Comparing State Spaces in Automatic Security Protocol Verification

Cas Cremers Pascal Lafourcade

Department of Computer Science, ETH Zurich, 8092 Zurich, Switzerland

Abstract

Many tools exist for automatic security protocol verification, and most of them have their own particular language for specifying protocols and properties. Several protocol specification models and security properties have been already formally related to each other. However, there is an important difference between verification tools, which has not been investigated in depth before: the explored state space. Some tools explore all possible behaviors, whereas others explore strict subsets, often by using so-called scenarios. Ignoring such differences can lead to wrong interpretations of the output of a tool. We relate the explored state spaces to each other and find previously unreported differences between the various approaches.
We apply our study of state space relations in a performance comparison of several well-known automatic tools for security protocol verification. We model a set of protocols and their properties as homogeneously as possible for each tool. We analyze the performance of the tools over comparable state spaces. This work enables us to effectively compare these automatic tools, i.e. using the same protocol description and exploring the same state space. We also propose some explanations for our experimental results, leading to a better understanding of the tools.

Keywords: Automatic Verification, Security Protocol, State Space, Performance, Tools

1 Introduction

Cryptographic protocols form an essential ingredient of current network communications. These protocols use cryptographic primitives to ensure secure communications over insecure networks. The networks are insecure in two ways: there can be attackers analyzing or influencing network traffic, and communication partners might be either dishonest or compromised by an attacker. Despite the relatively small size of the protocols it is very difficult to design them correctly, and their analysis is complex. The canonical example is the famous Needham-Schroeder protocol [32], that was proven secure by using a formalism called BAN logic. Seventeen years later G. Lowe [27], found a flaw by using an automatic tool known as Casper/FDR. The flaw was not detected in the original proof because of different assumptions on the intruder model. However, the fact that this new attack had escaped the attention of the experts was an indication of the underestimation of the complexity of protocol analysis. This example has shown that automatic verification is critical for assessing the security of such cryptographic protocols, because humans can not reasonably verify their correctness.

This paper is electronically published in
Electronic Notes in Theoretical Computer Science
URL: `www.elsevier.nl/locate/entcs`

A few years later it was proven that the security problem is in fact undecidable [1, 24] even for restricted cases (see e.g. [19] for a survey). This inherent undecidability is one of the major challenges for automatic verification. For instance, in the tool used by Lowe, undecidability was addressed by restricting the type of protocol behaviours that were explored. The protocol that Lowe investigated has two roles, and these may be performed any number of times, by an unbounded number of agents. Lowe used his tool to check a very specific setup, known as a *scenario*. Instead of exploring all possible protocol behaviours, the protocol model given to the tool considers a very small subset, in which there are only a few instances of each role. Furthermore, the initiator role is always performed by a different agent than the responder role. The attack that Lowe found fits exactly in this scenario. However, if there would have been an attack that requires the intruder to exploit two instances of the responder role, Lowe would not have found this particular attack with the tool. Addressing this problem, he managed to give a manual proof for the repaired version of the protocol, which states that if there exists any attack on that protocol, then there exists an attack within the scenario. Ideally, we would not need such manual proofs, and explore the full state space of the protocol, or at least a significant portion of it.

Since then, many automatic tools based on formal analysis techniques have been presented for the verification of cryptographic protocols [2, 7, 12, 13, 18, 20, 29, 30, 31, 34, 35]. These tools address undecidability in different ways: either by restricting the protocol behaviours similar to the approach used by Lowe, or by using abstraction methods. However the restrictions put on the protocol behaviour are rarely discussed or compared to the related work. Moreover, the tools provide very different mechanisms to implicitly restrict the protocol behaviours, and the relations between these mechanisms has not been investigated yet.

This work started out as an investigation into the relative performance of protocol verification tools. Given the number of security protocol verification tools, it is rather surprising that there exists hardly any comparison studies between different tools. In each tool paper the authors propose some tests about their tools' efficiency, by describing the performance of the tool for a set of protocols. These tests implicitly use behaviour restrictions, which are often not specified and sometimes are designed specifically to include known attacks on the tested protocols. Choosing different restrictions has a very clear impact on the accuracy as well as the performance of verification process. In particular, imposing stronger restrictions on the protocol behaviour implies that fewer behaviours need to be explored, which means that for most tools verification time will be exponentially lower.

Here we address two distinct, but very closely related problems. First, we discuss types of behaviour restriction models used by protocol verification tools, which we will also refer to as the explored *state space*. We show how these different state spaces are related to finding, or missing, attacks on protocols, and show how to match up certain state space types. Second, we use the knowledge gained about state spaces to perform a tool performance case study, where we try to match up the exact restrictions used in each tool, and compare their performance on similar state spaces. This leads to new insights in the performance of the tools.

Related work

To the best of our knowledge, the difference between state spaces in security protocol analysis has not investigated before. However, some work exists on comparing the performance of different security protocol analysis tools.

In [37], a large set of protocols is verified using the Avispa tool suite [2] and timing results are given. As the Avispa tool suite consists of four back end tools, this effectively works as a comparison between these four tools. However, this test does not detail the behavioural restrictions that were used. Furthermore, no conclusions about the results are drawn.

A qualitative comparison between Avispa and Hermes [13] is presented in [25] . Unfortunately this test is not very detailed and leads only to some general advice for users of these two tools.

We conjecture that tests as performed here, have not been done before because of the amount of work involved in setting up comparable test cases, and the amount of detailed knowledge required for using each tool. In fact, we believe that a number of researchers in this field have neglected to compare their work with other tools, at any deeper level than just citing from publications.

Outline

In Section 2 we describe and relate some of the different state spaces considered in protocol analysis. In Section 3 we present our performance comparison experiments. We describe the choice of tools and test setup. We then discuss the results of the analysis before we move to the conclusions and future work in the last section.

2 State spaces in security protocol analysis

2.1 Process model

We first give a high-level description of the protocol verification problem in terms of processes. It is not our intent to go into full detail but only to provide the required knowledge for understanding the state space restrictions in the remainder of this paper. For further details we refer the reader to e.g. [20].

A security protocol is defined as a set of communicating processes, typically called "roles". Any of these roles can be performed any number of times in parallel. A role usually consists of a sequence of send and receive events, and implicit or explicit generation of fresh values such as nonces or session keys. The network is considered to be insecure. This is modeled as a single "intruder" named e and represented by the process q. The intruder can take messages from the network, manipulate them, insert messages into the network, or generate fresh values.

We introduce some notation for describing composition of processes. We write $P||Q$ to denote the parallel composition of processes P and Q. We denote by P^* the parallel composition of any number of P processes, i.e. $P^* = P||P||P||\ldots$. Using this notation, the full protocol system describing a protocol with two roles $(r1, r2)$ and the intruder process is given by $q||r1^*||r2^*$. This system exhibits all possible behaviours of the protocol in presence of an active intruder.

For basic security properties such as authentication and secrecy we can define

behaviours of the system as all possible execution traces. We denote the set of all possible traces of the protocol as *Traces*. If there exists an attack on the abstract protocol, it is represented in this set. Conversely, if no trace in Traces exhibits an attack, there is no attack on the abstract protocol.

2.2 Restricting the state space

Because of undecidability or efficiency concerns, protocol analysis tools usually apply some restrictions on the system and do not explore all elements from the set Traces.

Definition 2.1 Terminology.

- A *run* is a single (possibly partial) instance of a role, performed by an agent (this notion is known under various names in protocol models, e.g. "regular strand", "process", or "thread").
- A *run description* of a protocol with $|R|$ roles is a set of roles. An element of a run description is of the form $r(a_1, a_2, \ldots, a_{|R|})$, where r denotes the role that the run is performing. The parameters a_x denote the assignment of agents a_x to each role x. A *symbolic run description* contains variables, whereas a *concrete run description* does not contain variables.
- A *Scenario* is a multiset of run descriptions. $\mathcal{S}$ denotes the set of all possible scenarios and $\mathcal{S}_c$ the set of *concrete scenarios* in which no variables occur.

Alternatively, one may view a run description as a process that performs a particular role and in which the choice of agents (both the executing agent and the expected communication partners) is explicitly encoded by the parameters. A trace of a protocol is an interleaving of runs and (possibly) intruder events.

Example 2.2 The concrete scenario $\{r1(a, e), r2(a, b)\}$ for the Needham-Schroeder protocol exhibits the man-in-the-middle attack. The first process is an initiator run (denoted by $r1$), performed by a, talking to a compromised agent e. The second process is a responder run ($r2$), performed by b, who thinks he is talking to a. Note that the same attack occurs when we substitute a by b. In fact, the attack exists for the following symbolic scenario $\{r1(X, e), r2(X, Y)\}$. Note that both scenarios above do not cover the following scenario: $\{r1(a, e), r1(a, b)\}$ which corresponds to two runs that are *both* performing the initiator role. In particular, there are traces in the second scenario that are not in the third scenario, and vice versa. Thus, if we evaluate the second scenario, we might get completely different results than with the third scenario.

2.3 State space classes

A restriction on the behaviour of a process model changes its state space. Thus, we can view a protocol behaviour restriction as a representation of exploring an alternative state space for the protocol. We identify a set of possible state spaces representing classes of protocol behaviour restrictions.

Definition 2.3 Let n be an integer, and let s be a scenario.

- *Traces* is the set of all traces (possible executions of the protocol) of any length, and any combination of agents.
- *MaxRuns(n)* is the set of traces with at most n runs.
- *Scenario(s)* is the set of traces with at most the runs defined in the scenario s. Thus, the multiset of runs in each trace are by definition a subset of s.
- *RepScen(s)* is the set of traces built only with runs that are present in s. The runs defined by the scenario s can be executed any number of times. In other words, each run in each trace corresponds to an element of s.

Each restriction above effectively restricts the state space of a process model. When we talk about the correctness of a protocol, we refer to the full set of possible behaviours, denoted by Traces. As we will see in Section 3, most tools do not cover this set.

2.4 Relations between state space restrictions

For the set of scenarios $\mathcal{S}$, we have the following relations:

$$\forall n \in \mathbb{N} : \text{MaxRuns}(n) \subset \text{Traces} \tag{1}$$
$$\forall s \in \mathcal{S} : \text{Scenario}(s) \subset \text{Traces} \tag{2}$$
$$\forall s \in \mathcal{S} : \text{RepScen}(s) \subseteq \text{Traces} \tag{3}$$
$$\exists s \in \mathcal{S} : \text{RepScen}(s) = \text{Traces} \tag{4}$$

Proof. The relations (1), (2) and (3) above are immediate consequences of the definitions, as the left hand sides imply restrictions on the full set Traces. For relation (4) we have that if the scenario s includes all possible process descriptions, the repetition of the processes in s effectively amounts to the full set of behaviours without any restrictions. □

We write $|s|$ to denote the number of elements of the scenario s. Relating the scenario-based approaches to the bounding of runs, we have that:

$$\forall s \in \mathcal{S} : \text{Scenario}(s) \subseteq \text{MaxRuns}(|s|) \tag{5}$$

Observe that if the scenario contains $|s|$ runs, the resulting traces will never contain more, and thus this included in MaxRuns($|s|$).

The next formula expresses that there exist no concrete scenarios that correspond exactly to a MaxRuns trace set.

$$\forall n \in \mathbb{N}^*, s \in \mathcal{S}_c : \text{Scenario}(s) \neq \text{MaxRuns}(n) \tag{6}$$

Proof. For any $n > 0$, MaxRuns(n) contains a trace with n runs. So, it will also contain a trace containing n instances of the first role, and also a trace containing n instances of the second role. To match MaxRuns(n) to Scenario(s), s must also contain exactly n runs. Because we are considering only concrete scenarios, we need to define in s the first case (n times the first role). However, by this definition of s we have excluded the second type of traces with only the second role. □

Furthermore, under the assumption that we have a finite number of agents we have that

$$\forall n \in \mathbb{N}, \exists k : \exists s_1, \ldots, s_k \in \mathcal{S}_c : \bigcup_{i=1}^{k} \text{Scenario}(s_i) = \text{MaxRuns}(n) \tag{7}$$

The last formula expresses that if we have finite agents, we can simply enumerate all possible run descriptions of n runs, and turn them into scenario sets. The result of this formula is that we can match up the trace sets of MaxRuns and Scenario by unfolding. This opens up a way to make the state spaces uniform.

2.5 Generation of uniform state spaces

Starting from a state space described using MaxRuns(n) for an integer n, we generate a set of concrete scenarios that exactly covers the same state space, by using Formula (7). Further parameters involved in the generation of this set of scenarios are the number of roles of the protocol and the number of agents.

Required number of agents
In protocol verification it is common to use two honest agents and a single untrusted (compromised) agent. In general, attacks might require more agents, depending on the protocol under investigation and the exact property one wants to verify. A number of results related to this can be found with proofs in [17]. We recall the results of this paper that we use here:

- Only a single dishonest (compromised) agent e, is enough.
- For the verification of secrecy, only a single honest agent a is sufficient.
- For the verification of authentication, we only need two honest agents a and b.

For example, for a single honest agent a and a single compromised agent e, for a protocol with roles $\{r1, r2\}$, we have that:

$$\text{MaxRuns}(1) = \Big(\bigcup_{k \in \{a,e\}} \text{Scenario}(\{r1(a,k)\}) \Big) \cup \Big(\bigcup_{k \in \{a,e\}} \text{Scenario}(\{r2(k,a)\}) \Big)$$

yielding a set of four scenarios.

Computing the minimal number of concrete scenarios
For a given integer n, we can derive the minimal size of a set M of concrete scenarios, such that $\bigcup_{s \in M} s = \text{MaxRuns}(n)$. This minimal size of M directly corresponds to determining the minimal value of k in Formula (7).

For a single agent (involved in the verification of secrecy), the generation of a set of concrete scenarios is a trivial application of the binomial coefficient. For two agents or more the situation is not so simple. In particular, the generation of the set is complicated by the fact that the scenario sets are considered equivalent up to renaming of the honest agents.

Example 2.4 [Renaming equivalence] Let P be a protocol with a single role $r1$. Consider the state space space MaxRuns(2) for two honest agents a, b. Then, we could generate the following scenario set:

$$\{\ \{r1(a), r1(a)\}\ ,\ \{r1(a), r1(b)\}\ ,\ \{r1(b), r1(b)\}\ \}$$

However, as the names of the honest agents are interchangeable, the last scenario is equivalent up to renaming to the first one. In order to verify security properties, we would need only to consider the first two scenarios.

We generalize this approach by considering $|R|$ roles in the protocol description. we assume that we have two agents a and b and one intruder. Let n be the parameter of MaxRuns(n) for which we want to generate the equivalent set of scenarios. In order to choose a run description $X(a_1, \ldots, a_{|R|})$, we have $|R|$ choices for the role X, two choices for a_1 a or b and 3 possible values: a, b or the attacker for each $a_2, \ldots, a_{|R|}$, and we find there are $2 * |R| * 3^{(|R|-1)}$ different possible run descriptions. Now we have to choose a multiset of n run descriptions among this set of all possible runs descriptions. We use the following formula:

$$\binom{2 * |R| * 3^{(|R|-1)} + n - 1}{n}$$

However, this does not take into account that scenarios are equal up to the renaming of the (honest) agents. For example, we observe that $\{r1(a, b), r2(b, a)\}$ is equivalent to $\{r1(b, a), r2(a, b)\}$ under the renaming $\{a \to b, b \to a\}$.

We now use a group theory result to compute the minimal number of scenarios needed. We just recall that $\binom{n}{k} = \frac{n!}{k!(n-k)!}$ and Burnside's lemma [14].

Lemma 2.5 (Burnside's lemma) *Let G be a finite group that acts on a set X. For each g in G let X^g denote the set of elements in X that are fixed by g. Then the number of orbits, denoted $|X/G|$, is:*

$$|X/G| = \frac{1}{|G|} \sum_{g \in G} |X^g|$$

where $|X|$ denotes the cardinality of the set X.

Thus the number of orbits (a natural number or infinity) is equal to the average number of points fixed by an element of G (which consequently is also a natural number or infinity). A simple proof of this lemma was proposed by Bogard [10].

We have to consider all the renamings and compute the number of scenarios that are stable by this operation. Because we have only two agents, we have only two possible renamings:

(i) $\{a \to a, b \to b\}$ (the trivial renaming)

(ii) $\{a \to b, b \to a\}$

Observe first that $|G| = 2$ because we have two renamings. In the first case, all elements are fixed so we have $\binom{2*|R|3^{(|R|-1)}+n-1}{n}$ possibilities. In the second case,

the fixed elements are multisets of the size n where the terms are associated by two and of their arguments are of the form $a, x_1, \ldots, x_{|R|-1}$ or $b, x_1, \ldots, x_{|R|-1}$. It corresponds to choosing a multiset of $\frac{n}{2}$ elements in a set where the first parameter is fixed, i.e, in a set of cardinality $|R| * 3^{(|R|-1)}$. Notice that if n is odd the second set is empty because in this case there is no way to get a fixed element using the second renaming.

Lemma 2.5 leads to the following formula, where ϵ_n is 0 if n is odd and 1 otherwise. $k(n, |R|)$ is the minimal number of scenarios k for Formula (7).

$$k(n, |R|) = \frac{\binom{2*|R|*3^{(|R|-1)}+n-1}{n} + \epsilon_n \binom{|R|*3^{(|R|-1)}+\frac{n}{2}-1}{\frac{n}{2}}}{2} \tag{8}$$

For instance for a protocol with $|R| = 2$ roles, we have that the set of traces MaxRuns(2) is equal to the union of the trace sets defined by $k(2, 2) = 42$ different scenarios. [1] Note that if we would not have taken the renaming equivalence into account, we would instead generate $\binom{2*2*3^{2-1}+2-1}{2} = 78$ scenarios.

2.6 Practical implications

In general, tools can explore radically different state spaces. With protocol verification, we are looking for two possible results: finding attacks on a protocol or having some assurance of the correctness of the protocol.

If an attack on a protocol is found, any unexplored parts of the state space are often considered of little interest. However, if one is trying to establish a level of assurance for the correctness of a protocol, the explored state space becomes of prime importance. As established in the previous section, even for two honest agents, the simplest protocols already need 42 concrete scenarios to explore exactly all attacks involving two runs.

In many cases, protocol models attempt to increase the coverage of small (i.e. involving few runs) attacks by including a scenario with a high number of runs. This process is very error prone: as an example we mention that in the Avispa modeling [4] of the TLS protocol [33] a large scenario is used in the example files, which covers many small attacks, but not scenarios in which an agent can communicate with itself. As a result, the protocol is deemed correct by the Avispa tools, whereas other tools find an attack. This is a direct result of the fact that the used scenario does not cover all attacks for even a small number of runs. One can discuss the feasibility of such an attack, and argue that an agent would not start a session with herself, but the fact remains that the protocol specification does not exclude this behaviour, and therefore certainly means that the protocol does not meet the security properties for its specification.

When one uses a tool for verification of a protocol one should be aware of the impact the state space choices have on the verification result, in order to avoid getting a false sense of security from a tool.

[1] Interested readers can inspect this scenario by running 'Scenario.py' in the test archive [21].

3 Experiments

In this section we use the state space analysis of Section 2 to perform a comparison between several tools on a set of well-known cryptographic protocols considering the same state space. We first discuss some of the choices made for these experiments, after which we give the results of the tests and discuss them.

3.1 Settings

Tool selection

We have compared tools that we could freely download and for which a Linux command-line version exists. Consequently, we had to exclude some tools. In particular, we cannot confirm the results of Athena [35] and NRL [30] as these tools are not available. We do not compare Hermes [13] because the current version of the tool has only a web interface, which is unsuitable for performance testing. We also do not consider the Murphi [31] tool since the current version does not seem to be compatible with any compiler version we had available.

Currently we have used the most recent versions of the following six tools:

Avispa (Version: 1.1) consists of the following four tools that take the same input language called HLPSL [2]:

- **CL-Atse:** (Version: 2.2-5) Constraint-Logic-based Attack Searcher applies constraint solving with simplification heuristics and redundancy elimination techniques [36].
- **OFMC:** (Version of 2006/02/13) The On-the-Fly Model-Checker employs symbolic techniques to perform protocol falsification as well as bounded verification, by exploring the state space in a demand-driven way. OFMC [2] implements a number of optimizations, including constraint reduction, which can be viewed as a form of partial order reduction [5].
- **Sat-MC:** (Version: 2.1, 3 April 2006) The SAT-based Model-Checker builds a propositional formula encoding all the possible traces (of bounded length) on the protocol and uses a SAT solver [3].
- **TA4SP:** (Version of Avispa 1.1) Tree Automata based on Automatic Approximations for the Analysis of Security Protocols approximates the intruder knowledge by using regular tree languages and rewriting to produce under- and over-approximations [11].

The first three Avispa tools (CL-Atse, OFMC and Sat-MC) take a concrete scenario (as required by the HLPSL language) and consider all traces of Scenario(s).

The last Avispa backend, TA4SP, also takes a HLPSL scenario, but verifies the state space that considers any number of repetitions of the runs defined in the scenario, yielding RepScen(s). TA4SP is based on overapproximations and hence might find false attacks. As no trace is ever reconstructed by TA4SP, the user has no indication of whether the output "attack" corresponds to a false or true attack. Furthermore, there is a "level" parameter that influences whether just to use the

[2] OFMC can also consider symbolic scenarios. However, this feature cannot be used through the common Avispa input language HLPSL, but requires changing the Intermediate Format (IF) description of the protocol. Therefore we have not considered this option here, but will do so in future extensions of this work.

Tools	State spaces	Constraints
CL-Atse,Sat-MC,TA4SP	Scenario(s)	$s \in \mathcal{S}_c$
OFMC,ProVerif	Scenario(s)	$s \in \mathcal{S}$
Scyther	MaxRuns(n), Traces	$n \in \mathbb{N}$

Table 1
State spaces explored by the tools

overapproximation (level = 0), or underapproximations of the overapproximation (level > 0). In the Avispa default setting only level 0 is explored by default, which in our test cases would have resulted in never finding any attacks, finding in 57% of all cases "inconclusive", and in the remaining 43% "correct". For our tests, we start at level 0 and increase the level parameter until it yields a result that is not "inconclusive", or until we hit the time bound. This usage pattern is suggested both by the authors in [11] as well as by the output given by the backend when used from the Avispa tool.

ProVerif: (Version: 1.13pl8) analyzes an unbounded number of runs by using overapproximation and represents protocols by Horn clauses. ProVerif [7,9] accepts two kind of input files: Horn clauses and a subset of the Pi-calculus. For uniformity with the other tools we choose to model protocols in the Pi-calculus.

ProVerif takes a description of a set of processes, where each defined processes can be started any number of times. The tool uses an abstraction of fresh nonce generation, enabling it perform unbounded verification for a class of protocols. When ProVerif is given a protocol description, one of four things can happen. First, the tool can report that the property is false, and will yield an attack trace. Second, the property can be proven correct. Third, the tool reports that the property cannot be proved, for example when a false attack is found. Fourth, the tool might not terminate. It is possible to describe protocols in such a way that RepScen(s) is correctly modeled, resulting in the exploration of Traces. Note that the examples provided with ProVerif in general do not consider Traces.

Scyther: (Version: 1.0-beta6) verifies bounded and unbounded number of runs, using a symbolic analysis with a backwards search based on (partially ordered) patterns [20]. [3] Scyther does not require the input of scenarios. It explores MaxRuns(n) or Traces: in the first case, even for small n, it can often draw conclusions for Traces. In the second case, termination is not guaranteed.

In the default setup Scyther explores MaxRuns(5), is guaranteed to terminate, and one of the following three situations can occur. First, the tool can establish that the property holds for MaxRuns(5) (but not necessarily for Traces). Second, the property is false, and an attack trace is shown. Third, the property can be proven correct for Traces.

A summary of the tools and their respective state spaces is given in Table 1.

In order to compare the tools fairly, we match up the state spaces. As a candidate

[3] Note that one of the authors is the author of the Scyther tool. We have tried to stay as objective as possible. However, given the lack of comparative studies in this area, we feel that performing a thorough comparison like this is better than not doing it.

for a common state space, any unbounded set (Traces, RepScen) is not suitable. Furthermore, as Scenario can be used to simulate MaxRuns, but not the other way around, we choose MaxRuns as the common denominator of the selected tools. We automatically generate for each number of runs n the corresponding input files to perform a fair time comparison over MaxRuns(n). Note that the time measurements for the tools only include their actual running times, and does not include the time needed for the generation of the input files.

Security properties
In the tests we analyze secrecy of nonces and session keys, i.e. is it possible that an intruder learns a secret, as well as authentication, i.e. ensuring that an agent communicates with another agent and not with the intruder. For each of the selected tools, secrecy can be modeled. This is not the case for authentication. TA4SP cannot model authentication at all. Other tools provide support for varying notions of authentication, ranging from several forms of agreement [28] to synchronisation [23]. As an exception, we have not modeled authentication properties for ProVerif although it supports authentication [8]. In the test archive there is a file 'proverif-authentication.txt' which gives some of the underlying reasons for this choice.

Protocol test set
We consider a number of well-known protocols found in the literature [16,15,26,4]. We select a set that can be modeled in all tools, which excludes protocols that use e.g. algebraic properties. We have restricted ourselves to the following four protocols : the famous Needham-Schroeder [32] using public keys, and the corrected version by Lowe [27], EKE [6] which uses symmetric and asymmetric encryption, and finally TLS [33] as an example of a larger protocol.

Final setup details
With respect to verification of multiple properties, we observe that some tools can test multiple properties at once, some cannot, and some stop verification at the first property where an attack is found and therefore do not evaluate all properties. We decide to analyze just one security property at a time. Thus we create appropriate files for each tool to check each security property for each protocol.

Producing all the input files requires intimate knowledge of each tool in order to create comparable protocol descriptions. Guided by our results of the previous section, we automatically construct input files for the right scenarios for each protocol and for each tool. This involves the generation of all concrete scenarios to match the state space of a given number of runs. In such cases, we generate all scenarios (ignoring renaming equivalences) in the first phase, and filter out scenarios that are equivalent under renaming in the second phase. The resulting number of scenarios matches exactly with the theoretical number computed in the previous section. For practical purposes we use a time limit of 30 seconds for each security property considered.

All our tests can be reproduced: all used scripts and models are downloadable from [21], and we have only used freely downloadable tools and well-known proto-

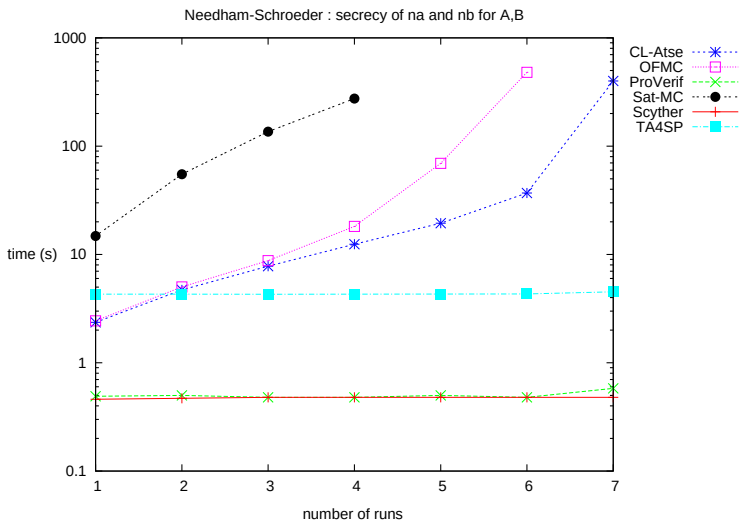

Fig. 1. Time efficiency comparison of the tools on Needham-Schroeder [32], for secrecy

cols. The tests have been performed using an AMD Sempron 3000 processor with 1GB of ram and the Linux operating system.

3.2 Results

We used each tool to verify security properties of the selected protocols. In the first set of results we present the total verification time, which corresponds to the total time needed for verifying all the properties of each protocol, for each tool. In Figure 1, we show the time for verifying the secrecy properties of the Needham-Schroeder protocol as a function of the number of runs n of the explored MaxRuns(n) state space. When a tool reaches the time limit, no conclusion can be drawn about the verification time and hence no points (and connecting lines) are displayed for such cases. For a better visualisation of the exponential behaviour of some of the tools, we use a logarithmic scale.

In the technical report [22] we give further comments, time results for both secrecy and authentication and the graphs obtained for the other protocols.

This time performance analysis shows that overall, the fastest tool is ProVerif, then Scyther, TA4SP, CL-Atse, OFMC and finally Sat-MC.

- ProVerif shows the fastest performance due to abstraction of nonces. This allows the tool to obtain an efficient verification result quickly for an unbounded number of runs.
- Scyther outperforms the other bounded tools, and the fact that full verification can be achieved means that it can often achieve the same performance as tools based on abstraction methods. For some protocols, no full verification can be performed, and the tool exhibits exponential verification time with respect to the number of runs.
- The behaviour of OFMC and CL-Atse is mostly similar for the tested protocols. We observe that the curves produced by these two tools are exponential, which confirms the theoretical results for their underlying algorithms. Finally we

observe that CL-Atse is somewhat faster than OFMC for higher numbers of runs.

- Sat-MC also has an exponential behaviour and is slower than CL-Atse and OFMC for lower numbers of runs. In particular, Sat-MC has proven to be especially slow for the more complex protocol TLS. In contrast, for the smaller protocols we found that Sat-MC starts slower, but its time curve is less steep than that of CL-Atse and OFMC, causing the curves to cross for particular setups. Consequently, for some protocols and a high number of runs Sat-MC can be more efficient than CL-Atse and OFMC, which can be observed in the graphs for the EKE protocol found in the technical report [22].

3.3 Discussion

During these tests we ran into many peculiarities of the selected tools. We highlight the issues we least expected.

In general, the modeling phase has proven to be time-consuming and error-prone, even though we already knew the abstract protocols well. Modeling the protocols in ProVerif took us significantly more time than for the other tools. Furthermore, in the protocol description files provided with the tool, we have not found any modeling of the famous protocol Needham-Schroeder which considers all the possible interactions between the different principals, but only examples which model some particular scenarios.

Avispa has a common input language called HLPSL for the four backend tools. This has the benefit of allowing the user to use different tools based on a single protocol modeling. However, in HLPSL, the link between agents and their keys is usually hard-coded in the scenarios, making the protocol descriptions unsuitable for unbounded verification, as one cannot in general predict how these links should be for further role instances (outside of the given scenario).

Even in this limited test, we found two instances where TA4SP indicates that a property of a protocol is false, where we expected the property to hold (see technical report [22] for details). Because no traces are produced by the tool, we are unable to investigate this further.

We realize that each of the tools has its particular strengths in other features than just performance on simple protocols. However, the focus of this research is clearly on the efficiency only. Furthermore, our tests involve only one particular efficiency measure, and there are certainly other efficiency measures (e.g. mapping all state spaces to symbolic scenarios, and many others) that we expect would lead to slightly different results. However, given that no research has ever been performed before on this topic, we see our tests as a base case for others to work from.

4 Conclusion

We have for the first time analyzed the state space models used in automatic security protocol analysis. The analysis shows the relations between the various models, revealing that protocol analysis tools actually explore by default very different state spaces. This can lead to a false sense of security, because if a tool states that no attack is found, only the explored state space is considered, and other attacks might

have been missed.

Next, we match up the state spaces in order to have different tools explore similar state spaces, where we take into account that traces are considered equal up to renaming of honest agents. This leads to a result relating the number of concrete scenarios needed to cover all traces of a protocol involving a particular number of runs.

We applied these theoretical results to a performance comparison of several automatic protocol verification tools. Rather unexpectedly, such a performance comparison has never been performed before. We modeled four well-known protocols identically for each tool, and verify the protocols using the tools on similar state spaces to obtain a fair performance comparison. The resulting scripts and data analysis programs are available from [21].

The performance results show that ProVerif is the fastest of the tools under consideration for this set of protocols, and that its approximation techniques are effective. Scyther comes in as a very close second, and has the advantage of not using approximations. CL-Atse and OFMC (with concrete sessions) are close to each other, and are the most efficient of the Avispa tools, followed by Sat-MC. For a higher number of runs of simple protocols, it seems that Sat-MC can become more efficient than the two other tools. In some cases TA4SP can complement the other Avispa tools, but in general it is significantly slower than the other tools that can handle unbounded verification (Scyther and ProVerif), and has the added drawback of not being able to show attack traces.

As future work we would like to include some other tools into the analysis, e.g. Casper/FDR [29, 34] CoProVe [18], Hermes [13], STA [12], or the symbolic sessions specification for other tools. To improve the reliability of the tests it would be interesting to investigate a wider range of protocols.

References

[1] R. Amadio and W. Charatonik. On name generation and set-based analysis in the Dolev-Yao model. In *CONCUR'02*, volume 2421 of *LNCS*, pages 499–514, 2002.

[2] A. Armando, D. Basin, Y. Boichut, Y. Chevalier, L. Compagna, J. Cuellar, P. H. Drielsma, P.-C. Heám, O. Kouchnarenko, J. Mantovani, S. Mödersheim, D. von Oheimb, Michael R., J. Santiago, M. Turuani, L. Viganò, and L. Vigneron. The AVISPA tool for the automated validation of internet security protocols and applications. In *Proc. of CAV'2005*, LNCS 3576, pages 281–285. Springer, 2005.

[3] A. Armando and L. Compagna. An optimized intruder model for SAT-based model-checking of security protocols. In A. Armando and L. Viganò, editors, *ENTCS*, volume 125, pages 91–108. Elsevier Science Publishers, March 2005.

[4] AVISPA Project. AVISPA protocol library. Available at `http://www.avispa-project.org/`.

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

[6] S.M. Bellovin and M. Merritt. Encrypted Key Exchange: Password-based protocols secure against dictionary attacks. In *SP '92: Proc. of the 1992 IEEE Symposium on Security and Privacy*, page 72. IEEE Comp. Soc., 1992.

[7] B. Blanchet. An efficient cryptographic protocol verifier based on prolog rules. In *Proc. CSFW'01*, pages 82–96. IEEE Comp. Soc. Press, 2001.

[8] B. Blanchet. From secrecy to authenticity in security protocols. In *Proc. 9th International Static Analysis Symposium (SAS)*, pages 342–359. Springer, 2002.

[9] B. Blanchet. *Cryptographic Protocol Verifier User Manual*, 2004.

[10] K. P. Bogart. An obvious proof of Burnside's Lemma. *Am. Math. Monthly*, 98(12):927–928, 1991.

[11] Y. Boichut, P.-C. Héam, O. Kouchnarenko, and F. Oehl. Improvements on the Genet and Klay technique to automatically verify security protocols. In *Proc. AVIS'04*, April 2004.

[12] M. Boreale. Symbolic trace analysis of cryptographic protocols. In *Proc. 28th International Colloquium on Automata, Languages and Programming (ICALP)*, LNCS. Springer, 2001.

[13] L. Bozga, Y. Lakhnech, and M. Perin. HERMES: An Automatic Tool for Verification of Secrecy in Security Protocols. In *Computer Aided Verification*, 2003.

[14] W. Burnside. *Theory of groups of finite order*. Cambridge University Press., 1897.

[15] M. Burrows, M. Abadi, and R. Needham. A logic of authentication. In *Proc. 12th ACM Symposium on Operating System Principles (SOSP'89)*, pages 1–13. ACM Press, 1989.

[16] J. Clark and J. Jacob. A survey of authentication protocol literature. http://www.cs.york.ac.uk/~jac/papers/drareviewps.ps, 1997.

[17] H. Comon-Lundh and V. Cortier. Security properties: two agents are sufficient. *Science of Computer Programming*, 50(1-3):51–71, 2004.

[18] R. Corin and S. Etalle. An improved constraint-based system for the verification of security protocols. In *Proc. 9th International Static Analysis Symposium (SAS)*, volume 2477 of *LNCS*, pages 326–341. Springer, Sep 2002.

[19] V. Cortier, S. Delaune, and P. Lafourcade. A survey of algebraic properties used in cryptographic protocols. *Journal of Computer Security*, 14(1):1–43, 2006.

[20] C.J.F. Cremers. *Scyther - Semantics and Verification of Security Protocols*. Ph.D. dissertation, Eindhoven University of Technology, 2006.

[21] C.J.F. Cremers and P. Lafourcade. Protocol tool comparison test archive. http://people.inf.ethz.ch/cremersc/downloads/performancetest.tgz.

[22] C.J.F. Cremers and P. Lafourcade. Comparing state spaces in automatic security protocol verification, April 2007. Technical Report 558, Department of Computer Science, ETH Zurich.

[23] C.J.F. Cremers, S. Mauw, and E.P. de Vink. Injective synchronisation: An extension of the authentication hierarchy. *Theoretical Computer Science*, 2006.

[24] N. Durgin, P. Lincoln, J. Mitchell, and A. Scedrov. Undecidability of bounded security protocols. In *Proc. Workshop FMSP'99*, 1999.

[25] M. Hussain and D. Seret. A comparative study of security protocols validation tools: HERMES vs. AVISPA. In *InProc. ICACT'06*, volume 1, pages 303– 308, 2006.

[26] F. Jacquemard. Security protocols open repository. Available at http://www.lsv.ens-cachan.fr/spore/index.html.

[27] G. Lowe. An attack on the Needham-Schroeder public key authentication protocol. *Information Processing Letters*, 56(3):131–133, November 1995.

[28] G. Lowe. A hierarchy of authentication specifications. In *Proc. 10th IEEE Computer Security Foundations Workshop (CSFW)*, pages 31–44. IEEE Computer Society, 1997.

[29] G. Lowe. Casper: a compiler for the analysis of security protocols. *J. Comput. Secur.*, 6(1-2):53–84, 1998.

[30] C. Meadows. Language generation and verification in the NRL protocol analyzer. In *Proc. CSFW'96*, pages 48–62. IEEE Comp. Soc. Press, 1996.

[31] J.C. Mitchell, M. Mitchell, and U. Stern. Automated analysis of cryptographic protocols using Murphi. In *IEEE Symposium on Security and Privacy*, May 1997.

[32] R. Needham and M. Schroeder. Using encryption for authentication in large networks of computers. *Communication of the ACM*, 21(12):993–999, 1978.

[33] L. C. Paulson. Inductive analysis of the internet protocol TLS. *ACM Trans. Inf. Syst. Secur.*, 2(3):332–351, 1999.

[34] A. W. Roscoe. Modelling and verifying key-exchange protocols using CSP and FDR. In *IEEE Symposium on Foundations of Secure Systems*, 1995.

[35] D. Song, S. Berezin, and A. Perrig. Athena: A novel approach to efficient automatic security protocol analysis. *Journal of Computer Security*, 9(1/2):47–74, 2001.

[36] M. Turuani. The CL-Atse protocol analyser. In *Proc. RTA'06*, LNCS, August 2006.

[37] L. Viganò. Automated security protocol analysis with the AVISPA tool. *ENTCS*, 155:61–86, 2006.