AVISPA

www.avispa-project.org

IST-2001-39252

Automated Validation of Internet Security Protocols and Applications

Deliverable 5.2: Infinite-State Model-Checking

Abstract

Protocol verification requires the analysis of an infinite number of configurations and therefore calls for infinite-state model checking techniques. In Deliverable 5.1, we have proposed several sound abstractions for reducing the verification to a finite number of states. In this deliverable, we first introduce a verification algorithm for time-sensitive security protocols. The verification is performed by symbolic exploration of upward closed set of configurations. We then present an extension of the tree automata technique introduced in Deliverable 5.1, which allows us to reduce the number of states generated by the approximation function and therefore handle more protocols from the AVISPA library. Finally, we report on some finer abstractions on nonces in fixed-point computations, which allow us to analyse the ASW contract signing protocol and to point to a new attack on it.

Deliverable details

Deliverable version: *v1.0*
Date of delivery: *23.09.2004*
Classification: *public*

Person-months required: *9*
Due on: *31.08.2004*
Total pages: *30*

Project details

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

Information Society Technologies

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

Contents

1 Introduction

Model checking is today a well-established formal technique for the verification of finite-state systems. The increasing acceptance of model checking by industry is due to its "push-button" appeal, i.e. the promise to allow for fully automatic checking of a program or a system — the model — against a logical specification, typically a formula of some temporal logics. As model checking is based on state-space exploration, the size of a system that can be checked is limited and often only relatively small systems can be verified with a model checker. The question many research groups are currently trying to address is how to apply model checking to large industrial software products. Limitations of model checking by the system size imply that verification is possible only using *abstractions* and/or *compositional* techniques. These techniques allow for the construction of a model whose state space is smaller than the one of the original system, but providing a formal proof of correctness for each abstraction or composition step is usually prohibitively expensive.

Several approaches have been proposed to deal with the problem of *verification of infinite systems*. Among them, we distinguish approaches based on abstraction techniques, symbolic reachability analysis, and techniques based on a variety of well-established logical frameworks such as languages and automata theory [20, 41, 50, 1, 2, 11], logics [8, 23], rewriting systems [18, 17, 25], constraints solving [22, 3].

The techniques that we have formalised and implemented so far in the context of the AVISPA project already support the verification of security protocols whenever the number of sessions between the agents is finite (independent of whether the number of sessions is specified explicitly or symbolically, using the symbolic sessions technique described in Deliverable 5.1 [5]). However, the project also aims at providing methods for verifying protocols even when this restriction is uplifted, i.e. for unbounded numbers of sessions. In general, given a protocol specification P specifying exchange of messages by roles, we are interested in the following decision problem: is the protocol correct for all its instances? If the answer is *yes*, then we say the protocol is *verified*.

Known (un)decidability results. Various authors have tackled the problem of protocol verification. Here, we are only interested in the works in which no "geometrical" restrictions, such as conditions on the occurrence on variables or shape of messages, are imposed. In this case, Even and Goldreich [35], and later Heintze and Tygar [40], have proved that the problem of deciding whether a protocol is correct or not for arbitrary instances is un-

decidable when messages of unbounded size (depth) are exchanged. In [33], Durgin, Lincoln, Mitchell and Scedrov have further shown that the problem is undecidable if the depth of messages is bounded and if there is an infinite number of constants. They have also proved that the problem is DEXPTIME-complete even if the message depth is bounded *and* there is a finite number of constants.

Abstraction. A classical way to circumvent the undecidability barriers is to employ abstraction techniques, which, quoting the seminal work [26], have been developed as "a theory of discrete approximation of the semantics of computer systems mainly applied to the static analysis and verification of software". In the case of security protocols, the semantics of a protocol instance is given as an initial state and a finite set of rewrite rules independent from this initial state. Abstractions in our protocol verification context are thus mappings from the original rewrite rules model into a simpler model such that every protocol flaw (of the original model) is contained in the abstract model. The converse usually does not hold since, due to the simplification, we may have false positives in the abstract model even when the original model is flawless. In Deliverable 5.1 [5], a number of different abstractions that we have been considering in our protocol analysis tools, were presented. More specifically, we have presented abstractions of the nonces and of the intruder knowledge, which have been developed and implemented by the AVISPA group at INRIA, as well as a general approach for designing abstractions, which has been formalised by the group at ETH Zürich (ETHZ).

Symbolic Exploration. By using constraints as a rich assertional language [41], it is possible to represent infinite collection of system states. Constraint operations are then incorporated into the standard fixpoint-based model checking algorithms so as to obtain a verification method applicable to a wide range of systems, e.g. systems with unbounded data variables as in [15, 31], or systems with unboundedly many components as in [13, 28, 29, 32]. In recent years, there have been several applications of constraint-based verification techniques to verification of protocols and, more in general, of reactive systems. However, their application to security protocol verification is largely unexplored.

Organisation. In this deliverable, we report on the development of new techniques for security protocol verification based on the paradigms of abstraction and symbolic exploration. More specifically, we present a symbolic exploration approach developed by UNIGE (Section 2), an approach based

on the tree-automata approximations developed by the CASSIS group at INRIA (Section 3), and abstractions techniques in fixed-point computation developed by ETHZ (Section 4).

2 Symbolic Exploration

Several authentication and key-establishment protocols make use of *time-stamps* to avoid possible replay attacks of malicious intruders. However, time-stamps are often abstracted away in formal models of security protocols (see, e.g., [8, 12, 51]). One reason for the use of this abstraction in verification tools is that all known decidability results for verification of security protocols are given for untimed models (see, e.g., [45, 49]). On the other hand powerful (semi-)automatic theorem provers like Isabelle in [7], Spass in [21], and PVS in [34] have been applied to verify timed dependent security properties. It is thus interesting to develop automatic verification procedures capable to directly deal with time-sensitive protocols and properties.

As a preliminary research study within the AVISPA project, in [30] we have defined a *constraint-based* automatic procedure for proving secrecy and authentication properties for an unbounded number of sessions of a time-dependent security protocol. Our procedure is an extension of the symbolic exploration method devised for the specification language MSR($\mathcal{L}$) of [13] based on a combination of multiset rewriting and constraints. A prototypical implementation has been applied to verify timed authentication properties of protocols like Wide Mouthed Frog [16]. In the following subsections, we will describe the main ideas underlying this approach.

2.1 Time-sensitive Security Protocols in MSR($\mathcal{L}$)

Let us first recall the main definitions of the language MSR($\mathcal{L}$) [13]. We will use $\cdot\centerdot\cdot$ and ϵ as multiset constructors and $\oplus$ to denote multiset union and $\preccurlyeq$ to denote multiset inclusion. Furthermore, we will use $Fv(t)$ to denote the set of free variables of a term/formula t.

An MSR($\mathcal{L}$) specification $\mathcal{S}$ is a tuple $\langle \mathcal{P}, \Sigma, \mathcal{V}, \mathcal{I}, \mathcal{R} \rangle$, where $\mathcal{P}$ is a set of predicate symbols, Σ is a first order signature, $\mathcal{V} = \mathcal{V}_t \cup \mathcal{V}_i$ where $\mathcal{V}_t$ is a set of *term* variables and $\mathcal{V}_i$ is a set of *integer* variables with $\mathcal{V}_t \cap \mathcal{V}_i = \emptyset$, $\mathcal{I}$ is a set of initial configurations, and $\mathcal{R}$ is a finite set of labelled rules. A labelled rule has the form $\alpha : \mathcal{M} \longrightarrow \mathcal{N} : \varphi$, where α is a label, $\mathcal{M}$ and $\mathcal{N}$ are two (possibly empty) multisets of atomic formulae built on $\mathcal{P}$, Σ and $\mathcal{V}$, and φ is a *linear integer constraint* such that $Fv(\varphi) \subseteq (Fv(\mathcal{M} \oplus \mathcal{N}) \cap \mathcal{V}_i)$. Variables ranging over integers are embedded within unary function symbols, called *casting*

symbols, used to indicate the type of the variables. For instance $ts(x)$, where ts is a casting symbol, is used to denote a variable of type "time-stamp".

Under the standard perfect cryptography assumption, time-sensitive security protocols can be specified in MSR($\mathcal{L}$) as follows. Identifiers of honest principals are represented as positive integer numbers. The intruder is represented by 0. The current state of honest principals is represented via atomic formulae like $r_i(id(n), \langle t_1, \ldots, t_n \rangle)$, meaning that the honest agent $n > 0$ has executed the i-th step of the protocol in the role r (and it is ready for the next step); $t_1, \ldots, t_n$ are terms representing its current knowledge. Data will be represented here by means of first order terms like $id(\cdot)$ (principal identifier), $sk(\cdot)$ (secret key), $sk(\cdot, \cdot)$ (shared key), $ts(\cdot)$ (time-stamp), $enc(\cdot, \cdot)$ (encrypted message), and $\langle \cdot, \ldots, \cdot \rangle$ (tuple constructor).

The knowledge of the intruder is represented by a collection of atomic formulae of the shape $mem(t)$ for some term t. The behaviour of the intruder can be described by using a multiset rewriting theory describing the Dolev-Yao model as in [18]. A *message* sent on a directional channel is represented by means of an atomic formula $net(sender, receiver, message)$. Encrypted data is represented as the term $enc(key, t)$, where key is a term like $sk(a, b)$ denoting a key shared between a and b, and t is a term.

To model time-stamps, we use a *global clock* whose current value is represented via an atomic formula $clock(\cdot)$. The value of the clock is non-deterministically updated via the following rule:

$$\textbf{time}: \quad clock(ts(now)) \longrightarrow clock(ts(next)) \ : \ next > now$$

This rule introduces arbitrary *delays* between principals actions. Variables shared between the clock formula and a message can be used to represent generation of a time-stamp.

The behaviour of honest principals is defined by rules of the form

$$\alpha : clock(t) \centerdot fresh(sk(y)) \centerdot r(id(a), k) \centerdot net(id(b), id(a), m) \longrightarrow clock(t') \centerdot fresh(sk(y')) \centerdot r'(id(a), k') \centerdot net(id(a), id(b'), m') \ : \ \varphi$$

representing the following transition: at instant t principal a receives message m apparently from b, updates its state from r to r' and its knowledge from k to k' and, at instant t', sends a new message m'. Freshness of a value k (e.g. a key or a nonce) can be ensured by adding (by conjunction) the constraint $y' > k > y$ to the constraint φ. If we assume that a single principal step requires negligible time, then $t = t' = ts(now)$ for some integer variable now representing the current instant. Parameters in the validity conditions of time-stamps (e.g. delays) can be specified by including atomic formulae working as global memory in the initial protocol configuration.

As a more concrete example of protocol specification in MSR($\mathcal{L}$), the following rule formally describes the *server step* of the Wide Mouthed Frog [16] protocol as specified in [30].

server :
$clock(ts(n)) \,.\, server(id(s)) \,.\, delay(ts(d_1), ts(d_2)).$
$\quad . \, net(id(a), id(s), enc(sk(a,s), \langle r(0), id(b), sk(k), ts(t), ts(st)\rangle))$
$\longrightarrow$
$clock(ts(n)) \,.\, server(id(s)) \,.\, delay(ts(d_1), ts(d_2)).$
$\quad . \, net(id(s), id(b), enc(sk(b,s), \langle r(1), id(a), sk(k), ts(n), ts(st)\rangle)$
$: \; a > 0 \wedge b > 0 \wedge n - d_1 \leq t \wedge t \leq n$

In this rule, the server checks if the time-stamp contained in a message sent by a principal a (containing a secret key and the id of the agent b to whom the key must be delivered) is recent and then forwards the message (tagged with $r(1)$) to b. This test is modelled in a natural way by using the constraint $n - d_1 \leq t \wedge t \leq n$, where d_1 represents a parametric delay; parameters are stored in atomic formula like $delay(ts(d_1), ts(d_2))$ as part of the initial configuration.

Operational Semantics The operational semantics of $\mathcal{S}$ is defined via the rewriting relation $\Rightarrow_{\mathcal{R}}$ defined over *configurations*. A configuration is a multiset of ground atomic formulae like $p(ts(3)) \,.\, q(ts(4), h(5))$. Given two configurations $\mathcal{M}_1$ and $\mathcal{M}_2$, $\mathcal{M}_1 \Rightarrow_{\mathcal{R}} \mathcal{M}_2$ if and only if there exists a configuration $\mathcal{Q}$ such that $\mathcal{M}_1 = \mathcal{N}_1 \oplus \mathcal{Q}$, $\mathcal{M}_2 = \mathcal{N}_2 \oplus \mathcal{Q}$, and $\mathcal{N}_1 \longrightarrow \mathcal{N}_2$ is a ground instance of a rule in $\mathcal{R}$. A ground instance of a rule $\alpha : \mathcal{M} \longrightarrow \mathcal{N} : \varphi$ is obtained by extending a substitution in the set of solutions $Sol(\varphi)$ of the constraint φ to a grounding substitution (i.e. with ground terms in its range) for $Fv(\mathcal{M} \oplus \mathcal{N})$. Given a set of MSR($\mathcal{L}$) configurations S, the *predecessor* operator is defined as: $Pre_{\mathcal{R}}(S) = \{\mathcal{M} \mid \exists \mathcal{M}' \in S. \, \mathcal{M} \Rightarrow_{\mathcal{R}} \mathcal{M}'\}$. A configuration $\mathcal{M}$ is *reachable* if $\mathcal{M}_0 \in Pre^*_{\mathcal{R}}(\{\mathcal{M}\})$ for some $\mathcal{M}_0 \in \mathcal{I}$, where $Pre^*_{\mathcal{R}}$ is the transitive closure of the predecessor relation.

2.2 Verification via Symbolic Exploration

In order to define a symbolic representation of sets of configurations of multiple parallel protocol sessions, we proceed in two steps. Following [12], we first introduce a special term constructor $\sup t$ that can be used to finitely represent all terms containing t as a subterm. Then, we incorporate the resulting *extended terms* in a symbolic representation of collections of protocol sessions. The set of extended terms over the variables $\mathcal{V}$ and the signature

Σ is built by induction as follows: constants in Σ and variables in $\mathcal{V}$ are extended terms; if t is an extended term, $\sup t$ is an extended term; given $\mathbf{t} = t_1, \ldots, t_n$ where t_i is an extended term for $i : 1, \ldots, n$ if f is a symbol of arity n in Σ then $f(\mathbf{t})$ is an extended term. Given a *ground* extended term t, its *denotation* is defined by induction as follows:

- $[\![c]\!] = \{c\}$ if c is a constant;
- $[\![\sup t]\!] = \{s \mid s \text{ is a ground term, t occurs in } s\}$;
- $[\![f(t_1, \ldots, t_n)]\!] = \{f(s_1, \ldots, s_n) \mid s_1 \in [\![t_1]\!], \ldots, s_n \in [\![t_n]\!]\}$.

A *symbolic configuration* is a multiset of atomic formulae defined over extended terms annotated with linear integer constraints and is defined as the formula:

$$p_1(\mathbf{t}_1) \centerdot \cdots \centerdot p_n(\mathbf{t}_n) : \varphi \,,$$

where $p_1, \ldots, p_n$ are predicate symbols, each $\mathbf{t}_i$ is an *extended* term for $i : 1, \ldots, n$, and the satisfiable linear integer constraint φ is such that $Fv(\varphi) \subseteq (Fv(p_1(\mathbf{t}_1) \centerdot \ldots \centerdot p_n(\mathbf{t}_n)) \cap \mathcal{V}_i)$. Let σ denote a substitution and $\sigma_{|X}$ denote the restriction of σ to the set $X \subseteq \mathcal{V}$. The *ground denotation* of $M = A_1 \centerdot \ldots \centerdot A_n : \varphi$ is defined as follows:

$$[\![M]\!] \;=\; \{\, B_1 \centerdot \ldots \centerdot B_n \;\mid\; \exists\, \sigma \text{ grounding for } A_1 \centerdot \ldots \centerdot A_n \,, \; B_i \in [\![A_i\sigma]\!] \text{ for } i : 1, \ldots, n, \text{ and } \sigma_{|Fv(\varphi)} \in Sol(\varphi) \,\}$$

This definition is extended to sets of symbolic configurations in the natural way. Finally, in order to reason about configurations consisting of an arbitrary number of parallel protocol sessions, we define the *upward closed denotation* of a set of symbolic configurations $\mathbf{S}$ as follows:

$$\langle\!\langle \mathbf{S} \rangle\!\rangle = \{\mathcal{N} \mid \exists \mathcal{M} \in [\![\mathbf{S}]\!]. \, \mathcal{M} \preccurlyeq \mathcal{N}\}$$

This extended semantics is particularly useful for locally representing *violations* to properties like secrecy, where, independently from the number of sessions and principals, disclosed secrets are shared between the intruder and a finite number of agents. The use of extended terms gives us a further level of parametrisation, e.g. we can locally require that a piece of data occurs at some depth inside a message. As an example the upward closed denotation of the singleton set containing:

$$p(ts(x)) \centerdot r(ts(y), \sup f(ts(z), \sup ts(x))) \;:\; x > y \wedge y > z$$

contains all configurations $p(ts(v_1)) \centerdot r(ts(v_2), T_1[f(ts(v_3), T_2[ts(v_1)])]) \;\oplus\; Q$ for some $v_1 > v_2 > v_3$, two ground terms with a hole $T_1[\cdot]$ and $T_2[\cdot]$, and some configuration Q. The previous semantics is well-suited for a backward analysis of a protocol model in which we do not impose restrictions on the number of parallel sessions, range of time-stamps and generated nonces.

Symbolic Operations. Contrary to ordinary unification, given a pair of extended terms t and t', the extended unification algorithm of [12] computes the *set* of *maximal general unifiers* $max.g.u.(t, t')$. Actually the algorithm of [12] computes the sets of maximal general unifiers for sets of pairs of extended terms. The interesting case in the extension of the unification algorithm occurs when a term like $\sup t$ must be unified with t': we have to search for a subterm of t' that can be unified with t. For example, $x \mapsto \sup a$ is a $max.g.u.$ for $g(\sup a)$ and $\sup g(x)$. We can avoid bindings between integer variables and terms by restricting the unification algorithm in such a way that it cannot descend the term structure of terms like $ts(\cdot)$. For example, $\sigma = \{x \mapsto a\}$ is a $max.g.u.$ for $\sup a$ and $ts(x)$ only if ts is not a casting symbol. If ts is a casting symbol, we are forced to unify a with $ts(x)$ (we cannot select other subterms), and, thus, unification fails. When combined with constraint satisfiability and simplification, the previous operations can be used to extend the unification algorithm to symbolic configurations (i.e. multisets of atomic formulae defined over extended terms annotated with linear integer constraints). Specifically, let the pair $\langle \sigma, \gamma \rangle$ denote a *maximal constrained unifier* (m.c.u.) for $\mathcal{M} : \varphi$ and $\mathcal{N} : \psi$, if (i) there exists a bijection B between the atomic formulae of $\mathcal{M}$ and $\mathcal{N}$ such that $(p(t), p'(t')) \in B$ iff $p = p' \in \mathcal{P}$; and let $T = \{(t, t') \mid \exists p \in \mathcal{P} \wedge (p(t), p(t')) \in B\}$ denote a set of pairs of extended terms then (ii) $\sigma \in max.g.u.(T)$; and (iii) $\gamma \equiv \varphi \wedge \psi \wedge \sigma_{|\mathcal{V}_i}$ is satisfiable.

Example 1. *Consider the following symbolic configurations:*

$$\begin{aligned} M_1 &= p(ts(x)) \centerdot r(ts(y), \sup f(ts(z), \sup ts(x))) \\ &\quad : x > y \wedge y > z \\ M_2 &= p(ts(u)) \centerdot r(ts(v), f(ts(w), h(g(ts(u))))) \centerdot p(ts(m)) \\ &\quad : u > v \wedge v > w \wedge w > m \end{aligned}$$

Given $\sigma = \{u \mapsto x, v \mapsto y, w \mapsto z\}$ *and* $\gamma \equiv x > y \wedge y > z \wedge x = u \wedge y = v \wedge z = w$, $\langle \sigma, \gamma \rangle$ *is an m.c.u. for* M_1 *and the submultiset obtained by removing* $p(ts(m))$ *from* M_2.

Similarly to unification, we can define a subsumption relation $subs_t$ over extended terms such that $subs_t(t, t')$ implies $[\![t']\!] \subseteq [\![t]\!]$ (see [30] for details). When combined with constraint entailment, extended term subsumption allows us to define a comparison relation $\sqsubseteq$ between symbolic configurations such that $M \sqsubseteq N$ implies $\langle\!\langle N \rangle\!\rangle \subseteq \langle\!\langle M \rangle\!\rangle$. (Again see [30] for details.)

Symbolic Predecessor Operator. Let $\mathbf{S}$ be a set of symbolic configurations such that for any $Fv(M) \cap Fv(N) = \emptyset$ for all $M, N \in \mathbf{S}$. Symbolic

backward exploration is based on the pre-image operator $\mathbf{Pre}_R$ defined over a rule R, and to its natural extension $\mathbf{Pre}_{\mathcal{R}}$ to a set $\mathcal{R}$ of rules. These operators take into account the upward closed semantics of symbolic configurations by applying rewriting rules as follows: submultisets of symbolic configurations are matched against submultisets of right-hand side of rules (we reason modulo any possible *context*).

Namely, given $R = \alpha : \mathcal{A} \longrightarrow \mathcal{B} : \psi$, the operator $\mathbf{Pre}_R$ is defined as follows ($\equiv$ denotes logical equivalence of constraints):

$$\mathbf{Pre}_R(\mathbf{S}) = \left\{ (\mathcal{A}\sigma \oplus \mathcal{Q}\sigma \ : \ \xi) \;\middle|\; \begin{array}{l} \exists M \in \mathbf{S}.\, \exists Q.\, \exists \mathcal{B}'.\, \exists \sigma, \gamma, \xi. \\ M{=}(\mathcal{M} : \varphi), \quad \mathcal{M} = \mathcal{M}' \oplus \mathcal{Q}, \quad \mathcal{B}' \preccurlyeq \mathcal{B}\,, \\ \langle \sigma, \gamma \rangle \text{ is a m.c.u. for } \mathcal{M}' : \varphi \text{ and } (\mathcal{B}' : \psi)\,, \\ \xi \equiv \exists \mathbf{x}.\, \gamma \text{ where } \mathbf{x} = Fv(\gamma) \setminus Fv(\mathcal{A}\sigma \oplus \mathcal{Q}\sigma) \end{array} \right\}$$

Let $\mathcal{R}$ be a set of MSR($\mathcal{L}$) rules and $\mathbf{S}$ a set of symbolic configurations, then we have that $\langle\!\langle \mathbf{Pre}_{\mathcal{R}}(\mathbf{S}) \rangle\!\rangle = Pre_{\mathcal{R}}(\langle\!\langle \mathbf{S} \rangle\!\rangle)$. Let $\mathbf{U}$ be a set of symbolic configurations. Our symbolic operations can be used to build a *symbolic backward reachability graph* $G = \langle \mathbf{N}, \mathbf{E} \rangle$ as follows. The set of nodes $\mathbf{N}$ is initially set to $\mathbf{U}$. At each step, for any $M \in \mathbf{N}$ and $Q \in \mathbf{Pre}_{\mathcal{R}}(\{M\})$ if there exists a *visited* symbolic configurations $O \in \mathbf{N}$ such that $O \sqsubseteq Q$, then we discharge Q and add an edge $O \rhd M$ to $\mathbf{E}$ (where we use $\rhd$ for an arrow denoting an edge). If there are no *visited* symbolic configurations that subsume Q (i.e. Q brings new information), then Q is added to the set of nodes $\mathbf{N}$, and an edge $Q \rhd M$ is added to the graph. If $\mathbf{Pre}_{\mathcal{R}}(\{M\})$ is empty, then the node M has no incoming edges. Also note that if $M \rhd N$, then there exists R such that either $\{M\} \in \mathbf{Pre}_R(\{N\})$ or $\exists P.\, \{P\} \in \mathbf{Pre}_R(\{N\})$ and $M \sqsubseteq P$.

Verification of Secrecy and Authentication. Violations of secrecy properties can often be generated by only looking at one honest principal and at part of the intruder's knowledge; they are denoted by (according to the upward closed semantics) symbolic configurations like:

$$mem(\mathbf{secret}) \,.\, r(id(a), \langle \ldots, \mathbf{secret}, \ldots \rangle) \ : \ true$$

The following property then holds.

Theorem 1 (Secrecy [30]). Let $\mathcal{S} = \langle \mathcal{P}, \Sigma, \mathcal{V}, \mathcal{I}, \mathcal{R} \rangle$ be an MSR($\mathcal{L}$) protocol and intruder specification, and $\mathbf{U}$ be the set of symbolic configurations representing violations to a secrecy property. If the symbolic backward reachability terminates when invoked on $\mathcal{S}$ and $\mathbf{U}$, and the set $\mathbf{N}$ of nodes of the resulting *graph* does not contain any configurations of $\mathcal{I}$ (the set of initial

configurations), then the protocol has no attack to secrecy. The property holds for any number of nonces, principals, and parallel sessions.

Since we work on a *concrete protocol model* and a *symbolic semantics* without approximations (unless we apply widening operators during search), our procedure with an on-the-fly check for the initial configurations can be also used for *falsification*, i.e, to extract finite-length symbolic trajectories that represent attacks.

Let us now consider authentication properties like *(non-injective) agreement* (see [43] for a formal definition). Violations of agreement properties *cannot* be represented as upward closed sets of configurations. They are sets of *bad traces*. However, we can still exploit our symbolic search by using the following idea. We first define the two symbolic configurations:

$$M_I = r_i(id(a), \langle \ldots, id(b), \ldots \rangle) : \varphi \qquad M_R = r'_f(id(b), \langle \ldots, id(a), \ldots \rangle) : \psi$$

$\langle\langle M_I \rangle\rangle$ is the set of configurations containing *at least* the initial state of the initiator a in a session, apparently with b, and $\langle\langle M_R \rangle\rangle$ is the set of configurations containing *at least* the final state of the responder b in a session, apparently with a. We select two generic values a and b for the identifiers of the two principals, and generic nonces and keys for the data we want to observe. Data may appear in M_I and M_R; φ and ψ can be used to define constraints over them. Our verification method is based then on the following property (formulated for the responder).

Theorem 2 (Non-injective agreement [30]). Let $\mathcal{S}$ be an MSR($\mathcal{L}$) protocol and intruder specification, M_I and M_R be as described before. Suppose that the symbolic backward search terminates when invoked on $\mathbf{U} = \{M_R\}$ and $\mathcal{S}$. If for every path $M_0 \rhd \ldots \rhd M_k \rhd M_R$ in the resulting graph going from an initial (symbolic) configuration M_0 to M_R, there exists an i such that $M_I \sqsubseteq M_i$, then non-injective agreement property holds for any number of parallel sessions, principals, and nonces.

Being formulated over a time-sensitive specification logic, the previous properties can be used to verify time-dependent secrecy and authentication properties. In other words, we reason about time *inside* our specification logic.

Symbolic configurations can be used in other interesting ways. As an example, we might be interested in checking whether nested encryption could appear in the messages forged by the honest principals. This property can be used to formally justify the use of specialised theories for the intruder. We can then search for traces leading to $net(x_1, x_2, \sup enc(x_3, \sup enc(x_4, x_5))) :$

true. Furthermore, following [12], when dealing with intruder theories in which we allow arbitrary use of rules like *decryption with the intruder's key* it is often possible to apply dynamic widening operators that approximate sets of terms like $enc(sk(0), \ldots, enc(sk(0), msg)\ldots)$ with $enc(sk(0), \sup msg)$. This kind of accelerations may return over-approximations.

2.3 Prototype Implementation and Experimental Results

By using the term manipulation library, the graph package, and the linear constraint solver provided by Sicstus Prolog, we have extended the symbolic verification procedure defined for verification of *parametric data consistency protocols* in [13] to cope with the new class of protocols, properties, and symbolic data structures defined in [30]. As a practical contribution, in [30] we have presented a detailed analysis of a timed model of the Wide Mouthed Frog protocol of [16]. Among other properties, using our method we have automatically proved *non-injective timed agreement* for the version of the protocol proposed by Lowe in [42]. Our results are obtained for an unbounded protocol model and for *parametric delays* in the validity conditions of time-stamps. Previous analyses of this protocol were based on semi-automated [34] and finite-state verification methods [44]).

The main novelty of the resulting method with respect to existing ones is that it allows us to handle unbound parallelism, complex term structures, time-stamps, and freshness of names both at the level of specification and analysis in a uniform and effective way. Furthermore, the symbolic reachability graph allows us to verify security and authentication properties or to extract potential attacks (in form of a symbolic trace), a feature often not so easy in other verification methods based on theorem proving like TAPS [45].

The current version of the prototype implementation does not scale well on complex protocols. A major engineering of our CLP-based prototype is needed to attain scalability. A considerable amount of work (both at the theoretical and at the implementational level) aimed at devising efficient data structures for storing and manipulating collections of symbolic configurations seems necessary. For this reason integration of the prototype within the AVISPA Tool is considered premature and will be reconsidered in the future.

3 Tree automata approximations

In this section we describe an approach, initiated by Genet and Klay [37], based on the ideas of over-estimating the intruder knowledge by using reg-

ular tree languages. This method allows one to show that some states are unreachable, and hence that the intruder will never be able to know certain terms. Regular tree-languages can be used here to effectively model the knowledge that the intruder might have acquired from previous sessions.

This section describes extensions to this specification method for security protocols based on tree automata. Additional expressiveness is added to enable the treatment of protocols that were out of reach of earlier work. Experimental results for protocols of the AVISPA library are presented.

3.1 Abstraction-based Approximation of the Intruder Knowledge

The main idea of Genet and Klay [37] is to build an automaton whose language represents an over-approximation of the network's configuration with an unbounded number of sessions. The automaton $\mathcal{A}_0$ represents the initial configuration of the network and different actions the intruder is able to do (composition of messages and encryption). A term rewriting system (TRS) $\mathcal{R}$ defines the abilities of the intruder to analyse messages, and each step of the protocol. For instance, in the rewriting rule $l \rightarrow r$, l is the message received, and r the message sent. A term rewriting system $\mathcal{R}$ is said to be *left-linear* if the left-hand side of each rule $l \rightarrow r \in \mathcal{R}$ is linear, i.e., if any variables of $Var(l)$ has exactly one occurrence in l. A term rewriting system $\mathcal{R}$ is said to be *runnable* if it is left-linear (a weaker restriction can be used as described in subsection 3.2) and if for each rule $l \rightarrow r$ the condition $Var(r) \subseteq Var(l)$ holds. Notice that $Var(t)$ is the set of variables appearing in the term t.

Given an initial automaton, a term rewriting system and an approximation function, the tool Timbuk [1] [38] computes an automaton $\mathcal{A}_k$ for recognising an over-approximation of the set of messages that could appear in the network. Let us summarise the algorithm for deriving $\mathcal{A}_k$. Starting from the initial automaton, a sequence of automata $\mathcal{A}_i$ is built incrementally by adding new transitions. At step i Timbuk2 looks for a rule $l \rightarrow r \in \mathcal{R}$ and a substitution σ of variables by states of $\mathcal{A}_i$ such that $l\sigma \rightarrow^*_{\mathcal{A}_i} q$ and $r\sigma \not\rightarrow^*_{\mathcal{A}_i} q$. The notation $t \rightarrow^*_{\mathcal{A}} q$ means that using finitely many transitions of the automaton $\mathcal{A}$, a term t can be reduced to the state q (notice that one requires $Var(r) \subseteq Var(l)$ to ensure $r\sigma$ to be a ground term). Then, the approximation function will extend the current automaton $\mathcal{A}_i$ by new transitions to get a new automaton $\mathcal{A}_{i+1}$ such that $r\sigma \rightarrow^*_{\mathcal{A}_{i+1}} q$. The computation

[1] An `OCAML` tree automata library developed by T. Genet at IRISA-Rennes under GPL. Notice that we use the second version of Timbuk (Timbuk2).

is iterated until it reaches a fixed-point $\mathcal{A}_k$, if it exists.

The language recognised by $\mathcal{A}_k$ represents an over-approximation of the set of messages that could appear in the network. To check a property, its negation is represented by a tree automaton $\mathcal{A}_{neg}$ and the intersection between the two languages $\mathcal{L}(\mathcal{A}_{neg})$ and $\mathcal{L}(\mathcal{A}_k)$ — classically $\mathcal{L}(\mathcal{A})$ denotes the set of terms accepted by $\mathcal{A}$ — is computed. If the intersection is empty then the property is verified, otherwise we cannot conclude.

Previous work on abstractions. We have proposed in [47] an approximation function whose main advantage is that it can be automatically generated (which is not the case of the approximation function devised by Genet and Klay [37]). As explained in [5], abstractions play here an important role in order to enforce the termination of the resulting procedures to automatically generate the approximation function as well as to verify secrecy properties. Moreover, following [24] we only consider two agents to verify secrecy properties.

In [10] an additional expressiveness has been added to enable the treatment of protocols specified in the version of the HLPSL used in the AVISS Project (which was based on Alice&Bob-style notation) that were out of reach in earlier work. To be more precise, the improvements are as follows:

- abstractions of keys and nonces are finer than in [47] (see Deliverable 5.1 [5]),
- intruder knowledge specification,
- goals and differentiation of shared keys and nonces.

All improvements proposed in [10] maintain the conditions given in [37, 47] without modifying the approximation function.

3.2 Symbolic Approximation

In order to be able to analyse the Internet security protocols and applications considered in the project (cf. the AVISPA library [6]), we have completely designed anew the approximation function given in [47], as explained below.

Symbolic variables. The most important new feature is that the approximation function is based on the use of *symbolic variables* whose domains are states (of the current automaton and of the approximation function). Indeed, the idea for limiting the number of states is to reuse the states created during the transition normalisation procedure and the automaton completion. To

be more precise, for a TRS rule $l[x,y,z] \to r[x,y,z]$, we first search for a reduction of $r\sigma$ in $\mathcal{A}_i$ to obtain $r\sigma'$. Second, $r\sigma'$ has to be normalised in order to obtain $r\sigma' \to q$. This normalisation uses always the same state for each TRS rule and each position.

We illustrate the idea by means of the following example. Let $l[x,y,z] \to r[x,y,z]$ be a TRS rule where $r[x,y,z] = f(x,g(y,h(z)))$. The symbolic normalisation $f(x,g(y,h(z))) \to \alpha$ requires the symbolic transition set $\{h(z) \to y_{l\to r,2.1}, g(y,y_{l\to r,2.1}) \to y_{l\to r,2}, f(x,y_{l\to r,2}) \to \alpha\}$. The variable $y_{l\to r,p}$ represents the symbolic state associated to the rule $l \to r$ of the TRS for the position p (of r), and the variable α is the symbolic state on which the term $r = f(x,g(y,h(z)))$ is reduced to. This reduction is explained below:

- $h(z)$: this term represents the subterm of r at position 2.1, and it is reduced to the symbolic state associated $y_{l\to r,2.1}$.
- $g(h(z))$: This term representing the subterm of r at position 2, it is reduced to the symbolic state $y_{l\to r,2}$. Since $h(z)$ is reduced to $y_{l\to r,2.1}$, we need the symbolic transition $g(y,y_{l\to r,2.1}) \to y_{l\to r,2}$.
- $f(x,g(y,h(z)))$: Finally, to reduce $f(x,g(y,h(z)))$ to the symbolic state α, the symbolic transition $f(x,y_{l\to r,2}) \to \alpha$ is added to the symbolic transition set.

For instance, let σ be a substitution such that $\sigma(x) = q_1$, $\sigma(y) = q_2$ and $\sigma(z) = q_3$. Assume $l[q_1,q_2,q_3] \to^*_{\mathcal{A}_i} q$ and $h(q_3) \to_{\mathcal{A}_i} q_4$ is a transition of $\mathcal{A}_i$.

- Given $r\sigma = f(q_1,g(q_2,h(q_3)))$ and $h(q_3) \to_{\mathcal{A}_i} q_4$, for $f(q_1,g(q_2,h(q_3))) \to q$ we first obtain $\{h(q_3) \to q_4, g(q_2,q_4) \to y_{l\to r,2}, f(q_1,y_{l\to r,2}) \to q\}$.
- Next, since $g(q_2,q_4)$ is not a left hand side of a transition of $\mathcal{A}_i$, for the rule $l \to r$ and the position 2 in the term $f(x,g(y,h(z)))$, the state corresponding by default to the symbolic variable $y_{l\to r,2}$ is used.

We have modified the procedure described in Deliverable 5.1 [5], in the above way. Recall that the approximation function was generated by our tool `Is2TiF` and then applied by Timbuk2 as a help to normalise terms which were not recognised by the current automaton. In the current implementation, our approximation function is automatically generated and this generation reuses states provided by Timbuk2 normalisation. Notice that this approximation function is simpler than the previous one. Although this generation needs to interact with Timbuk2, its automatic generation is faster. In fact, the number of states generated is smaller than in the previous approximation function. Moreover the protocol specification used by Genet and Klay [37]

requires an AC-operator U to encode sets. Using the roles allows us to put the end to the combinatorial explosion due to this operator. Let us precise this point by the following example.

Genet and Klay encode "if an agent knows $f(x)$, $g(y)$ and $h(z)$, then he can send message $crypt(f(x), pair(g(y), h(z)))$" using the following rule:

$$U(f(x), U(g(y), h(z))) \rightarrow crypt(f(x), pair(g(y), h(z))),$$

where the symbol U encodes the union of sets. Since the operator U is associative and commutative, a lot of possibilities have to be considered. Indeed, the cases, listed below, have to be handled.

- $U(f(x), U(h(z), g(y))) \rightarrow crypt(f(x), pair(g(y), h(z)))$,
- $U(g(y), U(f(x), h(z))) \rightarrow crypt(f(x), pair(g(y), h(z)))$,
- $U(g(y), U(h(z), f(x))) \rightarrow crypt(f(x), pair(g(y), h(z)))$,
- $U(h(z), U(f(x), g(y))) \rightarrow crypt(f(x), pair(g(y), h(z)))$
- and $U(h(z), U(g(y), f(x))) \rightarrow crypt(f(x), pair(g(y), h(z)))$.

However, since protocol steps are fixed, we do not need to encode unbounded sets: we can encode a set by a list with elements arbitrarily ordered. This is done by introducing roles since the above rule is encoded by $role(f(x), g(y), h(z)) \rightarrow crypt(f(x), pair(g(y), h(z)))$. Consequently, there is only one rule to handle.

Types for the left-linearity problem. In order to apply the symbolic approximation function to more complex protocols than the protocols represented by left-linear term rewriting systems, we propose to introduce variables/constants types that allow us to easily verify a left-linearity criterion.

By definition, a term t can be rewritten into s by a rule $l \rightarrow r$ if there exist a position p of t and a substitution μ from the variables of t by $terms$ such that $t = t_p[l\mu]$ and $s = t_p[r\mu]$. The idea is to substitute variables by states. This way, given the automaton $\mathcal{A}_i$ and the TRS $\mathcal{R}$, at each step i our procedure has to verify the following condition. For each TRS rule $l \rightarrow r$, each state q of $\mathcal{A}_i$, and each substitution μ of the variables of l by terms, if

$$l\mu \rightarrow^*_{\mathcal{A}_i} q$$

then there exists a substitution σ of the variables of l by states such that

$$l\mu \rightarrow^*_{\mathcal{A}_i} l\sigma \rightarrow^*_{\mathcal{A}_i} q.$$

If the term rewriting system is left-linear, this condition is trivially satisfied. However, modelling security protocols by term-rewriting systems requires some non left-linear rules. To solve this problem, in [36], Genet et al. propose general conditions that have to be (not trivially) checked at each step of the completion semi-procedure. We want to emphasise the fact that these conditions are not well-adapted to the automatic computations.

In practice, non left-linear variables often concern either the identity of the agents or numbers for nonces/keys, which are constants in the framework of nonce abstraction we have been implementing. Thus, we type constants, variables and states to ensure the validity of the computation. Unlike the conditions [36] on the TRS $\mathcal{R}$ and $\mathcal{A}_i^*$, the verification of which is needed at each completion step i for $\mathcal{A}_i$, our criterion is very simple to verify. Indeed, it is enough to check this condition on the initial automaton $\mathcal{A}_0$ and on the TRS $\mathcal{R}$. Moreover, the condition keeps holding for all automata generated during the completion by the TRS $\mathcal{R}$. Notice that if the criterion does not apply, we can rename non-linear variables to obtain a new coarser term rewriting system; This coarser TRS is not a problem since we compute an over-approximation.

As an illustration, let us consider the following step of the protocol SHARE expressed in IF as follows:

$$\begin{aligned} &\mathit{state\_SHARE\_Init}(A, B, Ka, Kb, 2, Na, Msg, Nb, K, CID). \\ &\qquad\qquad \mathit{iknows}(\mathit{scrypt}(K, \mathit{pair}(\mathit{one}, Msg))) \rightarrow \\ &\mathit{state\_SHARE\_Init}(A, B, Ka, Kb, 3, Na, Msg, Nb, K, CID) \end{aligned}$$

where A and B are variables are of type $agent$, Ka and Kb are of type $public_key$, Na, Msg, Nb are of type $text$, K is of type $message$, and CID is of type nat.

The variables Msg and K appear twice in the left-hand side of this rule. Since the translation of each information, such that its type is $text$, is built using the identity of agents concerned, the non left-linearity of Msg is removed. However, the type of K is $message$, and the size of information represented by K may be unbounded. Consequently, K cannot be typed. So the occurrence of K in the role description is substituted by K'. Obviously, the terms representing the messages sent and received in the protocol are supposed to be left-linear. Otherwise, the method cannot be performed on this kind of protocols (a term like $\mathit{iknows}(Pair(X, X))$ with $X : message$ makes the TRS non runnable). The step of the protocol is translated into the following rewriting rule:

$$state\_SHARE\_Init(agt(A), agt(B), pubkey(Ka), pubkey(Kb), symbcst(2),$$
$$text3(agt(A), agt(B), symbcst(t1)), text3(agt(A), agt(B), symbcst(t0)),$$
$$text3(agt(X0), agt(X1), symbcst(t2)), K', symbcst(CID),$$
$$msg(scrypt(K, pair(symbcst(one), text3(agt(A), agt(B), symbcst(t0))))))$$
$$\rightarrow$$
$$state\_SHARE\_Init(agt(A), agt(B), pubkey(Ka), pubkey(Kb),$$
$$symbcst(3), text3(agt(A), agt(B), symbcst(t1)),$$
$$text3(agt(A), agt(B), symbcst(t0)), text3(agt(X0), agt(X1), symbcst(t2)),$$
$$K', symbcst(CID), msg(symbcst(nil)))$$

Note that functional symbols are denoted by words beginning with a lower-case letter, and variables are denoted by words beginning with a capital letter. Note also that the functional symbol *msg* represents the message received in the LHS and the message sent in the RHS for a given rule. So, in the previous example, the agent playing *state_SHARE_Init* role ends the protocol and receives a message encoded with a fresh symmetric key. We consider a message received or sent as a piece of knowledge of an agent, so this message is contained inside a state term. The intruder can create and analyse *msg* terms, but he cannot have access to the other information contained inside a *state* term, except when he is the main "actor" of the role. The separation between a state and a message received or sent is handled by the tree automaton. Although our semantics differs a little from that of IF, a good use of tree automata allows us to be compatible with the standard of the project.

Notice too that the approximation obtained by renaming the variable K is not too coarse, since this protocol is successfully verified by our tool (see Section 3.4).

To summarise the symbolic approximation main advantages, notice that

- the normalisation procedure reuses states in a dynamic way,
- its implementation is simpler than previously since there is one approximation function rule per rule of the TRS under consideration,
- this approximation function use is fully automatic,
- roles have been introduced, and the TRS generated allows us to reuse already known information in the same manner as described in Deliverable 5.1 [5].

$A := \mathcal{L}(\mathcal{B}_0) \cap \mathcal{L}(\mathcal{A}_{neg})$;

$E := \mathcal{L}(\mathcal{A}_0) \cap \mathcal{L}(\mathcal{A}_{neg})$;

$n := 0$;

while ($E = \emptyset$ **and** $A \neq \emptyset$) **do**

$n := n + 1$;
$A := \mathcal{L}(\mathcal{B}_n) \cap \mathcal{L}(\mathcal{A}_{neg})$;
$E := \mathcal{L}(\mathcal{A}_n) \cap \mathcal{L}(\mathcal{A}_{neg})$;

endwhile

if ($E \neq \emptyset$)

then return false;
else return true;

endif

Figure 1: A semi-algorithm

Semi-algorithm. In this section, we describe a semi-algorithm based on the symbolic approximation above to under-approximate and to over-approximate the network and thus the intruder knowledge.

Let $\mathcal{A}_0$ be the automaton representing the initial configuration of the network and different actions the intruder is able to perform. Let n be a completion step number where one begins to apply the approximation function. Let $\mathcal{A}_n$ be the automaton resulting n completion steps without approximation. Let $\mathcal{B}_n$ be the automaton obtained by symbolic approximation applied to $\mathcal{A}_n$. Let $\mathcal{A}_{neg}$ be the automaton coding a property negation, i.e. accepting the sets of secret terms.

The idea of the semi-algorithm is as follows. If $\mathcal{L}(\mathcal{A}_n) \cap \mathcal{L}(\mathcal{A}_{neg}) \neq \emptyset$ then an attack may exist. In any case $\mathcal{R}^*(\mathcal{L}(\mathcal{A}_0)) \cap \mathcal{L}(\mathcal{A}_{neg}) \neq \emptyset$ and the protocol cannot be checked in the framework of abstraction-based approximation approach (this question is addressed in Section 3.5). Otherwise, if $\mathcal{L}(\mathcal{B}_n) \cap \mathcal{L}(\mathcal{A}_{neg}) = \emptyset$ then the protocol is safe. If $\mathcal{L}(\mathcal{B}_n) \cap \mathcal{L}(\mathcal{A}_{neg}) \neq \emptyset$ we cannot conclude because of the approximation.

More precisely, we use the semi-algorithm we give in Figure 1. It is not difficult to show, as we do in [9], that then the following holds.

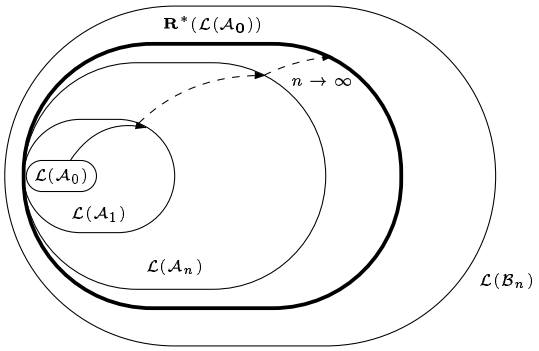

Figure 2: Inclusions of languages involved in Theorem 3

Theorem 3. *(Symbolic approximations, see Figure 2 for inclusions of languages)*

- $\mathcal{L}(\mathcal{A}_n) \subseteq \mathcal{R}^*(\mathcal{L}(\mathcal{A}_0))$ *and* $\lim_{n\to\infty} \mathcal{L}(\mathcal{A}_n) = \mathcal{R}^*(\mathcal{L}(\mathcal{A}_0))$,
- $\mathcal{R}^*(\mathcal{L}(\mathcal{A}_0)) \subseteq \mathcal{L}(\mathcal{B}_n)$.

Proof. (sketch) The inclusion $\mathcal{R}^*(\mathcal{L}(\mathcal{A}_0)) \subseteq \mathcal{L}(\mathcal{B}_n)$ can be obtained by adapting the proof of Theorem 1 in [36]. Similar arguments may be used to prove that $\mathcal{R}(\mathcal{L}(\mathcal{A}_0)) \subseteq \mathcal{L}(\mathcal{A}_1)$. By a direct induction on n, one easily obtains that $\mathcal{R}^n(\mathcal{L}(\mathcal{A}_0)) \subseteq \mathcal{L}(\mathcal{A}_n)$. Therefore $\mathcal{R}^*(\mathcal{L}(\mathcal{A}_0)) \subseteq \lim_{n\to\infty} \mathcal{L}(\mathcal{A}_n)$. Next, we prove that a term t of $\mathcal{L}(\mathcal{A}_1)$ belongs to $\mathcal{R}^*(\mathcal{L}(\mathcal{A}_0))$ by induction on the number of transitions of $\mathcal{A}_1 \setminus \mathcal{A}_0$ used to reduce t to a final state in $\mathcal{A}_1$. Consequently, $\mathcal{L}(\mathcal{A}_1) \subseteq \mathcal{R}^*(\mathcal{L}(\mathcal{A}_0))$. A direct induction on n provides that $\mathcal{L}(\mathcal{A}_n) \subseteq \mathcal{R}^*(\mathcal{L}(\mathcal{A}_0))$. As a direct consequence, one also has $\lim_{n\to\infty} \mathcal{L}(\mathcal{A}_n) = \mathcal{R}^*(\mathcal{L}(\mathcal{A}_0))$. □

3.3 Implementation

The symbolic approximation approach has been implemented (the architecture of the tool, called TA4SP, is displayed in Figure 3). It combines the existing tools, the IF translator and the Timbuk2 library, and the following tools we have been implementing.

- A Timbuk2 package which computes a tree automaton recognising an over-approximation of the reachable terms as defined in Section 3.2. As for Timbuk2, its implementation is done in `OCAML`.

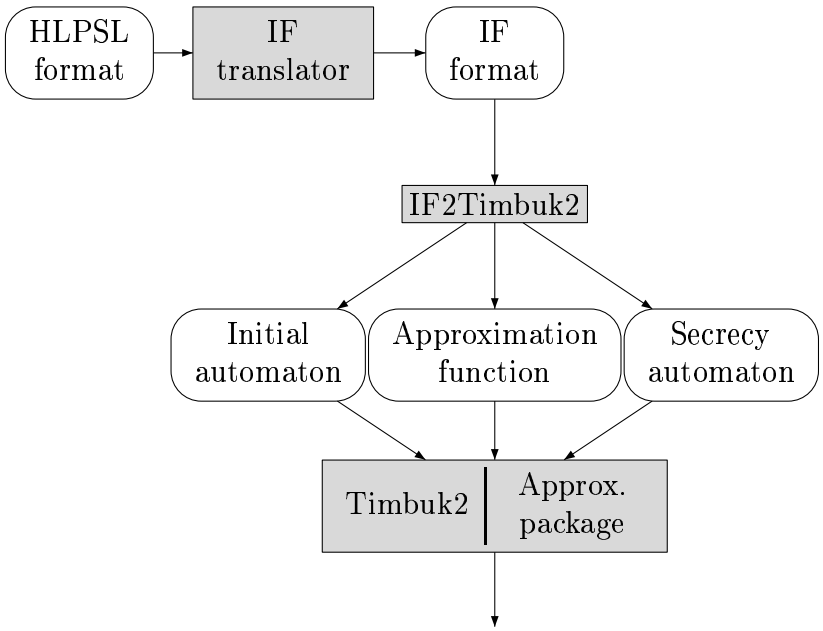

Figure 3: Architecture of TA4SP

- A translator from the IF into Timbuk2 input files, called *IF2Timbuk2*. This tool generates:
  - a file encoding the initial automaton from the `section inits` of the IF specification,
  - a file encoding the set of undesirable configurations (nonces, keys, etc.) as a tree automaton $\mathcal{A}_{neg}$ from the `section goals` of the IF specification, and
  - a file encoding the approximation function used for the normalisation from the `section rules` of the IF specification.

3.4 Experiments

In this section, we present the results of the experiments we have carried out to measure the benefits of the approximation for several protocols of the AVISPA library [6]. Recall that our tool only targets the secrecy property.

For all tested protocols (see Figure 4), the intersection of the over-approximation of the reachable terms and of the set of secret terms is empty; all the protocols are verified. These results show that the abstractions and the approximations used are not too coarse. Notice that for the protocols

Protocol	*Secrecy*	*Time (min)*
UMTS_AKA	verified	≤ 1
CHAPv2	verified	≤ 1
EKE	verified	≤ 1
SHARE	verified	5
SPEKE	verified	≤ 1
IKEv2-CHILD	verified	4
SRP_Siemens	verified	952

Figure 4: Results

IKEv2-CHILD, *SHARE*, and *SPEKE* we have modified the term rewriting system by renaming some non left-linear variables. The experiments show that the computation time depends on the size of the TRS, on the number of variables occurring in the rules, and on the renaming of non left-linear variables. For instance the computation time for the protocol *SRP_siemens* is due to the renaming of variables plus the fact that the intruder can compose a large number of acceptable[2] messages. Among all these messages, many of them would not have to be accepted by the agents in a real execution of the protocol. The tests are performed on a Pentium4 2.4GHz with 640 Mb of RAM.

3.5 Perspectives

Attack (re)-construction. At present our tool does not supply an attack trace on unsafe protocols. We intend to develop a technique aiming to insure this functionality. As explained in Section 3.2, for a given protocol our approach may answer:

true: meaning that the protocol is verified. In our approach we assume that messages are typed, so we do not check for possible type-flaw attacks on the protocols.

false: meaning that a secret term is reachable from an initial configuration using finitely many rewriting rules of the TRS $\mathcal{R}$. At this stage, we are sure that with the considered initial automaton $\mathcal{A}_0$ and the TRS $\mathcal{R}$, the protocol cannot be checked. However, it does not mean that

[2]For a given rule of the TRS, the pattern matching between the left-hand side and an existing term represents the fact that the agent receives an expected — well-formed — message and can compose the next one.

there is an attack. Indeed, the abstraction used and term rewriting systems generated are already two kinds of approximation. Our idea is as follows.

(1) We mark the states of the computed final automaton $\mathcal{A}_k$ in order to build a derivation leading from an initial configuration to the secret term.

(2) Then, using this derivation, we try to build an attack. If we succeed, the protocol is not verified, and we can produce the attack. Otherwise, we can use the derivation to point out the states where the approximations/abstractions are too coarse, so that we can then try to refine the symbolic abstraction-based approximation.

don't know: meaning that the user has manually stopped the semi-algorithm during the while loop. At this stage, we know that $\mathcal{L}(\mathcal{B}_n) \cap \mathcal{L}(\mathcal{A}_{neg}) \neq \emptyset$ and that $\mathcal{L}(\mathcal{A}_n) \cap \mathcal{L}(\mathcal{A}_{neg}) = \emptyset$. We know that the over-approximation $\mathcal{L}(\mathcal{B}_n)$ contains secret terms. We may use the same approach as for the case **false**.

(1) We try to build a derivation from an initial configuration to a secret term. If we do not succeed, we try to refine the approximation.

(2) Otherwise, we proceed as in (2) of the case **false**.

We intend to develop automatic procedures leading to the common output format.

Non left-linear term rewriting systems. The main limitation of the abstraction-based approximation approaches is that the term rewriting system under consideration have to be left-linear ones. We explain in Section 3.2 how to circumvent this problem in most of cases by using typed constants. However, it does not work in all cases. In order to consider typed attacks or to extend the IF fragment of runnable protocols, for instance to those with XOR or algebraic properties, we intend to develop specific techniques for the XOR operator or the exponentiation as well as general techniques where a possible way could be to use tree automata with constraints.

4 Abstractions in Fixed-Point Computation

In this section, we summarise the current work on abstraction-based infinite-state model-checking that has been formalised and incorporated in the tool

OFMC-FP.

4.1 Control Abstraction

As already described in Deliverable 5.1 [5], part of our work is to identify the formal relationship between models used in (abstract) model-checking and other analysis methods. We have identified that a form of control abstraction, which is implicitly used in a wide variety of approaches, can be traced back up to the approach of Paulson [48] which is based on interactive theorem-proving. Based on the preliminary work of [5], we have investigated the precise formal relationship and shown that our standard IF-based model of protocols under the control abstraction is equivalent to Paulson's model. In particular, if OFMC-FP can prove the security of a protocol, then this proof can be translated into a respective security proof for the model of [48]. This allows us to consider OFMC-FP just as a "proof-tactic" for the security of protocols in the interactive theorem proving environment of [48]. This might enhance confidence in the results of OFMC-FP, as the statement is then established inside a widely accepted framework. This is work in progress and we will report on it in the future.

4.2 Data Abstraction

One of the basic ideas in infinite-state model-checking is the abstraction on the domain of data, abstracting the domain into a finite one.[3] In particular, in protocol analysis, the infinite set of fresh data that honest agents might create is abstracted into finitely many equivalence classes.

As an intuition, one may imagine that several honest agents execute the classical Needham-Schroeder public-key protocol NSPK [19, 46], but whenever they talk to the same agent they always use the same nonce (rather than creating a fresh one for every communication). This may lead to additional vulnerabilities of the protocol (that would not have been present if the agents indeed always created fresh data), but it does not exclude other existing vulnerabilities. Thus, if the protocol is safe even under the abstraction that agents use the same nonce in several protocol runs, then it is also safe in the unbounded setting. As the abstract model contains only finitely many different data, it can be checked (by exhaustive search) if the abstract model contains an attack.

This approach allows us to check secrecy for a variety of protocols, however for authentication the described abstraction is often too coarse. For

[3] In Deliverable 5.1 [5], we used the more specific term "nonce abstraction" for data-abstraction, as the data that appears unbounded in security protocols are usually nonces.

instance, in the case of the NSPK, if an honest agent always uses the same nonce, he looses all his freshness guarantees.

Based on the work described in Deliverable 5.1 [5], we have been investigating several finer abstractions, such as the abstraction that we call *genuine-Skolem abstraction*, which are used to distinguish the data into finer equivalence classes, e.g. when they were created as reactions to different inputs.

These refined abstractions are very useful, in particular to prove authentication properties, where several other abstraction-based approaches failed. Also, the abstractions are closely tailored to the problem, as opposed to a "blind" refinement strategy which is not specialised to protocol verification as for instance in [14, 49].

4.3 Finer Abstractions

For several protocols, these abstractions are still too coarse. As a concrete example, we discuss the ASW protocol [4]. This is a contract signing protocol with a trusted 3rd party (T3P) which maintains a data-base of aborted and resolved contracts, identified by the (hash-value) of the nonces appearing in these contracts. Abstracting the nonce values leads necessarily to trivial attacks in such a protocol as we will now explain.

The problem is that the abstraction must distinguish nonces that appear in a contract which was aborted from those that appear in a contract which was resolved. However, when a nonce is first created and used it is not yet clear whether it will later be aborted or resolved: no abstraction to finitely many equivalence classes (no matter how fine the abstraction is) can 'correctly' classify these nonces (in the sense that then the protocol is not flawed for trivial reasons).

The problem arises as in this and many other cases, we have agents (like server-processes) who maintain data-bases of values and their behaviour depends on the equivalence class induced by the data-base into which the input-data falls. As several of the protocols that we want to analyse in the AVISPA project actually use such features, we would like to have a general method for this infinite-state analysis.

The idea that we are currently following is that during a transition, a piece of data might change the equivalence class to which it belongs in the abstract model. For instance, in the ASW protocol, we may distinguish for every nonce three classes according to the following three predicates:

- The T3P has never seen a contract with this nonce yet.
- The T3P has already issued an abort contract with this nonce.

- The T3P has already issued a resolve contract with this nonce.

Suppose that in a transition the T3P receives an abort request and the nonce contained in this abort request falls into the first of the above three classes. Then the T3P can abort the contract, but after the transition the nonce of this contract belongs to the second class.

This idea is loosely based on the idea of *predicate abstraction* [27], i.e. abstractions where the equivalence classes are characterised by predicates on the data. We combine this with the standard way of abstracting control and data to again obtain a method well-tailored to the analysis of security protocols.

The first experiments with the new kind of refined abstractions are encouraging [39]: in particular, we have been able to identify an "attack" on the ASW protocol that is due to a common mistake in the specification of the goals of the protocol, as well as show that the protocol satisfies a corrected (slightly weakened) goal.

4.4 Perspectives

As part of our current work, we are improving the OFMC-FP tool, which is still in a preliminary state, into a tool that can be used by non-experts, as is currently the case for OFMC-core (for finitely many sessions). To this end, we are currently implementing the different abstractions described above and in [5], in order to scale the applicability of our verification tools to the industrial-strength protocols.

5 Conclusion

We have presented several promising protocols verification procedures for handling an arbitrary number of sessions and infinite datatypes. They cover a large class of protocol and properties. As future work, the symbolic exploration procedure for time-sensitive protocols will be optimised, and the tree automata based software will be extended in order to supply attack traces and also to cope with non-linear variables.

References

[1] P. Abdulla and B. Jonsson. On the existence of network invariants for verifying parametrized systems. In *Correct System Design – Recent Insights and Advances*, LNCS 1710, pages 180–197. Springer-Verlag, 1999.

[2] P. A. Abdulla, A. Bouajjani, B. Jonsson, and M. Nilsson. Handling global conditions in parametrized system verification. In *Proceedings of CAV'99*, LNCS 1633, pages 134–145. Springer Verlag, 1999.

[3] A. Annichini, E. Asarin, and A. Bouajjani. Symbolic techniques for parametric reasoning. In *Proceedings of CAV'00*, LNCS 1855, pages 419–434. Springer Verlag, 2000.

[4] N. Asokan, V. Shoup, and M. Waidner. Asynchronous protocols for optimistic fair exchange. In *Proceedings of the IEEE Symposium on Research in Security and Privacy*, pages 86–99, 1998.

[5] AVISPA. Deliverable 5.1: Abstractions. Available at `http://www.avispa-project.org`, 2003.

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

[7] G. Bella and L. C. Paulson. Kerberos version IV: Inductive analysis of the secrecy goals. In J.-J. Quisquater, editor, *Proceedings of the 5th European Symposium on Research in Computer Security*, LNCS 1485, pages 361–375. Springer-Verlag, 1998.

[8] B. Blanchet. An efficient cryptographic protocol verifier based on prolog rules. In *Proceedings of CSFW'01*, pages 82–96. IEEE Computer Society Press, 2001.

[9] Y. Boichut, P. C. Héam, and O. Kouchnarenko. Symbolic normalisation for verification of security protocols. in preparation.

[10] Y. Boichut, P.-C. Heam, O. Kouchnarenko, and F. Oehl. Improvements on the Genet and Klay Technique to Automatically Verify Security Protocols. In *Automated Verification of Infinite States Systems (AVIS'04)*, ENTCS, 2004. To appear.

[11] A. Bouajjani, B. Jonsson, M. Nilsson, and T. Touili. Regular model checking. In *Proceedings of CAV'00*, LNCS 1855, pages 403–418. Springer-Verlag, 2000.

[12] L. Bozga, Y. Lakhnech, and M. Perin. Pattern-based abstraction for verifying secrecy in protocols. In *Proceedings of TACAS 2003*, LNCS 2619. Springer-Verlag, 2003.

[13] M. Bozzano and G. Delzanno. Automated Protocol Verification in Linear Logic. In *Proceedings of the International ACM SIGPLAN Conference on Principle and Practice of Declarative Languages (PPDP 2002)*, pages 38–49. ACM Press, 2002.

[14] P. Broadfoot, G. Lowe, and A. Roscoe. Automating data independence. In *Proceedings of Esorics 2000*, LNCS 1895, pages 175–190. Springer-Verlag, 2000.

[15] T. Bultan, R. Gerber, and W. Pugh. Model-checking concurrent systems with unbounded integer variables: Symbolic representations, approximations, and experimental results. *ACM Transactions on Programming Languages and Systems*, 21(4):747–789, 1999.

[16] M. Burrows, M. Abadi, and R. Needham. A Logic of Authentication. *ACM Transactions on Computer Systems*, 8(1):18–36, 1990.

[17] F. Butler, I. Cervesato, A. Jaggard, and A. Scedrov. A formal analysis of some properties of Kerberos 5 using msr. In *Proceedings of the 15th IEEE Computer Security Foundations Workshop: CSFW'02*, pages 175–190. IEEE Computer Society Press, 2002.

[18] I. Cervesato, N. Durgin, P. Lincoln, J. Mitchell, and A. Scedrov. A meta-notation for protocol analysis. In *Proceedings of the 12th IEEE Computer Security Foundations Workshop: CSFW'99*, pages 55–69. IEEE Computer Society Press, 1999.

[19] 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`.

[20] E. M. Clarke, O. Grumberg, and S. Jha. Verifying parameterized networks using abstraction and regular languages. In *Proceedings of CONCUR'95*, pages 395–407. Springer-Verlag, 1995.

[21] E. Cohen. TAPS: A first-order verifier for cryptographic protocols. In *Proceedings of CSFW'00*. IEEE Computer Society Press, 2000.

[22] H. Comon and Y. Jurski. Multiple counters automata, safety analysis and presburger arithmetic. In *Proceedings of CAV'98*, LNCS 1427. Springer-Verlag, 1998.

[23] H. Comon-Lundh and V. Cortier. New decidability results for fragments of first-order logic and application to cryptographic protocols. Technical

Report LSV-03-3, Laboratoire Specification and Verification, ENS de Cachan, Cachan, France, January 2003.

[24] H. Comon-Lundh and V. Cortier. Security properties: two agents are sufficient. In *Proceedings of ESOP'2003*, LNCS 2618, pages 99–113. Springer-Verlag, 2003.

[25] K. Compton and S. Dexter. Proof techniques for cryptographic protocols. In *Proceedings of the 26th International Colloquium ICALP'99*, 1999.

[26] P. Cousot. Abstract interpretation. *ACM Computing Surveys*, 28(2):324–328, June 1996.

[27] S. Das and D. L. Dill. Successive approximation of abstract transition relations. In *Proceedings of LICS'01*, 2001.

[28] G. Delzanno. Automatic verification of parameterized cache coherence protocols. In E. A. Emerson and A. P. Sistla, editors, *Proceedings 12th International Conference on Computer Aided Verification (CAV'00)*, volume 1855 of *LNCS*, Chicago, Illinois, 2000. Springer-Verlag.

[29] G. Delzanno and T. Bultan. Constraint-based verification of client-server protocols. In Proc. of Constraint Programming 2001, volume 2239 of *LNCS*, pages 286–301. Springer-Verlag, 2001.

[30] G. Delzanno and P. Ganty. Automatic Verification of Time Sensitive Cryptographic Protocols. In *Proceedings of TACAS'04*. Springer-Verlag, 2004. Available at `http://www.avispa-project.org`.

[31] G. Delzanno and A. Podelski. Model checking in CLP. In R. Cleaveland, editor, *Proceedings 5th International Conference on Tools and Algorithms for Construction and Analysis of Systems (TACAS'99)*, volume 1579 of *LNCS*, pages 223–239, Amsterdam, The Netherlands, 1999. Springer-Verlag.

[32] G. Delzanno, J.-F. Raskin, and L. V. Begin. Attacking Symbolic State Explosion. In G. Berry, H. Comon, and A. Finkel, editors, *Proceedings 13th International Conference on Computer Aided Verification (CAV'01)*, volume 2102 of *LNCS*, pages 298–310, Paris, France, 2001. Springer-Verlag.

[33] N. Durgin, P. D. Lincoln, J. C. Mitchell, and A. Scedrov. Undecidability of Bounded Security Protocols. In *Proceedings of the FLOC'99 Workshop on Formal Methods and Security Protocols (FMSP'99)*, 1999.

[34] N. Evans and S. Schneider. Analysing time dependent security properties in CSP using PVS. In *Proceedings of Esorics 2000*, LNCS 1895, pages 175–190. Springer-Verlag, 2000.

[35] S. Even and O. Goldreich. On the security of multi-party ping pong protocols. In *Proceedings of 24th IEEE Symposium on Foundations of Computer Science*. IEEE Computer Society, 1983.

[36] G. Feuillade, T. Genet, and V. VietTriemTong. Reachability analysis over term rewriting systems. *Journal of Automated Reasoning*, 2004. To appear.

[37] T. Genet and F. Klay. Rewriting for cryptographic protocol verification. In *Proceedings of CADE'00*, LNCS 1831, pages 271–290. Springer-Verlag, 2000.

[38] T. Genet and V. Viet Triem Tong. Reachability analysis of term rewriting systems with timbuk. In *Proceedings of LPAR'01*, LNCS 2250, 2001.

[39] P. Hankes Drielsma and S. Mödersheim. The asw protocol revisited: A unified view. In *Proceedings of the IJCAR04 Workshop ARSPA*, 2004. To appear in ENTCS, available at `http://www.avispa-project.org`.

[40] N. Heintze and J. Tygar. A model for secure protocols and their compositions. *IEEE Transactions on Software Engineering*, 22(1):16–30, 1996.

[41] Y. Kesten, O. Maler, M. Marcus, A. Pnueli, and E. Shahar. Symbolic model checking with rich assertional languages. In *Proceedings of CAV'97*, pages 424–435. Springer-Verlag, 1997.

[42] G. Lowe. A family of attacks upon authentication protocols. Technical Report 1997/5, Department of Mathematics and Computer Science, University of Leicester, 1997.

[43] G. Lowe. A hierarchy of authentication specifications. In *Proceedings of the 10th IEEE Computer Security Foundations Workshop (CSFW'97)*, pages 31–43. IEEE Computer Society Press, 1997.

[44] G. Lowe. Casper: a Compiler for the Analysis of Security Protocols. *Journal of Computer Security*, 6(1):53–84, 1998. See `http://web.comlab.ox.ac.uk/oucl/work/gavin.lowe/Security/Casper/`.

[45] G. Lowe. Towards a completeness result for model checking of security protocols. In *Proceedings of the 11th IEEE Computer Security Foundations Workshop (CSFW'98)*, pages 96–105. IEEE Computer Society Press, 1998.

[46] R. M. Needham and M. D. Schroeder. Using Encryption for Authentication in Large Networks of Computers. Technical Report CSL-78-4, Xerox Palo Alto Research Center, Palo Alto, CA, USA, 1978. Reprinted June 1982.

[47] F. Oehl, G. Cécé, O. Kouchnarenko, and D. Sinclair. Automatic approximation for the verification of cryptographic protocols. In *Proceedings of Conference on Formal Aspects of Security*, LNCS 2629. Springer-Verlag, 2003.

[48] L. C. Paulson. The Inductive Approach to Verifying Cryptographic Protocols. *Journal of Computer Security*, 6(1):85–128, 1998.

[49] A. Roscoe and P. Broadfoot. Proving security protocols with model checkers by data independence techniques. *Journal of Computer Security*, 7:147–190, 1999.

[50] A. P. Sistla and V. Gyuris. Parameterized verification of linear networks using automata as invariants. *Formal Aspects of Computing*, 11(4):402–425, 1999.

[51] D. Song. Athena: A new efficient automatic checker for security protocol analysis. In *Proceedings of the 12th IEEE Computer Security Foundations Workshop (CSFW '99)*, pages 192–202. IEEE Computer Society Press, 1999.