Experimental Comparison of Automatic Tools for the Formal Analysis of Cryptographic Protocols[*]

M. Cheminod[†] I. Cibrario Bertolotti[†] L. Durante[†] R. Sisto[‡] and A. Valenzano[†]
[†]IEIIT-CNR — [‡]Politecnico di Torino
C.so Duca degli Abruzzi, 24
I-10129 Torino (Italy)
{manuel.cheminod, ivan.cibrario, luca.durante, riccardo.sisto, adriano.valenzano}@polito.it

Abstract

The tools for cryptographic protocols analysis based on state exploration are designed to be completely automatic and should carry out their job with a limited amount of computing and storage resources, even when run by users having a limited amount of expertise in the field. This paper compares four tools of this kind to highlight their features and ability to detect bugs under the same experimental conditions. To this purpose, the ability of each tool to detect known flaws in a uniform set of well-known cryptographic protocols has been checked.

1. Introduction

In the last years, researchers devoted much effort to develop techniques to formally analyse cryptographic protocols. Their efforts have been stimulated by a widespread, increasing interest in security issues, and have led to the development of several prototypes of formal analysis tools, able to find out attacks even on well-known protocols which for long had been considered secure.

Tools are usually developed with the aim of implementing and automating analysis techniques, so as to help finding out possible vulnerabilities with minimal effort. Of course, the degree up to which this objective is achieved by each tool may vary, depending on both the different analysis techniques used and the particular implementation choices made.

Unfortunately, up to now not much work has been published on comparing tools for cryptographic protocol analysis: each author provides experimental data about his/her own tool, but the results are often not directly comparable because the sets of protocols used to test different tools are not the same. In [12], four state exploration tools are compared, but this comparison is based only on some of the tool features, and in no way on experimental data. Another work [2] is based on a case study, and compares the use of two languages, namely Haskell and Maude, for modelling and reasoning about cryptographic protocols.

In this paper we compare four fully automatic state exploration tools for cryptographic protocol analysis, focusing on both qualitative and quantitative experimental issues. First, Sect. 2 briefly presents each tool and the kind of analysis performed. Then, in Sect. 3, the results of our experimental comparison of the tools on a suitable set of protocols taken from [16], an online repository

[*] This work was developed in the framework of the EU project IST-026600 DESEREC, "Dependability and Security by Enhanced Reconfigurability".

of security protocols based on the work of Clark and Jacob [8], are presented. The reported results measure both the ability of each tool to find out known vulnerabilities on the test protocols. Section 4 draws some conclusions.

2. The Tools

In this section, several tools for cryptographic protocol analysis based on state exploration are discussed, namely Casper/FDR, STA, S^{3}A, and OFMC.

2.1. Casper/FDR

One of the first tools used to analyse cryptographic protocols is based on the FDR model checker, a tool marketed by Formal Systems [17]. The FDR input consists of a description of the model to be analysed and a specification representing the desired properties. Both the model description and the specification are expressed in a machine-readable dialect of the process algebra CSP [13]. FDR can generate the state spaces of the model and of the specification and check whether the model satisfies the specification, i.e., whether the model is a refinement of the specification.

However, since the task of writing such CSP descriptions is quite difficult and error-prone, a front-end called Casper [14] has been developed, which takes protocol models and security properties expressed in a simpler language and translates them into CSP.

Casper models the intruder according to the Dolev-Yao model [9] and in a way which is completely transparent to the user. The analysis performed by FDR is a classical explicit model checking, and it is not performed on-the-fly, i.e. the checks are executed only after the whole state space has been built. FDR2, the last revision of FDR, has some built-in reductions to limit the size of the state space. No symbolic technique is used to represent messages. In order to keep the model finite, Casper limits the length of messages built by the intruder as well as the number of agents operating in parallel.

2.2. S^{3}A

S^{3}A (Spi calculus Specifications Symbolic Analyser) [11], is a tool that reduces the verification of secrecy and authenticity properties to checks of testing equivalence between specifications. S^{3}A performs such checks automatically, by exhaustive state exploration.

The input language of S^{3}A is a machine readable version of the spi calculus [1], a process algebra that derives from π calculus [15], with some simplifications and the addition of cryptographic primitives. The spi calculus has two basic language elements: terms, to represent data, and processes, to represent behaviours. Terms are elements of a free term algebra and are untyped, for maximum expressive power. The spi calculus process operators let specify the input and output operations carried out by each process, as well as the operations performed on the received data (decomposition and decryption of messages, and equality tests).

Two kinds of security properties can be specified: secrecy and authenticity. To specify secrecy, it is just necessary to specify which terms are expected to be kept secret. The secrecy concept adopted by S^{3}A is stronger than the one normally adopted by other tools, because it is based on testing equivalence and, besides capturing the fact that an intruder must not be able to acquire knowledge of a secret data item, also requires that an intruder must not be able to *infer* anything about it. In other words, this secrecy specification requires that, even if the intruder knows two distinct values

M and M' that a secret data item can assume, it must be unable to distinguish between two sessions where M and M' are transmitted, respectively.

With respect to authenticity, S^{3}A requires two specifications to be written: the first one is the description of the protocol while the second one is a reference specification, in which the authenticity of the messages is enforced. Testing equivalence of the two specifications implies that authenticity holds.

S^{3}A deals with the whole spi calculus, with the only exception of the replication operator, which introduces an unbounded number of processes. Leaving replication out, models are kept finite thanks to symbolic representations of messages. When checking for authenticity, S^{3}A does not work on-the-fly: it first generates the whole state space of the two specifications to be compared and then checks for equivalence. Instead, when checking for secrecy, S^{3}A can work on-the-fly. If the testing equivalence check fails, S^{3}A is capable of synthesising the spi calculus specification of an intruder that can discriminate between the checked specifications, thus possibly leading to an attack. To overcome the issue of state explosion, inherent in exhaustive state exploration methods, S^{3}A exploits state space symmetries and a limited form of partial order.

2.3. STA

STA (Symbolic Trace Analyser) [5, 6] is a model checker for cryptographic protocols relying on symbolic techniques that avoid the explicit construction of the infinite messages that an attacker can send to protocol agents when they carry out an input action.

Also in this case, protocols are described by means of a dialect of the spi calculus [1], but a single public channel over which data are exchanged is assumed, and integers are not supported. In the underlying theory of STA, terms are untyped and can be arbitrarily nested, but all keys must be atomic, although the implementation of STA provides limited support for non-atomic keys.

The intruder is modelled implicitly and conforms to the well-known Dolev-Yao model [9], wit the additional ability (that has to be specified explicitly) for the intruder to assume the role of a legitimate protocol participant.

STA allows to express and verify authentication properties based on correspondence assertions: in any protocol trace a certain action β must follow an action α in the same trace. Moreover, secrecy properties about certain values are verified by means of ad-hoc actions in the specification, designed so as to check that the intruder does not learn secrets at any interaction point between the intruder and the protocol.

The lack of parameterisation in this language leads specification sizes to grow rapidly as more instances of a role are needed.

Roles must be finite in number and behaviour. On the other hand, the symbolic representation of terms allows to replace the infinite set of messages the intruder can send to the legitimate participants on each input action of the protocol with a finite one. Variables provide a finite representation of the infinite set in the first place, and can subsequently be constrained during the course of the analysis, in order to satisfy tests and requirements posed on them by the receiving agents as they check and use them.

In order to simplify the symbolic trace generation system, some symbolic traces may not have, in general, a corresponding concrete one. Thus, when a flaw is detected by the on-the-fly analysis, some kind of refinement is needed to determine if there really exists at least one concrete trace to exploit it.

The analysis stops when a violation is detected, and the protocol steps leading to it are then shown.

2.4. OFMC

OFMC (On-the-Fly Model Checker) [4] is a model-checker for cryptographic protocols relying on *lazy* techniques to reduce the computational effort required to carry out the analysis.

Protocols are described by means of HLPSL (High-Level Protocol Specification Language) that, in an untyped free-term algebra context, supports both symmetric and asymmetric non-atomic keys, one-way functions and inequalities. Moreover, some kind of support is provided for operators with algebraic properties, for example exponentiation. HLPSL specifications are then translated into an intermediate language, IF (Intermediate Format), which is used to carry out the analysis by means of a software tool written in Haskell.

The intruder is modelled implicitly and conforms to the well-known Dolev-Yao model [9], with the additional ability (that has to be specified explicitly) for the intruder to assume the role of a legitimate protocol participant.

OFMC is based on two complementary techniques: the *lazy demand-driven search* and the *lazy intruder*. The former provides a finite representation of an infinite state space, because each portion of the state space is really computed only when it is being analysed.

On the other hand, the lazy intruder technique replaces the infinite set of messages the intruder can send to the legitimate participants on each input action of the protocol by symbolic *variables*. Variables provide a finite representation of the infinite set in the first place, and can subsequently be constrained during the course of the analysis, in order to satisfy tests and requirements posed on them by the receiving agents as they check and use them.

Recent work on OFMC has been focused on the integration of reduction techniques based on partial-order reduction [3], and the eventual introduction of heuristics is foreseen to further improve the efficiency of the analysis.

3. Experimental Results

In order to assess their error-detection capabilities, the tools described in Sect. 2 have been tested on a subset of the security protocols described in [16] and supported by all the tools under test. Namely, we discarded those protocols that were based on the use of either algebraic operators that go beyond the free-term algebra assumption (for example, *exponentiation* or *exclusive or*), or cryptographic algorithms that do not satisfy the *perfect encryption* assumptions. Moreover, we took into account only protocols that either do not use timestamps, or use them in a way trivial enough to consider them as nonces, even if this assumption could weaken the correctness of the analysis to some extent.

For each protocol, the analysis has been limited to the known flaws explicitly mentioned in [4, 8, 16]. However, to make the results meaningful even when looking for undocumented bugs, we wrote the protocol specifications without using any “a priori” knowledge of the bugs and kept them as simple and close to the Alice&Bob-style notation as possible, according to the widely-accepted statement that these tools should require little expertise to be used proficiently. As a consequence, in a small number of cases where we assert that a tool does not detect a flaw, it might indeed be possible to find it by an appropriate, ad-hoc manipulation of the specification.

For example, STA is unable to detect the replay attack in the Denning-Sacco shared key protocol, number 5 in Table 1, unless the intruder’s knowledge is enlarged with messages exchanged in a previous run of the protocol, even though that run is not included in the specification to limit the complexity of the analysis itself. Similarly, S^{3}A and STA are unable to find the type-flaw attacks 15–

Table 1. Error-Detection Capabilities of the Tools, Tested on a Subset of the Security Protocols Open Repository [16]

#	Protocol	Attack Type	S^{3}A	OFMC	STA	Casper
1	Andrew Secure RPC	Freshness	**Y**	**Y**	**Y**	**Y**
2	BAN mod. Andrew Sec. RPC	Parallel session	**Y**	**Y**	**Y**	**Y**
3	BAN concrete Andrew Sec. RPC	Parallel session	**Y**	**Y**	**Y**	**Y**
4	CCITT x509 (3)	Parallel session	**Y**	**Y**	**Y**	**Y**
5	Denning-Sacco shared key	Freshness	**Y**	**Y**	N^h	**Y**
6	Kao Chow authentication 1	Freshness	**Y**	**Y**	**Y**	**Y**
7	KSL (rep. part)	Parallel session	**Y**	**Y**	**Y**	**Y**
8		Parallel session	**Y**	**Y**	**Y**	**Y**
9	KSL	Parallel session	**Y**	N^e	**Y**	N^r
10		Parallel session	**Y**	N^e	N^f	N^r
11	Neumann Stubblebine (rep. part)	Parallel session	**Y**	**Y**	**Y**	**Y**
12	Neumann Stubblebine	Type-flaw	**Y**	**Y**	**Y**	N^h
13	Needham-Schroeder Public Key	Parallel session	**Y**	**Y**	**Y**	**Y**
14	Needham-Schroeder Symmetric Key	Freshness	**Y**	**Y**	**Y**	**Y**
15	Otway Rees	Type-flaw	N^h	**Y**	N^h	N
16		Type-flaw	N^h	**Y**	N^h	N
17		Type-flaw	N^h	N^f	N^h	N
18	SPLICE/AS	Parallel session	**Y**	**Y**	**Y**	N
19		Binding	**Y**	**Y**	**Y**	**Y**
20		Parallel session	N^r	**Y**	**Y**	N^r
21	Hwang/Chen mod. SPLICE/AS	Parallel session	**Y**	**Y**	**Y**	**Y**
22	Clark/Jacob mod. Hwang/Chen	Freshness	N^r	**Y**	N	N^r
23	TMN	Other	**Y**	**Y**	**Y**	**Y**
24		Other	**Y**	**Y**	**Y**	**Y**
25		Parallel session	N^r	N^f	N^f	N
26	Woo/Lam mutual authentication	Parallel session	**Y**	**Y**	**Y**	N
27		Type-flaw	**Y**	**Y**	**Y**	N^h
28	Woo/Lam Π	Parallel session	**Y**	N^f	N^f	N
29		Parallel session	**Y**	**Y**	**Y**	**Y**
30		Parallel session	**Y**	**Y**	**Y**	**Y**
31	Woo/Lam Π^1	Type-flaw	**Y**	**Y**	**Y**	N^h
32	Woo/Lam Π^2	Type-flaw	**Y**	**Y**	**Y**	N^h
33	Woo/Lam Π^3	Type-flaw	**Y**	**Y**	**Y**	N^h
34	Yahalom	Type-flaw	N^h	**Y**	N^h	N
35	BAN simplified Yahalom	Type-flaw	**Y**	**Y**	**Y**	N
36		Parallel session	**Y**	**Y**	**Y**	**Y**
37		Type-flaw	**Y**	**Y**	**Y**	N
	Total		**30**	**32**	**28**	**18**

17 and 34 in Table 1, without forcing a suitable associativity on several tuples, and Casper cannot detect a type-flaw attack unless the types of the data items involved are explicitly enumerated in the specification [10].

Table 1 shows the results of the tests, carried out on a total of 37 distinct flaws and fully documented in [7]. The left-hand side of the table briefly describes each flaw and the affected protocol; in analogy to the classification given in [8], we assume that:

- A *freshness* attack occurs when a message captured by the intruder in a previous protocol session is replayed, possibly as a message component, in the current protocol session.
- A *type-flaw* attack involves the replacement of a message component with another message of a different type by the intruder.
- A *parallel session* attack requires the parallel execution of more than one protocol sessions to be exploited, and the intruder uses messages coming from one session to synthesise messages in the other(s).
- In a *binding* attack, the intruder exploits the protocol's failure to establish a proper binding between a public key and its owner.

When an attack may be assigned to multiple categories, the last matching category in the list prevails. It is also worth noting that several protocols listed in the table are made up of two parts: the initial part, performed once per session, is usually concerned with the generation and exchange of a session key, while the final part, that can be repeated several times, performs a mutual authentication. During the tests, the tools have been used to check both the full-fledged protocol and the mutual authentication part on its own. In the table, the latter case has been marked "rep. part".

On the right-hand side, Table 1 summarises the behaviour of the tools, namely, information is given on whether the tools were successful in discovering the flaw or not, and on any problem they encountered during the analysis.

Table 2 presents the results in aggregate form, by giving their distribution for each tool, and gives more details on the keys being used to represent the test results. In particular, the symbol N^h means that the tool was unable to find a given flaw unless it was given some help by means of a custom specification that somewhat reflected an "a priori" knowledge of the flaw itself, while N^r means that the tool was unable to complete the analysis because it ran out of system resources (more than 2 hours of CPU time, 2 GB of disk or 512 MB of RAM). Finally, the symbol N^e is used when the tool was unable to complete the analysis because either it encountered an internal error, or gave no meaningful result.

The symbol N^f is used only for those protocols affected by multiple flaws and for the tools which stop immediately after the first flaw is detected. It means that the tool was not able find out a flaw because it was "masked off" by another bug that was detected first and caused the tool to stop. In this case indeed, the tool might find the second bug if the specification were amended to fix the one detected first. However, this possibility has not been investigated further.

OFMC appears to be the best tool among those considered, both because it is able to find out the vast majority of known flaws without any help from the user (it has no failures in the N^h class) and because its resource requirements, due to the effectiveness of the lazy evaluation techniques it is based on, are quite small, as remarked by the absence of failures due to resource exhaustion (N^r). The only weakness is its impossibility to fully analyse a protocol with multiple flaws in a single run, because OFMC stops immediately when it finds an attack. From this point of view, tools like S^3A and Casper, that perform an exhaustive enumeration of the state space, behave better and, in fact, have no failures in the N^f class. The other side of the medal is a greater consumption of

Table 2. Distribution of the Test Results by Type

Key	Meaning of the Test Result	S^3A	OFMC	STA	Casper
Y	Flaw detected with a standard specification	**30**	**32**	**28**	**18**
N	Flaw undetected	—	—	1	10
N^h	Flaw detected only with a custom specification	4	—	5	5
N^r	Resources exhausted.	3	—	—	4
N^f	The tool stops when it detects the first flaw	—	3	3	—
N^e	Internal error or no result	—	2	—	—

CPU, memory and disk resources, which leads to several failures belonging to the N^r class. STA requirements in terms of resources are similar to those of S^3A, but STA also stops the analysis when it finds out a flaw.

The limited performance of S^3A can be partially justified if we observe that this tool carries out a testing equivalence check that is more powerful, but also more expensive, than the checks based on reachability analysis; an additional point in favour of S^3A is the more powerful definition of secrecy it adopts.

With respect to type-flaw attacks, both S^3A and STA often need help to describe the associativity of tuples in order to exploit the attack, while OFMC is able to find out type-flaw attacks without any intervention. On the other hand, the design of Casper makes it weak in this area; in fact, most failures in the N^h class as well as several failures in the N class can be attributed to this limitation.

Instead, other failures of Casper are probably due to its lack of symbolic data representation capabilities, that forces the tool to artificially place an upper limit on the length of the messages synthesised by the intruder, thus possibly neglecting a portion of the state space containing an attack trace.

Last, it should be noted that the joint usage of OFMC and S^3A covers the maximum number of flaws, and that flaws detected by STA and Casper are a proper subset of them.

4. Conclusions

The use of analysis and verification automatic tools based on formal methods is becoming more and more pervasive in the community of cryptographic protocols. This paper has presented some results obtained by comparing some popular publicly available tools on an experimental basis on a common ground.

To our knowledge, in fact, no work has still appeared in the literature, that offers the reader results collected by testing different tools on a (quite) large shared protocol basis, as each author usually provides different tests in different conditions for his/her own tool.

Instead, we have tried to set up a fair testing environment for all the tools considered in the paper. This has led to the selection of a suitable set of cryptographic protocols for the analysis and to some choices in the way specifications have been written. With respect to the comparison, attention has been focused on the ability of each tool to discover known flaws.

Obtained results are encouraging and show that the automatic tools we considered can be of significant help in analysing protocols, even though they are still available only in a prototype version.

Finally, we are conscious that our work cannot be considered exhaustive, but rather as a preliminary contribution to a deeper comparative evaluation of automatic tools. In fact, much work has still to be done, for instance to extend the experiments to other tools, to enlarge the protocol basis and to take into account other performance indices. In addition, the ability of the different tools to discover unknown bugs should also be tested and this will be one of the goals of our next researches in this area.

References

[1] M. Abadi and A. D. Gordon. A calculus for cryptographic protocols: The spi calculus. *Inf. Comput.*, 148(1):1–70, 1999. DOI 10.1006/inco.1998.2740.

[2] D. Basin and G. Denker. Maude versus Haskell: an experimental comparison in security protocol analysis. *Electr. Notes Theor. Comput. Sci*, 36, 2000.

[3] D. Basin, S. Mödersheim, and L. Viganò. Constraint differentiation: A new reduction technique for constraint-based analysis of security protocols. In *Proceedings of the 10th ACM Conference on Computer and Communications Security (CCS 2003)*, pages 335–344, New York, 2003. ACM Press.

[4] D. Basin, S. Mödersheim, and L. Viganò. OFMC: A symbolic model checker for security protocols. *Int. J. Inf. Secur.*, 4(3):181–208, 2005. Special issue on ESORICS 2003.

[5] M. Boreale and M. G. Buscemi. Experimenting with STA, a tool for automatic analysis of security protocols. In *Proceedings of the 2002 ACM Symposium on Applied Computing (SAC 2002)*, pages 281–285, New York, 2002. ACM Press.

[6] M. Boreale and M. G. Buscemi. A framework for the analysis of security protocols. In *Proceedings of the 13th International Conference on Concurrency Theory (CONCUR 2002)*, volume 2421 of *Lecture Notes in Computer Science*, pages 483–498, Berlin, 2002. Springer-Verlag.

[7] M. Cheminod. Analisi formale di protocolli crittografici. Master's thesis, Politecnico di Torino, 2005. In italian.

[8] J. Clark and J. Jacob. A survey of authentication protocol literature: Version 1.0. Available online, at `http://www-users.cs.york.ac.uk/~jac/papers/drareview.ps.gz`, 1997.

[9] D. Dolev and A. Yao. On the security of public key protocols. *IEEE Trans. Inf. Theory*, 29(2):198–208, 1983.

[10] 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.

[11] L. Durante, R. Sisto, and A. Valenzano. Automatic testing equivalence verification of spi calculus specifications. *ACM Trans. Softw. Eng. Meth.*, 12(2):222–284, 2003. DOI 10.1145/941566.941570.

[12] Kieran Healy, Tom Coffey, and Reiner Dojen. A comparative analysis of state-space tools for security protocol verification. In *Proceedings of the 3rd WSEAS International Conference on E-Activities*, Crete, Greece, 2004. WSEAS.

[13] C. A. R. Hoare. *Communicating Sequential Processes*. International Series in Computer Science. Prentice Hall, Englewood Cliffs, 1985.

[14] G. Lowe. Casper: a compiler for the analysis of security protocols. In *Proceedings of the 10th IEEE Computer Security Foundations Workshop (CSFW 1997)*, pages 18–30, Washington, 1997. IEEE Computer Society Press. DOI 10.1109/CSFW.1997.596779.

[15] R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes, parts I and II. *Inf. Comput.*, 100(1):1–77, 1992. DOI 10.1016/0890-5401(92)90008-4.

[16] Projet EVA (Explication et Vérification Automatique de protocoles cryptographiques). Security protocols open repository. Available online, at `http://www.lsv.ens-cachan.fr/spore/index.html`, 2003.

[17] A. W. Roscoe. *A Classical Mind, Essays in Honour of C. A. R. Hoare*, chapter Model-checking CSP. International Series in Computer Science. Prentice Hall, Hertfordshire, 1994.