our problem domain. In Section 3, we discuss the application of one standard search heuristic, A* [12], and its adaptation to the problem of protocol analysis.

2 Session Compilation

The session compilation heuristics is a heuristics that we have devised to speed up the detection of replay attacks, and that can be applied at compile time, yielding a heuristically-modified IF specification. In general, following Syverson [16], one can take

> "*replay* to mean the capture of a message or a piece of a message

> that is then used at a later time. This encompasses both cases where the original message is allowed to pass unimpeded and cases where it is prevented from arriving."

Malladi et al. [14] consider a more general definition of replay attack:

> "an attack on a security protocol using replay of messages from a different context, thereby fooling the honest participant(s) into thinking they have successfully completed the protocol run."

This definition has, however, the drawback that it does not distinguish replay attacks from other parallel session attacks [8], which occur when two or more protocol runs are executed concurrently and messages from one are used to form messages in another (such as, for instance, in the case of the man-in-the-middle attack to the Needham-Schroeder Public Key protocol NSPK).

Let us thus consider a more restricted form of replay attack, in which the attack can be separated in two phases (between which a long period of time may have occurred), where

- in the first phase the intruder learns messages that he replays in the second phase, and
- the sessions (partial runs of the protocol) in both phases are disjoint.

This is the intuition underlying the session compilation heuristics, in which we further impose the restriction that the intruder is passive in the first phase and just eaves-drops on communication between honest agents, while in the second phase he is active and interacts with the honest agents by sending message, possibly replaying messages (or parts thereof) of the first phase.[1]

The session compilation heuristics allows us to model replay attacks of this form without the need to search the space that would result from having the current session be executed in parallel with the ones that the intruder needs to be active in to carry out the attack. These other forms of replay will of course be covered, albeit less efficiently than the cases tackled with our heuristics, by the "normal" executions of our AVISPA tool on parallel sessions of the same protocol.

The heuristics works simply by pre-compiling a protocol session (and hence its name) and adding it to the intruder knowledge. That is, for every declared session instance between honest agents, we pre-compute the messages that are exchanged by the agents according to the protocol (inserting a

[1]Note that our heuristics includes the straightforward extension where the intruder takes part in the first session under his real name and obeys to the protocol (i.e. he behaves as a honest agent).

Without session compilation:

```
1.1. a    -> i(c) : a, b, n1(1)
2.1. i(a) -> c    : a, b, n1(1)
2.2. c    -> i(a) : c, {c,a,n1(1),kb}inv(kc)
1.2. i(c) -> a    : c, {c,a,n1(1),kb}inv(kc)
1.3. a    -> i(b) : a, b, {a,{n2(3)}kb}inv(ka)
3.3. i(a) -> b    : a, b, {a,{n2(3)}kb}inv(ka)
3.4. b    -> i(c) : a, b, n3(4)
2.4. i(b) -> c    : a, b, n3(4)
2.5. c    -> i(b) : c, {c,b,n3(4),ka}inv(kc)
3.5. i(c) -> b    : c, {c,b,n3(4),ka}inv(kc)
3.6. b    -> i(a) : b, a, {b,n2(3)+1}ka
4.1. i(a) -> c    : a, b, n1(7)
4.2. c    -> i(a) : c, {c,a,n1(7),kb}inv(kc)
5.3. i(a) -> b    : a, b, {a,{n2(3)}kb}inv(ka)
5.4. b    -> i(c) : a, b, n3(8)
4.4. i(b) -> c    : a, b, n3(8)
4.5. c    -> i(b) : c, {c,b,n3(8),ka}inv(kc)
5.5. i(c) -> b    : c, {c,b,n3(8),ka}inv(kc)
```

With session compilation:

```
1.1. i(a) -> c    : a, b, n1(7)
1.2. c    -> i(a) : c, {c,a,n1(7),kb}inv(kc)
2.3. i(a) -> b    : a, b, {a,{n2(3)}kb}inv(ka)
2.4. b    -> i(c) : a, b, n3(8)
1.4. i(b) -> c    : a, b, n3(8)
1.5. c    -> i(b) : c, {c,b,n3(8),ka}inv(kc)
2.5. i(c) -> b    : c, {c,b,n3(8),ka}inv(kc)
```

Figure 1: The replay attack on the SPLICE/AS Authentication Protocol without and with session compilation.

fresh constant for each generated nonce, etc.) and then add these messages to the initial intruder knowledge and start the analysis as usual.

As a concrete example, consider Figure 1, which shows the replay attack trace that is output by OFMC for the SPLICE/AS Authentication Protocol [8]. This is a mutual authentication protocol between a client `a` and a server `b` using a certification authority `c` to distribute public keys where necessary, and the goal that is violated and gives rise to the replay attack in the figure is that `b` authenticates `a` on nonce `n2(3)`.

As shown in the figure, session compilation shortens the attack trace and thus speeds up the analysis. In the first protocol session in the attack trace without session compilation, the intruder simply forwards the messages between the agents, and he becomes active only in the second session. Session compilation exploits precisely this: the initial intruder knowledge is extended with a pre-compiled protocol session (as if the intruder had just eaves-dropped it) so that the intruder sends (in fact, replays) to `b` a message that contains a nonce `n2(3)`, representing a constant used in the first session.

In order to assess the effectiveness of the session compilation heuristics, we have prototypically integrated it into the HLPSL2IF translator so to be able to automatically generate a heuristically-modified IF specification of a protocol that should be analysed with respect to (our form of) replay attacks. We have then employed the OFMC back-end of the AVISPA tool [4, 6] and carried out a number of experiments on those protocols of the Clark/Jacob library [8] that suffer from (our form of) replay attacks. As the results in Table 1 show, the heuristics is indeed very effective in speeding up the detection of these attacks. Note that we write "undef." to denote that we cannot measure the speed up, and that there is also a case where session compilation does not provide a speed up but rather slows the search, which is due to the larger initial state (i.e. the larger initial knowledge of the intruder). The two final rows sum up the speed up provided by session compilation. The total analysis time for the protocols listed in the table drops from 1.64 to 0.14 seconds, which corresponds to a total speed up of 11.71. Moreover, the total analysis time for the entire library drops from 2.71 to 1.64 seconds even though many examples are not replay attacks.

The integration of the session compilation heuristics into the HLPSL2IF translator allows the other back-ends of the AVISPA tool to benefit from this heuristics as well. This also opens up the possibility of providing experimental results on the applications of session compilation to a larger test-bench of protocols, in particular on the industrial-scale protocols of the AVISPA library (cf. deliverable [5]).

Table 1: Performance of the OFMC back-end for the detection of replay attacks without and with session compilation (sessco); times in seconds.

Protocol Name	Without sessco	With sessco	Speed up
ISO symmetric key 1-pass unilateral authentication	0.01	0.01	1
ISO symmetric key 2-pass mutual authentication	0.00	0.01	none
ISO CCF 1-pass unilateral authentication	0.01	0.00	undef.
ISO CCF 2-pass mutual authentication	0.01	0.00	undef.
Needham-Schroeder Conventional Key	0.04	0.01	4
Denning-Sacco (symmetric)	0.01	0.00	undef.
Kao Chow Repeated Authentication, 1	0.04	0.01	4
Kao Chow Repeated Authentication, 2	0.04	0.02	2
Kao Chow Repeated Authentication, 3	0.05	0.01	5
ISO public key 1-pass unilateral authentication	0.01	0.00	undef.
ISO public key 2-pass mutual authentication	0.01	0.00	undef.
SPLICE/AS Authentication Protocol	0.18	0.02	9
Davis Swick Private Key Certificates, protocol 1	0.04	0.01	4
Davis Swick Private Key Certificates, protocol 2	1.19	0.04	29.75
Total	1.64	0.14	11.71
Total for the entire Clark/Jacob library	2.71	1.64	1.65

3 Standard Search Heuristics

The strengths and weaknesses of standard search procedures are well known: while breadth-first search is space-inefficient, depth-first search can require too much time and does not always return the optimal path from the initial state to the goal state. Model checking problems involving very large or infinite state spaces can thus greatly benefit from the application of heuristic search algorithms, and the AVISPA tool is no exception.

3.1 A* and IDA*

Heuristic search algorithms like A* [12] and its iterative deepening variant IDA* [13] improve on the naïve search techniques of breadth-first search and depth-first search by ranking paths in the search tree according to an estimated cost. The search is then prioritised, expanding those nodes with minimum cost first in order to reach goal states faster. Both algorithms were proposed by the Artificial Intelligence community; however, [11] shows how they can be applied to model checking problems, yielding a technique the authors call "directed model checking". This often results in a significant reduction in search time, and properties of the heuristic search algorithms ensure that, for so-called *admissible* heuristics (that is, those that never overestimate the cost from the current state to the goal state), the resulting trace from initial to goal state is of optimal length.

A* defines two cost functions for evaluation of node expansions: for a given node u, $g(u)$ denotes the optimal/minimal cost so far to reach node u, and $h(u)$ is the estimated cost of the cheapest path from u to a goal state. Search then proceeds by successively visiting nodes with minimum summed cost $f(u) = g(u) + h(u)$.

We are confident that directed model checking techniques can be applied within the AVISPA tool and will speed up the detection of attacks. There are several possible instances of A*/IDA* that we can consider, i.e. concrete definitions of the heuristic cost function h. We present here the possible instances on which we are currently working.

3.2 Rule Ordering

Before presenting a technique that we call 'rule ordering', let us first introduce the underlying model that we assume. We assume a protocol consists of n steps in total, which are assigned step numbers $1, 2, ..., n$ in ascending order of intended execution. Each agent in the protocol A participates in some number n_A of these steps. Locally, an agent numbers these steps $1, 2, ..., n_A$ in the order in which he executes them and maintains his current state $State_A$, which ranges from 1 (the initial state) to $n_A + 1$ (after execution of the last transition).

We assume that the intruder has successfully executed an attack once he has brought at least one honest participant in a protocol run to complete a session in which he has in fact violated one or more of the protocol's security objectives. This gives us a lower bound of the number of steps required to complete an attack: namely, the minimum number of steps required for *some* agent to complete a protocol run. We can ensure that our heuristics are admissible if, for any node, we never assign weights exceeding this lower bound.

To this end, for each agent A, we define the number of steps required for him to complete his part of the current protocol run, d_A, as follows:

$$d_A = \begin{cases} n_A + 1 - State_A\,, & \text{if } State_A \neq n_A + 1 \\ n_A\,, & \text{otherwise} \end{cases}.$$

That is, so long as agent A has not yet completed his part of the current protocol run (i.e. $State_A \neq n_A{+}1$), then d_A equals $n_A{+}1{-}State_A$. Otherwise, A has already finished the current protocol run, and we assign $d_A = n_A$, expressing that this agent must start a new protocol run and execute all n_A steps before completing it.

This provides the sought-after lower bound: namely, the minimum value of d_A over all agents A. We call this lower bound m and define it, for a protocol executed by the set $Agents$ of agents, as follows:

$$m = min(\{d_A \mid A \in Agents\}) .$$

The idea underlying the rule ordering technique is that we can find attacks faster if we execute the highest possible protocol step first. That is, we order the IF rules corresponding to the protocol by giving priority to the rules that correspond to the highest protocol steps (so that the initial step has lowest priority). This order naturally induces an order on the successor states, so that those successors that are generated by the highest protocol steps are expanded first. As a consequence, the search focuses on the quickest way for the intruder to bring a protocol run to completion.

This speeds up the detection of some kinds of attacks, in particular those in which one honest participant is impersonated, in part or in total, by the intruder. The reason is simply that priority is assigned to those paths in the search tree in which progress towards completion of protocol runs is made rather than paths in which the intruder starts new protocol runs (which may not be necessary to stage an attack) before finishing existing ones.

For each node in the graph u (excluding the root), we label the transition leading to u with the corresponding (global) step number and denote this s. The weight function h is then defined as follows:

$$h(u) = m - \frac{s}{n+1} .$$

Admissibility follows from the discussion above: no cost overestimate is possible since the intruder must bring at least one agent's session to completion, a process requiring at least m steps. Subtracting the term $s/(n+1)$ serves to prioritise execution based on the step numbers of the rules.

This heuristics is already implemented in the current version of the backend OFMC [4, 6]. It was identified early in the development of OFMC as a "rule of thumb" that could speed up attack detection. We feel that the formulation of this technique as an instance of A*/IDA*, as presented here, will allow for a more general view of search heuristics with the possibility to exploit existing algorithms and proven properties.

3.3 Constraint Differentiation-Based Heuristic Search

Constraint differentiation (see [7] and also deliverable 4.2, [3]) is a correct and complete reduction technique, inspired by partial-order reduction [9, 15], to eliminate redundant search in the symbolic state space.

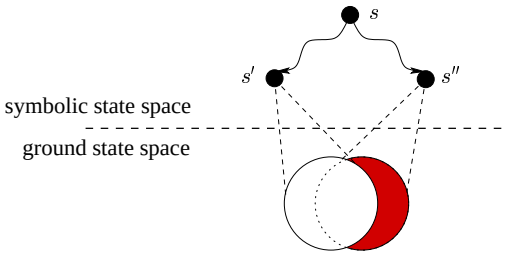

Figure 2: The intuition behind constraint differentiation.

Figure 2 illustrates the intuition behind this technique. Consider two symbolic states s' and s'' that can be reached from some state s by different interleavings of the same actions, e.g. message exchanges. In practice, there is often a substantial overlap between the sets of ground states represented by s' and s''. Constraint differentiation exploits these overlaps by restricting the constraints of s'' to those ground states that are not already covered by s', i.e. the shaded part in Figure 2. (Symmetrically, we could restrict s' instead of s''.) In essence, we *differentiate*, on the basis of intruder knowledge, between the two symbolic states by considering only the portion of s'' in which we require that the intruder make use of some "new" knowledge: that is, knowledge gained in the transition from s to s''.

Building on these ideas, we can also use the notion of "new" intruder knowledge as a heuristics to prioritise our search, assigning higher priority to node expansions in which the intruder uses items in his knowledge that he has recently learnt. In doing so, we sacrifice provable completeness while retaining the correctness of the technique.

Work is underway to define heuristic functions h that would prioritise the search according to newness of information. Several possibilities exist. One can take an "all or nothing" approach, designating those node expansions in which the intruder uses new knowledge as more desirable than those in which he does not. Alternatively, one can define a notion of the amount of new knowledge used: that is, if the intruder sends message m, those node expansions in which the greatest number of subterms of m are new are assigned lowest cost. Finally, one can consider extending this heuristics by assigning an "age" to each item in the intruder's knowledge and then prioritising those node expansions in which the messages sent intruder are the "youngest" with respect to the sums of the ages of their subterms.

We are optimistic that these heuristics and others will result in a speed up of attack detection within the AVISPA tool, thus extending its scope and usefulness. Moreover, by building on the ideas of constraint differentiation, we extend the applicability of this technique. In particular, constraint differenti-

ation itself can only be applied to symbolic models and can therefore cannot be directly implemented in the SATMC back-end [1]. These heuristics, however, are more generally applicable and will hopefully allow non-symbolic approaches to benefit from the ideas behind constraint differentiation.

4 Constraining Rule Variables

In the typed model, honest agents can check whether the received messages have the structure prescribed by the protocol. This has the beneficial effect of drastically reducing the number of legal transitions in the typed model. However this check is limited by the existence of expressions encrypted with keys unknown to the agent receiving the message. For instance, in the Kao-Chow protocol:

```
1. A -> S : A, B, Na
2. S -> B : {A,B,Na,Kab}Kas, {A,B,Na,Kab}Kbs
3. B -> A : {A,B,Na,Kab}Kas, {Na}Kab, Nb
4. A -> B : {Nb}Kab
```

`B` cannot decrypt the first part of message `(2)` since he does not know the symmetric key `Kas` shared between `A` and `S`. Hence, `B` cannot check that the occurrences of `A`, `B`, `Na`, and `Kab` in the first part of the message (namely `{A,B,Na,Kab}Kas`) are equal to those in the second part (namely `{A,B,Na,Kab}Kbs`). As a matter of fact, there might be different expressions at these positions.

During the analysis of the protocols, the *constraining rule variables* (CRV) heuristics amounts to giving precedence to those transitions that correspond to the exchange of messages having the form expected by the intended recipients. In our example, the heuristics would thus postpone all the transitions

```
2. S -> B : {A1,B1,Na1,Kab1}Kas, {A,B,Na,Kab}Kbs
```

with `A1,B1,Na1,Kab1` different from `A,B,Na,Kab`. The rationale for this heuristics is that often those messages that cannot be analysed by an honest agent (and are then simply forwarded) are later rejected by another agent and hence they are unlikely to lead to a successful attack. In the above example it is `A` that later analyses and rejects the message forwarded by `B`.

This heuristics can be applied to a generic IF specification and therefore can be exploited by all back-ends. Preliminary experiments carried out with SATMC show the effectiveness of the heuristics. Given a security problem, SATMC simplifies it according to the heuristics. The corresponding propositional encoding is usually much smaller than the encoding associated with

Table 2: Performance of the SATMC back-end with and without the heuristics *constraining rule variables*.

Protocol Name	**NoCRV**			**CRV**			**Saving**$(\frac{\textbf{NoCRV}}{\textbf{CRV}})$		
	Facts	**Rules**	**Time**	**Facts**	**Rules**	**Time**	$\frac{\textbf{Facts}}{\textbf{Facts}}$	$\frac{\textbf{Rules}}{\textbf{Rules}}$	$\frac{\textbf{Time}}{\textbf{Time}}$
Andrew Secure RPC Protocol	588	74,756	5266	465	15,650	123.51	1.3	4.8	4.3
Encrypted Key Exchange	1,603	24,134	116.76	1,148	10,924	77.84	1.4	2.2	1.5
ISO CCF 1-pass unilateral authentication	47	24	0.07	39	22	0.04	1.2	1.1	1.8
ISO CCF 2-pass mutual authentication	94	192	0.31	81	132	0.26	1.2	1.5	1.2
ISO public key 1-pass unilateral authentication	87	70	0.14	61	49	0.11	1.4	1.4	1.3
ISO public key 2-pass mutual authentication	162	421	0.76	125	279	0.59	1.3	1.5	1.3
ISO symmetric key 1-pass unilateral authentication	47	28	0.07	39	25	0.05	1.2	1.1	1.4
ISO symmetric key 2-pass mutual authentication	113	105	0.40	100	87	0.37	1.1	1.2	1.1
Kao Chow Repeated Authentication, 1	5,378	453,865	MO	2,753	2,042	8.74	2.0	222.3	-
Kao Chow Repeated Authentication, 2	66,238	MO	MO	32,963	22,344	156.09	2.0	-	-
Kao Chow Repeated Authentication, 3	92,270	MO	MO	49,378	55,727	627.19	1.9	-	-
Needham-Schroeder Conventional Key	14,795	78,636	377.74	6,755	5,220	18.42	2.2	15.1	20.5
Needham-Schroeder Public Key	358	875	2.05	298	604	1.62	1.2	1.5	1.3
Needham-Schroeder Public Key with key server	493	937	3.67	429	662	2.99	1.2	1.4	1.2
SPLICE/AS Authentication Protocol	1,510	3114	16.97	822	658	3.73	1.8	4.7	4.5
Neuman Stubblebine repeated part	2,487	135,574	1,483	1,409	2,408	6.49	1.8	56.3	228.9
Davis Swick Private Key Certificates, protocol 1	767	1,544	4.99	496	258	0.89	1.6	6.0	5.6
Davis Swick Private Key Certificates, protocol 2	1,040	648	2.02	693	450	1.79	1.5	1.4	1.1
Davis Swick Private Key Certificates, protocol 3	954	966	1.29	526	482	0.77	1.8	2.0	1.7
Davis Swick Private Key Certificates, protocol 4	2,209	2,493	40.38	1,357	1,283	10.46	1.6	1.9	3.9
Woo-Lam Mutual Authentication	74,723	MO	MO	45,584	27,051	312.18	1.6	-	-

Legenda: MO: Memory Out; -: not available.

the original protocol. Clearly this simplification does not introduce spurious attacks and hence, if an attack is detected, then it is reported to the user. If no attack is found, then this may be due to the simplification and hence SATMC generates and analyses the propositional encoding of the original protocol.

It turns out that the heuristics is really effective and all known attacks to the protocols in the Clark/Jacob library [8] are found by SATMC by analysing the simplified protocols (i.e. resorting to the analysis of the original protocol is never necessary).

It is also worth pointing out that in some cases the application of this heuristics is attack-preserving. The Kao-Chow protocol described above is one of these cases. In fact, before step `(2)`, the agent `A` knows his name, the name of agent `B`, and the number `Na`. As a consequence, when `A` receives the message forwarded from `B` (cf. step `(3)`), the only term that `A` could not check, because it is not in its knowledge, is `Kab`. Notice, however, that together with the forwarded message, `A` receives also `{Na}Kab` that allows him to check the appropriateness of `Kab`.

We have also considered a strengthened version of the heuristics that imposes that fresh terms associated with different variables and occurring in the same message must be different. For instance, in the second message of the NSPK:

```
2. A -> B : {Na,Nb}Kb
```

the heuristics gives precedence to the transitions in which `Na` is different from `Nb`. Even with this strengthened version of the heuristics, all known attacks to the protocols in the Clark/Jacob library are found by SATMC by analysing the simplified protocols only.

We have thoroughly experimented SATMC with this strengthened version of the heuristics on the Clark/Jacob library and the results are shown in Table 2. For each protocol we have built a corresponding security problem (IF instance) modelling a scenario with a bounded number of principals which exchange messages on a channel controlled by the most general intruder based on the Dolev-Yao model [10]. The dimension of the search-space associated with a security problem can be estimated by number of facts and of rules characterising the IF instance. For each protocol we report the total time (**Time**) spent by SATMC to find out the attack, the number of facts (**Facts**) and of rules (**Rules**) in the corresponding security problem with the heuristics enabled (**CRV**) and disabled (**NoCRV**), and a measure of the saving (**Saving**($\frac{\textbf{NoCRV}}{\textbf{CRV}}$)) obtained by applying the heuristics in terms of the rate between the timings ($\frac{\textbf{Time}}{\textbf{Time}}$), the number of facts ($\frac{\textbf{Facts}}{\textbf{Facts}}$), and the number of rules ($\frac{\textbf{Rules}}{\textbf{Rules}}$) with the heuristics disabled and enabled. It is immediate to see

that with the heuristics enabled SATMC is up to two orders of magnitude faster in analysing the IF instances; furthermore the size of IF instances is up to a factor 2.2 smaller with respect to the number of facts and up to 2 orders of magnitude smaller with respect to the number of rules. In conclusion thanks to this heuristics, SATMC can now quickly analyse protocols that could not be analysed before in a reasonable amount of time.

5 Conclusions

As shown by our experimental results, the heuristics that we presented above reduce search and thus speed up the performance of the back-ends of the AVISPA tool. Work is currently underway to define further heuristics and to specify, whenever possible, the heuristics at compile time so that all back-ends can benefit from them. Experimentation on the AVISPA protocol library is also underway.

References

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

[2] AVISPA. Deliverable 2.3: The Intermediate Format. Available at `http://www.avipa-project.org`, 2003.

[3] AVISPA. Deliverable 4.2: Partial-Order Reduction. Available at `http://www.avipa-project.org`, 2003.

[4] AVISPA. Deliverable 4.4: AVISPA tool v.1. Available at `http://www.avipa-project.org`, 2003.

[5] AVISPA. Deliverable 6.1: List of selected problems. Available at `http://www.avipa-project.org`, 2003.

[6] D. Basin, S. Mödersheim, and L. Viganò. An On-The-Fly Model-Checker for Security Protocol Analysis. In E. Snekkenes and D. Gollmann, editors, *Proceedings of ESORICS'03*, LNCS 2808, pages 253–270. Springer-Verlag, 2003. Extended version available as Technical Report 404, ETH Zurich, Dep. of Computer Science, `www.inf.ethz.ch/research/publications/`.

[7] D. Basin, S. Mödersheim, and L. Viganò. Constraint Differentiation: A New Reduction Technique for Constraint-Based Analysis of Security Protocols. In V. Atluri and P. Liu, editors, *Proceedings of CCS'03*, pages 335–344. ACM Press, 2003. Extended version available as Technical Report 405, ETH Zurich, Dep. of Computer Science, `www.inf.ethz.ch/research/publications/`.

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

[9] E. M. Clarke, O. Grumberg, and D. A. Peled. *Model Checking*. The MIT Press, Cambridge, Massachusetts, 1999.

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

[11] S. Edelkamp, S. Leue, and A. Lluch-Lafuente. Directed explicit-state model checking in the validation of communication protocols. Report 00161, Institut für Informatik, Universität Freiburg, 2001.

[12] P. E. Hart, N. J. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, SSC-4(2):100–107, 1968.

[13] R. E. Korf. Depth-first iterative-deepening: an optimal admissible tree search. *Artificial Intelligence*, 27(1):97–109, 1985.

[14] S. Malladi, J. Alves-Foss, and R. B. Heckendorn. On preventing replay attacks on security protocols. In *Proceedings of the International Conference on Security and Management*, pages 77–83, 1992.

[15] D. Peled. Ten Years of Partial Order Reduction. In *Proceedings of CAV 1998*, LNCS 1427, pages 17–28. Springer-Verlag, 1998.

[16] P. Syverson. A taxonomy of replay attacks. In *Proceedings of the 7th IEEE Computer Security Foundations Workshop (CSFW'94)*, pages 187–191. IEEE Computer Society Press, 1994.