The CL-Atse Protocol Analyser

Mathieu Turuani

The CL-Atse Protocol Analyser

Mathieu Turuani

Loria-INRIA, Vandoeuvre-lès-Nancy, France
turuani@loria.fr

Abstract This paper presents an overview of the CL-Atse tool, an efficient and versatile automatic analyser for the security of cryptographic protocols. CL-Atse takes as input a protocol specified as a set of rewriting rules (IF format, produced by the AVISPA compiler), and uses rewriting and constraint solving techniques to model all reachable states of the participants and decide if an attack exists w.r.t. the Dolev-Yao intruder. Any state-based security property can be modelled (like secrecy, authentication, fairness, etc...), and the algebraic properties of operators like xor or exponentiation are taken into account with much less limitations than other tools, thanks to a complete modular unification algorithm. Also, useful constraints like typing, inequalities, or shared sets of knowledge (with set operations like removes, negative tests, etc...) can also be analysed.

1 Introduction

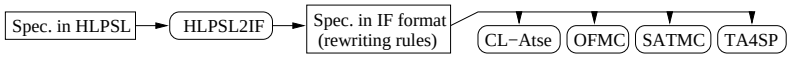
Designing secure communication systems in open environments such as the Internet is a challenging task, which heavily relies on cryptographic protocols. However, severe attacks have been discovered on protocols even assuming perfect cryptographic primitives. Also, a complete manual analysis of a security protocol is usually a very difficult work. Therefore, many decision procedures have been proposed to decide security properties of protocols w.r.t. a bounded number of sessions [1,7,16,15] in the so called Dolev-Yao model of intruder [13], the dominating formal security model in this line of research (see [14] for an overview of the early history of protocol analysis). In particular, among the different approaches the symbolic ones [15,10,12] have proved to be very effective on standard benchmarks [11] and discovered new flaws on several protocols.

The main design goals of CL-Atse[1] are modularity and performance. These two features proved crucial for i) easily extending the class of protocols that can be analysed (modularity) and ii) obtaining results for a large number of protocol sessions (performance). This appeared to be very useful for analysing protocols from the AVISPA [2] project in which CL-Atse is involved since a few years (with OFMC [5], SATMC [3] and TA4SP [6]), as well as for the RNTL Prouvé project that CL-Atse joined recently. The CL-Atse tool can be freely used, either by binary download on the CL-Atse web page[2], or through on-line execution on the AVISPA web page[3].

[1] CL-Atse stands for Constraint-Logic-based ATtack SEarcher.
[2] http://www.loria.fr/equipes/cassis/softwares/AtSe/
[3] http://www.avispa-project.org/web-interface/

The protocol analysis methods of CL-Atse have their roots in the generic knowledge deduction rules from casrul [10] and AVISPA. However, a lot of optimisations and major extensions have been integrated in the tool, like preprocessing of the protocol specifications of extensions to manage the algebraic properties of operators like xor[4] or exponentiation. In practice, the main characteristics of CL-Atse are:

– A general protocol language: CL-Atse can analyse any protocol specified as a set of IF rewriting rules (no restriction). The following figure shows the standard process of protocol analysis using the AVISPA tools, from a specification in HLPSL (role-based, same idea as strands) to any of the four tools available at the moment.



– Flexibility and modularity: CL-Atse structure allows easy integration of new deduction rules and operator properties. In particular, CL-Atse integrates an optimised version of the well-known Baader & Schulz unification algorithm [4], with modules for xor, exponentiation, and associative pairing. To our knowledge, CL-Atse is the only protocol analysis tool that includes complete unification algorithms for xor and exponentiation, with no limitation on terms or intruder operations.
– Efficiency: CL-Atse takes advantage of many optimisations, like simplification and re-writing of the input specification, or optimisations of the analysis method.
– Expressive language for security goals: CL-Atse can analyse any user-defined state-based property specified in AVISPA IF format.

Since protocol security is undecidable for unbounded number of sessions, the analysis is restricted to a fixed but arbitrary large number of sessions (or loops, specified by the user). Other tools provide different features. The closest to CL-Atse are:

The OFMC tool [5], also part of AVISPA, solves the same problem as CL-Atse except that loops and sessions are iterated indefinitely. However, OFMC proposes a different method to manage algebraic properties of operators: instead of hard-coding these properties in the tool, a language of operator properties is provided to the user. Equality modulo theories is solved through modular rewriting instead of direct unification with state-of-the-art algorithms for CL-Atse. However, since this language covers all theories, termination is only obtained by specifying bounds on message depths and number of intruder operations used to create new terms. Hence, completeness cannot be ensured. CL-Atse does not provide such flexibility on properties, but it also does not have any limitation for the theories it can handle (xor, exponentiation, etc...). Moreover, thanks to modularity in the unification algorithm and in knowledge deduction rules, it is quite easy to include new algebraic (or cryptographic) properties directly in the tool. Also, CL-Atse seems to be much faster than OFMC (see Section 3.3).

The Corin-Etalle [12] constraint-based system, which improves upon one developed by Millen & Schmatikov, relies on an expressive syntax based on strands and some efficient semantics to analyse and validate security protocols. Here, strands are extended to allow any agent to perform explicit checks (i.e. equality test over terms). This makes a quite expressive syntax for modelling protocols, that is however subsumed by IF rules. Moreover, to our knowledge no implementation for xor and exponential is provided.

[4] We specially thank Max Tuengal who largely contributed to the integration of xor in CL-Atse.

2 The internal of CL-Atse

We now describe how CL-Atse models protocols and states, and how these objects are used in analysis methods. We start with term signature in CL-Atse used to model messages sent by parties (honest or malicious):

$$\begin{aligned} \mathcal{T}erm &= \mathcal{A}tom \,|\, \mathcal{V}ar \,|\, \mathcal{T}erm.\mathcal{T}erm \,|\, \{\mathcal{T}erm\}^{s}_{\mathcal{T}erm} \,|\, \{\mathcal{T}erm\}^{a}_{\mathcal{T}erm} \\ &\quad |\, inv(\mathcal{T}erm) \,|\, \mathcal{T}erm \oplus \mathcal{T}erm \,|\, Exp(\mathcal{T}erm, \mathcal{P}roduct) \\ \mathcal{P}roduct &= (\mathcal{T}erm)^{\pm 1} \,|\, (\mathcal{T}erm)^{\pm 1} \times \mathcal{P}roduct \end{aligned}$$

Terms can be atoms, variables, concatenations (or pairing), and symmetric or asymmetric encryption (marked by s or a). Also, $inv(k)$ is the inverse of k for asymmetric encryption[5]. The $\oplus$ and $Exp(..)$ operators are presented in Section 3.1, and model the xor and exponentiation operators.

The intruder capabilities in CL-Atse match the Dolev-Yao model [13], extended for xor and exponentiation as in [8,9]. Following the formalism of [16,8,9], we write $Forge(E)$ for the infinite set of messages that the intruder can generate from a set of ground terms E. In particular, the intruder can compose pairs, encryption, xor and exponentiation terms, and decompose pairs, encryption (if possible), etc...

As usual, (ground) substitutions are (ground) term assignments to variables. See [9] for a discussion about how to rewrite a protocol specification to avoid products as variable values. Moreover, allowing agents to make tests of quadratic residuosity for the exponential is an easy extension of CL-Atse planned for near future.

2.1 Protocol and System state in CL-Atse

For performance issues, various algorithms are implemented in CL-Atse to simplify and optimise the input protocol specification, and also to guide the protocol analysis. However, these methods require working on a protocol specification with some special features. Listing these would be quite technical, but the most important ones are the fact that all protocol steps and roles must be local to only one participant, and that CL-Atse must eliminate all honest agents' knowledge by converting them into a small set of equality and inequality constraints over terms with global variables. This allows CL-Atse to compute closures of the participant's or intruder knowledge, unforgeable terms, sets or facts, and to optimise each role instance accordingly (preprocessing). The way CL-Atse converts an IF file is out of the scope of this paper.

An execution trace in CL-Atse is built over (protocol) steps and states, and represents the list of state changes when running a list of steps, starting from the initial state. The basic objects used by CL-Atse are defined as follows.

A system state in CL-Atse is a symbolic representation of an infinite number of "real" (i.e. ground) states. Since honest agent's states have been converted into constraints, only the intruder state is relevant in the definition of states, here. Formally:

$$\begin{aligned} state &= Subst,\, Sets,\, ToDec,\, Known \\ ToDec &= (\mathcal{T}erm, \mathcal{T}erm)^{*} \\ Known &= H(\mathcal{V}ar) \triangleright Known \,|\, D(\mathcal{T}erm) \triangleright Known \,|\, \epsilon \end{aligned}$$

[5] If k is a (random) term, then $inv(k)$ exists but is unknown to every agent.

with $Subst$ a (partial) substitution, $Sets$ a list of facts $t \in set$, with $\{t, set\} \subset \mathcal{T}erm$, saying that in this state the element t is present in the set *named* set, and $ToDec$ a list of opportunities of knowledge deductions: if $(m, k) \in ToDec$, then the intruder will get m as soon as he will be able to forge k. Typically, a knowledge $\{m\}_k^s$ creates an entry in $ToDec$ if k in not known at the point $\{m\}_k^s$ is obtained. Finally, $Known$ is a list of elementary 'D'ecomposed knowledge $D(t)$, and 'H'ypothesis $H(v)$ (i.e. variable constraints), ordered by creation time in the execution trace. For example:

$$Known = H(x) \triangleright H(y) \triangleright D(\{z\}_k^s) \triangleright H(z) \triangleright D(a) \triangleright D(b) \triangleright \epsilon$$

means that the intruder knows $\{z\}_k^s, a, b$ but must forge the value of z from $\{a, b\}$, and the values of x and y from $\{\{z\}_k^s, a, b\}$. We denote $E_{|D}$ the set of terms t such that $D(t) \in E$. Naturally, a symbolic state as above models the infinite set of ground states $\sigma(Known_{|D}, Sets)$ such that σ is a ground instance of $Subst$ and $\sigma(v) \in Forge(\sigma(F_{|D}))$, with $Known = E \triangleright H(v) \triangleright F$. The analysis methods of CL-Atse use rewriting of symbolic states in order to filter or update the set of ground states that it represents.

A protocol step in CL-Atse represents an elementary reaction of an agent: when receiving the message rcv, and provided that a list $CtrList$ of constraints $u = v$ or $u \neq v$ over terms and a list $SetTests$ of constraints $t \in set$ or $t \notin set$ over sets are satisfied, the agent sends a message snd as a response and executes a list $SetOperations$ of add or remove operations over sets and set elements. That is:

$$\begin{aligned} step = {} & iknows(rcv) \,\&\, CtrList \,\&\, SetTests \\ & \Rightarrow iknows(snd) \,\&\, SetOperations \end{aligned}$$

Note that IF facts are converted into constraints over sets. The semantic of ground step execution is defined as usual: given some intruder knowledge E, a populated list of named sets, and a ground substitution σ, if $\sigma(rcv) \in Forge(E)$, if $\sigma(u) = \sigma(v)$ or $\sigma(s) \neq \sigma(t)$ for any constraint $u = v$ or $s \neq t$ in $CtrList$, and if $\sigma(t) \in \sigma(set)$ (resp. $\sigma(t) \notin \sigma(set)$) for any test $t \in set$ (resp. $t \notin set$) in $SetTests$, then $\sigma(snd)$ is added to E and all add or remove operations in $SetOperations$ are performed modulo σ.

A role in CL-Atse is a tree-structured set of roles that captures the non-determinism of the execution of IF rules. Formally:

$$role \;=\; Step(step, role) \,|\, Choice(role\, list) \,|\, EndRole$$

where $Choice$ describes an agent's choice point, i.e. from that point only one role in $role\, list$ may be run, like in $A|B$. Thanks to equality and inequality constraints, this may model pattern matching. Moreover, thread creation is supported through tokens. For example, $A.(B||C)$ is modeled by 3 roles A, B, C where A send tok_1 at its last step and B and C wait for tok_1 in their first steps. Same for confluence ($(B||C).D$), with a pool of tokens.

A security property in CL-Atse is modeled as the negation of a list of attack states, defined as follows:

$$attack\_states \;=\; (iknows(rcv) \,\&\, CtrList \,\&\, SetTests)^*$$

with the same definitions as for $step$. An attack is found when at least one ground state of a symbolic system state satisfies the constraints of one of the attack states. This definition of security failure is quite versatile since it allows the user to use any IF facts (self-made or not) to define any property based on states adapted to his protocol. Standard properties like secrecy or authentication are naturally supported and an implementation of temporal security properties is planned for very near future. For example, fairness[6] in a two-party contract-signing protocol may be coded by:

$$fairness\_atk = (i, Alice, \text{text}) \in play\_together \,\&\, iknows(ctr(\text{text})) \\ \&\, ctr(\text{text}) \notin ctr\_list(Alice) \,\&\, Alice \in finished$$

with text a contract text, $ctr(\text{text})$ a term representing the valid signed contract, and $play\_together$, $ctr\_list(Alice)$ and $finished$ user-defined facts representing the lists of initiated sessions, contracts of $Alice$, and terminated agent's roles. Similar for $Alice$ playing with an honest agent.

A protocol specification in CL-Atse is simply a set of instantiated roles (one for each participant) plus an initial state and a set of attack states:

$$protocol \;=\; RoleList, InitState, attack\_state$$

2.2 Protocol Simplifications & Optimisations

During the AVISPA project, it became increasingly clearer that two important ingredients that may contribute to the efficiency of the CL-Atse tool would be protocol simplification strategies and optimisation operations on the protocol specification. Therefore, without neglecting the importance of efficiency for the analysis algorithm, some important efforts were devoted to the two axes of protocol simplification and optimisation.

Protocol simplifications reduce the overall size of the protocol, and specifically the number of steps, by merging as many steps together as possible, or at least marking them to be executed as soon, or as late, as possible. A step marked to be run as soon as possible will be run in any trace immediately after its parent step. Since these marks or merges put very restrictive constraints on the step interleaving, they greatly reduce CL-Atse computing time (the analysis is necessarily exponential in the number of unmarked steps). However, CL-Atse can only take such decisions when it can automatically build a proof that it would not void the insecurity of the protocol, i.e., that if the protocol was flawed then necessarily at least one attack remains. To do so, CL-Atse builds various protocol-dependant objects like a set of unforgeable terms (atoms, keys, etc.. that the intruder cannot create in any execution). Then, given a protocol step, CL-Atse tests its elements for possibilities of merging (or marking).

To do so, set tests and operations are checked for possibilities of being executed as soon, or as late, as possible. For example, if set is a set name unforgeable by the intruder, then an operation that removes the term t from set can be performed as soon as possible when either there exists no set operation that add t' to set' or tests $t' \in set'$ in

[6] It intuitively requires that whenever a participant obtains a valid contract, there is a way for it's partner to also obtain one.

any step of other roles that may be run before this operation; or the operation is useless or impossible (similar tests). Similar criteria are evaluated for each set operation or test, and for both as soon, or as late, as possible executions.

Also, all other step elements are tested in similar ways, as well as attack states: for example, marking a step to be run as soon as possible requires that if any attack state is validated, then it is also validated either before the previous step, or after the current step if no constraints could prevent running this step. When all tests are successful, the step is marked and another one is analysed.

Optimisations: Protocol optimisations aim at rewriting some parts of the protocol in order to accelerate the search for attacks. The acceleration can be significant, but the protocol structure can be changed. The idea is to track all possible origins of ciphertexts that the intruder must send but cannot create himself (i.e. necessarily obtained from an agent). By building an exhaustive list of origins for such terms, CL-Atse can reduce the future work of the analysis algorithm by unifying these terms with each of their possible origins and generate choice points accordingly. Analysis acceleration comes from a reduction of possible redundancy in step execution. Moreover, this strategy also fixes the moment when steps holding such cipher terms must be run in the analysis. The same must also be done on the awaited sets that the intruder cannot create himself (same idea). For example, if we have some protocol steps

$$\begin{aligned} step_1 &= iknows(\{m\}_k) \Rightarrow .... \\ step_2 &= ... \Rightarrow iknows(\{m'\}_{k'}) ... \\ step_3 &= ... \Rightarrow iknows(\{m''\}_{k''}) ... \end{aligned}$$

where CL-Atse computes that k is unforgeable by the intruder, and that $step_2$ and $step_3$ are the only origins of $\{m\}_k$, then these steps may be replaced by:

$$\begin{aligned} step_4 &= Choice(\, step_5,\, step_6\,) \\ step_5 &= iknows(token_1)\,\&\, equal(\{m\}_k\,, \{m'\}_{k'}) \Rightarrow .... \\ step_6 &= iknows(token_2)\,\&\, equal(\{m\}_k\,, \{m''\}_{k''}) \Rightarrow .... \\ step_7 &= ... \Rightarrow iknows(\{m'\}_{k'}\,,\, token_1) ... \\ step_8 &= ... \Rightarrow iknows(\{m''\}_{k''}\,,\, token_2) ... \end{aligned}$$

The big difference is that only atoms are now awaited in $step_5$ and $step_6$. This gives us the chance to optimise their execution (when possible) by running these steps immediately as soon as $token_1$ or $token_2$ is added to the intruder knowledge. This strategy allows CL-Atse to analyse rapidly some protocols that it could not analyse otherwise.

3 The Analysis Method

As said before, the analysis algorithm implemented in CL-Atse follows the general ideas developed in the AVISPA Project, that is, to symbolically execute the protocol in any possible step ordering. We saw in the previous section some of the important optimisations of CL-Atse for step interleaving above this generic method. Moreover, in order to perform this exploration of all possible execution traces, the analysis algorithm relies on two major components: a (generic) unification algorithm modulo the properties

of the operators, like xor or exponentiation, that provide all term-specific computations; and the management of states and constraints when running a protocol step. This modular structure allowed us to code the tool extensions required by the AVISPA project (like sets, properties, typing, etc..) in a direct and natural way. We now present the two major components of CL-Atse and the analysis method.

3.1 Modular Unification (with xor and exponentiation)

The unification module provides a (generic) complete unification algorithm modulo the algebraic or cryptographic properties of the CL-Atse operators (encryption, xor, exp, pair, etc..), as well as related algorithms like term purification or normalisation. From a general point of view, the problem that must be decided here is: given a (partial) substitution σ and two terms u and v, generate a list of most general unifiers $Mgu^{\sigma}_{u,v} = \left\{\sigma'_1, .., \sigma'_p\right\}$ of u and v that validate σ, i.e.:

$$\forall \sigma' \in Mgu^{\sigma}_{u,v}, \quad \sigma'(u) = \sigma'(v) \text{ and } \sigma'(\mathcal{V}ar) = \sigma'(\sigma(\mathcal{V}ar))$$

Since mgu(s) are used to generate new system states, a great care must be taken to generate a list of mgu as small as possible. The latest implementation of CL-Atse manages the properties of the xor operator, the exponentiation, and the associative concatenation. To manage these properties, the tool unifies terms thanks to an implementation of an optimised version of the well-known Baader & Schulz unification algorithm [4], which splits the unification problem into smaller unification problems, one for each theory. Therefore, the unification algorithm is very modular, and we consider that it would be reasonably difficult to add new operator properties to the previous ones. Currently, we have:

The xor operator: Denoted $\oplus$, this is an associative ($a \oplus (b \oplus c) = (a \oplus b) \oplus c$) and commutative ($a \oplus b = b \oplus a$) operator equipped with a unit element ($a \oplus 0 = a$) and nilpotent ($a \oplus a = 0$);

The exponentiation: Denoted $Exp(g,\ a)$, it represents g^a in some fixed group of prime order. Also, the product $\times$ on exponents models the multiplication in the corresponding (abelian) multiplicative group. Properties include inverse ($a \times a^{-1} = 1$), commutativity ($a \times b = b \times a$), normalisation ($Exp(Exp(g,\ M),\ N) = Exp(g,\ M \times N)$), ...

The associative concatenation: it represents the basic bit string concatenation, without any header giving the splitting position: in this case, associativity models the chance (or a risk) that an agent will not cut the concatenation correctly when parsing it. Naturally, a non-associative pairing operator is also provided.

3.2 The kernel: running a protocol step

The second foundational element of the protocol analysis is the kernel module, which aims at running a protocol step on a symbolic system state by adding new constraints, reducing them to elementary constraints, testing their validity, etc... All these operations are described as rewriting rules and follow carefully the IF semantics. For performance issues all these rules are directly implemented in the tool as operations on constraints.

Therefore, adding new intruder deduction rules requires to implement them in the tool. However, the recent extensions to algebraic properties proved that the tool is sufficiently modular to make such integration quite easy. In particular, the rewriting rules described below correspond to matching in the tool very precisely.

Hypothesis reductions: We call non-reduced a hypothesis $H(t)$ where t is not a variable. This is the received message of a protocol step. Assume that $s = (E \triangleright H(t) \triangleright F \triangleright \epsilon,\ td,\ set,\ \sigma)$ is a system state where $t \notin \mathcal{V}ar$ and F reduced already. Then, we reduce $H(t)$ depending on t with rewriting rules on $E \triangleright H(t) \triangleright F \triangleright \epsilon$ (and σ). For example:

- $E \triangleright H(u, v) \triangleright F \longrightarrow E \triangleright H(u) \triangleright H(v) \triangleright F$;
- $E \triangleright H(t) \triangleright F \triangleright D(t') \triangleright G \longrightarrow \sigma' \,(\, E \triangleright F \triangleright D(t') \triangleright G \,)$ with $\sigma' \in Mgu^{\sigma}_{t,t'}$;
- $E \triangleright H(\{t\}_k^{s \text{ or } a}) \triangleright F \longrightarrow E \triangleright H(t) \triangleright H(k) \triangleright F$ if k in not unforgeable;

These rules model respectively the creation of a pair, the redirection of a known message, and the creation of a cipher. Also, similar but more complex rules allow us to construct xor or exponentiation terms, by enumerating the possibilities available to an adversary for constructing such terms, like building an xor by combining xor and non-xor terms. If defined, σ' is the new state's substitution. These rules are naturally non-deterministic (create a set of states), and are iterated until all variables are reduced.

Knowledge deductions: We increase $Known$ with $K(t)$, $t \in \mathcal{T}erm$, for new non-decomposed 'K'nowledge (sent message in a protocol step), and $T(t)$ for a knowledge being processed (i.e. 'T'emporary). Reducing an $Known$ containing some $K(t)$ is done in two steps, similar to those for hypothesis. That is, the processing of a new knowledge follows this scheme:

$$... \triangleright K(t) \triangleright ... \xrightarrow{\text{decompose } K(t)} ... \triangleright T(t) \triangleright ... \xrightarrow{\text{analyse } ToDec \text{ with } t} ... \triangleright D(t) \triangleright ...$$

The first set of rules decompose any $K(t)$. For example:

- $E \triangleright K(u, v) \triangleright F \longrightarrow E \triangleright K(u) \triangleright K(v) \triangleright F$; $\quad E \triangleright K(t) \triangleright F \longrightarrow E \triangleright T(t) \triangleright F$;
- $E \triangleright K(\{m\}_k^{s \text{ or } a}) \triangleright F \longrightarrow E \triangleright K(m) \triangleright T(\{m\}_k^{s \text{ or } a}) \triangleright H(k') \triangleright F$ with $k' = inv(k)$ (for asymmetric encryption) or $k' = k$ (for symmetric encryption);

These rules model respectively the decomposition of a pair, the fact that a term may not be decomposable, and the decryption of a cipher. Rules in CL-Atse include various optimisations and variations of the techniques described above (like state filtering depending on key availability, ...). Moreover, rules for $\oplus$ or Exp are also included (to get a from $a \oplus t$, or g from $Exp(g,\ M)$). The second set of rules can analyse $ToDec$ to add or remove deduction opportunities depending on $T(..)$. That is, assuming that $Known = E \triangleright T(t) \triangleright F$, we:

- Add (m, k) to $ToDec$ when $t = \{m\}_k$;
- Remove $\{(m'_i, k'_i)\}_{i \in 1..n}$ from $ToDec$ when we can reduce the hypothesis $H(k'_1) \triangleright .. \triangleright H(k'_n) \triangleright E \triangleright D(t) \triangleright F$ to some G such that $D(t)$ is used at least once for each k'_i, and create a new state with $K(m') \triangleright G$. This is again non-deterministic. Also, create a new state with $E \triangleright D(t) \triangleright F$, in case no k'_i may be computed.

These rules, too, are significantly optimised in CL-Atse. Moreover, the last rule guarantee that we won't ever build k' is a way that has already been tried before, which is critical for tool performance.

Other operations: To run a protocol step, we need to perform other operations on states than the two above, like adding (and validating) new equality or inequality constraints, managing sets, etc.. Since they are quite straightforward and coded in a similar way as the two above, they are not detailed here.

3.3 Search for attacks

Using the previously described kernel module, we are now able to run a protocol step on a system state and get the resulting set of new states. Therefore, we can easily explore all possible runs of a protocol by iteratively running steps in any possible ordering, starting from the initial state. Moreover, we reduce step interleaving by using the step marking described in the simplification and optimisation Section 2.2. Finally, each time a protocol step is run, we test the non-satisfiability of each attack state.

Performances: The analysis algorithm of CL-Atse gives very good performances in practice, as shown in the small benchmark table that follows. Times are computing times of the latest versions (feb. 20, 2006) of OFMC and CL-Atse, and protocol specifications are taken from AVISPA. Note also that (2) is CL-Atse without some optimisations. The "Timeout" for QoS in that case is due to an explosion of the number of states. Both binaries and on-line tool execution are available (see introduction for URLs).

Protocol Name **Alg. theory**	**Result**	**OFMC**	**CL-Atse**	**CL-Atse**$^{(2)}$
ASW - Abort part	Secrecy failure	3.94s	**0.03s**	0.16s
EAP with Archie method	Safe	0.70s	**0.07s**	5.94s
EAP TTLS with CHAP	Safe	1.27s	**0.18s**	0.19s
Fair Zhou-Gollmann	Auth. failure	Timeout	**0.13s**	0.13s
Fair Zhou-Gollmann (fixed)	Safe	7.65s	**4.57s**	5.34
IKEv2 with MAC auth. - **Exp.**	Safe	20.29s	**7.62s**	7.62s
Kerberos, cross-realm ver. - **Exp.**	Safe	5.83s	**0.42s**	0.42s
Kerberos, forwardable tickets - **Exp.**	Safe	15.40s	**0.14s**	0.15s
Purpose Built Keys protocol	Auth. failure	0.35s	**0.00s**	0.00s
PEAP with MS-CHAP auth.	Safe	14.25s	**0.18s**	0.18s
Next Steps In Signaling, QoS	Safe	15.53s	**0.86s**	Timeout
SET - Purchase Request	Secrecy failure	1.17s	**0.14s**	0.15s
Diameter Session Init. Prot.	Safe	1.80s	**0.01s**	0.02s
SPEKE, with strong pwd. - **Exp.**	Safe	2.75s	**0.04s**	0.04s
SSH Transport Layer Prot. - **Exp.**	Safe	33.96s	**2.12s**	2.16s

4 Conclusion

As mentioned before, the analysis algorithm implemented in CL-Atse proposes a solution to the NP-Complete protocol insecurity problem w.r.t. a bounded number of sessions, and with (or without) the algebraic or cryptographic properties of operators, like xor, exponentiation, or associative pairing. The methods of CL-Atse include many important optimisations for step interleaving, either by preprocessing or by optimised data structures and deduction rules. This allows CL-Atse to reduce redundancies and limit

the overall number of elementary actions needed at each step (performance). Moreover, the tool proved to be sufficiently flexible to support major improvements and extensions of the past few years (modularity). For example, extensions to inequalities, set operations, state-based properties, or typing required only little recoding of previous works. Also, while the recent implementation of the Baader & Schulz unification required a significant amount of work, the extension of CL-Atse with new operator properties, like Cipher block chaining, is now largely facilitated, as well as planned extensions to temporal security properties of heuristics for unbounded analysis.

References

1. R. Amadio, D. Lugiez, and V. Vanackère. On the symbolic reduction of processes with cryptographic functions. *Theor. Comput. Sci.*, 290(1):695–740, 2003.
2. The AVISPA Team. The Avispa Tool for the automated validation of internet security protocols and applications. In *Proceedings of CAV 2005, Computer Aided Verification*, LNCS 3576, Springer Verlag.
3. A. Armando, L. Compagna. An Optimized Intruder Model for SAT-based Model-Checking of Security Protocols. In *Proceedings of the Workshop on Automated Reasoning for Security Protocol Analysis (ARSPA 2004)*, ENTCS 125(1):91-108, 2005.
4. F. Baader and K.U. Schulz. Unification in the Union of Disjoint Equational Theories: Combining Decision Procedures. In *Journal of Symbolic Computing*. 21(2): 211-243 (1996).
5. D. Basin, S. Mödersheim, L. Viganò. OFMC: A symbolic model checker for security protocols. In *International Journal of Information Security* 4(3):181–208, 2005.
6. Y. Boichut, P.-C. Héam, O. Kouchnarenko. Automatic Verification of Security Protocols Using Approximations. *INRIA Research Report*, October 2005.
7. M. Boreale. Symbolic trace analysis of cryptographic protocols. In *Proceedings of the 28th ICALP'01*, LNCS 2076, pages 667–681. Springer-Verlag, Berlin, 2001.
8. Y. Chevalier, R. Küsters, M. Rusinowitch, and M. Turuani. An NP decision procedure for protocol insecurity with xor. In *Proceedings of LICS 2003*, 2003.
9. Y. Chevalier, R. Küsters, M. Rusinowitch, and M. Turuani. Deciding the Security of Protocols with Diffie-Hellman Exponentiation and Products in Exponents. In *Proceedings of the Foundations of Software Technology and Theoretical Computer Science (FSTTCS'03)*, LNCS 2914, Springer-Verlag, December 2003.
10. Y. Chevalier and L. Vigneron. A Tool for Lazy Verification of Security Protocols. In *Proceedings of the Automated Software Engineering Conference (ASE'01)*. IEEE CSP, 2001.
11. 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`.
12. R. Corin and S. Etalle. An improved constraint-based system for the verification of security protocols. In *SAS*, LNCS 2477:326–341, Springer-Verlag, 2002.
13. D. Dolev and A.C. Yao. On the Security of Public-Key Protocols. *IEEE Transactions on Information Theory*, 29(2):198–208, 1983.
14. C. Meadows. Open issues in formal methods for cryptographic protocol analysis. In *Proceedings of DISCEX 2000*, pages 237–250. IEEE Computer Society Press, 2000.
15. J. Millen and V. Shmatikov. Symbolic protocol analysis with products and Diffie-Hellman exponentiation. In *Proceedings of the 16th IEEE Computer Security Foundations Workshop (CSFW'03)*, pages 47–61, 2003.
16. M. Rusinowitch and M. Turuani. Protocol Insecurity with Finite Number of Sessions is NP-complete. In *14th IEEE Computer Security Foundations Workshop (CSFW-14)*, pages 174–190, 2001.