Symbolic trace analysis of cryptographic protocols*

Michele Boreale

Dipartimento di Sistemi e Informatica, Università di Firenze, Via Lombroso 6/17, 50134 Firenze, Italia. E-mail boreale@dsi.unifi.it.

Abstract. A cryptographic protocol can be described as a system of concurrent processes, and analysis of the traces generated by this system can be used to verify authentication and secrecy properties of the protocol. However, this approach suffers from a state-explosion problem that causes the set of states and traces to be typically infinite or very large. In this paper, starting from a process language inspired by the spi-calculus, we propose a symbolic operational semantics that relies on unification and leads to compact models of protocols. We prove that the symbolic and the conventional semantics are in full agreement, and then give a method by which trace analysis can be carried out directly on the symbolic model. The method is proven to be complete for the considered class of properties and is amenable to automatic checking.
Keywords: spi-calculus, concurrency, formal methods for security protocols.

1 Introduction

In recent years, formal methods have proven useful in the analysis of cryptographic protocols, often revealing previously unknown attacks. A popular approach is that of modelling a protocol as a system of concurrent processes, described using an appropriate language, like CSP [13, 19, 21] or the spi-calculus [1] – the latter an extension of the π-calculus [17]. In this setting, Abadi and Gordon advocate the use of observational equivalences to formalize and verify protocol properties [2, 7]. Here, in the vein of [3, 4, 12, 13, 16, 19, 21], we analyze the sequences of actions (traces) that a given process may execute. As an example, a *secrecy* property like "protocol P never leaks the datum d", might be verified by adding to the description of P an 'error' action to be performed as soon as the environment learns d (the way this is done depends on the specific formalism, see e.g. [3]), and then checking that P never performs that 'error' action.

The main drawback of trace analysis is that the execution of a protocol typically generates infinitely many traces. The reason lies in the modelling of the environment, whose behaviour is largely unpredictable. Rather than trying to describe this behaviour as a specific process, it is sensible to simply assume that the communication network is totally under the control of the environment. The latter can store, duplicate, hide or replace messages that travel on the network. It can also operate according to the rules followed by honest participants and

* A preliminary version of this paper has been circulated as [5]. Research partly supported by the Italian MURST Project TOSCA *(Teoria della Concorrenza, Linguaggi di Ordine Superiore e Strutture di Tipi).*

synthetize new messages by pairing, decryption, encryption and creation of fresh nonces and keys, or by arbitrary combinations of these operations (this approach seems to date back to Dolev and Yao [11]). Thus, an agent waiting for an input at a given moment may expect any of the infinitely many messages the environment can produce and send on the network. This leads to a state explosion that makes the protocol model, typically a state-transition graph, infinite (more precisely, infinite-branching). In practice, those approaches that rely on *model checking* [13, 19, 21] cut down the model to a convenient finite size by imposing upper-bounds to the critical parameters (number of keys, number of pairing and encryption in messages,...). Exhaustive exploration of the state-space is then possible by standard techniques. However, this approach makes correctness in the general case (completeness) very difficult to establish – though some progress has recently been made [15, 20]. Furthermore, even when those upper-bounds can be justified and the model is finite, the branching factor of input actions may cause the number of states and traces to explode as larger systems are considered.

In this paper we explore an alternative approach to trace analysis of cryptographic protocols. As a base language, we consider a variant of the spi-calculus, but this choice is not critical for the development of the theory. The idea is to replace the infinitely many transitions arising from an input action by a single *symbolic* transition, and to represent the received message as a variable. Constraints on this variable are accumulated as the execution proceeds. Let us see this in more detail. In the variant of the spi-calculus we use, the receiver of a message is written as $\mathsf{a}(x).\,R$, where a is an arbitrary label, x is the input variable and R is the continuation. The conventional ('concrete') operational semantics of the language requires x to be instantiated with each message that can possibly be received from the network, and this causes the state explosion. In our symbolic semantics, x is not instantiated immediately, rather constraints on its value are added as needed. These constraints take the form of *most general unifiers*. As an example, suppose that a process P, after receiving a message x, tries decryption of x using key k, and, if this succeeds, calls y the result and proceeds like P'. This is written as $P \stackrel{\text{def}}{=} \mathsf{a}(x).\,\mathsf{case}\,x\,\mathsf{of}\,\{y\}_k\,\mathsf{in}\,P'$. We represent a state of the protocol as a pair (σ, Q), where σ is the trace of process' past actions and Q is a process term. The two initial symbolic transitions of $(\epsilon,\ P)$ will be:

$$(\epsilon,\ P) \ \longrightarrow_{\mathrm{s}}\ (\mathsf{a}\langle x\rangle,\ \mathsf{case}\,x\,\mathsf{of}\,\{y\}_k\,\mathsf{in}\,P') \ \longrightarrow_{\mathrm{s}}\ (\mathsf{a}\langle\{y\}_k\rangle,\ P'[\{y\}_k/x])$$

where $[\{y\}_k/x]$ is the most general unifier for x and $\{y\}_k$. In general, protocols not using replication/recursion will generate finitely many symbolic transitions. The resulting model is rather compact: sequential processes will just exhibit a single complete symbolic trace, while, for parallel compositions, traces will be obtained as usual by interleaving. We prove that the symbolic and the conventional semantics are in full agreement, and then give a method by which trace analysis can be carried out directly on the symbolic traces. We focus our attention on a specific class of properties, those of the form "in every execution of the protocol, action α happens prior to action β", for given α and β. As we shall see, this scheme is flexible enough to express interesting forms of authentication

and secrecy. The method is proven to be complete for the language we consider, and is easily mechanizable; this immediately yields decidability of trace analysis for the considered language. When a protocol does not satisfy a property, the method also gives an easy way to compute an attack, i.e. a trace that violates the property. A prototype implementation of the method is already available [6].

The language we consider does not contain replication/recursion operators, that would make trace analysis undecidable (see e.g. [12, 10]). Thus we consider only protocols with a bounded number of participants. However, recent work by Lowe [15] and Stoller [22] indicates that, in some cases, it is possible to reduce the analysis of an unbounded protocol to the analysis of a protocol with a bounded, statically determined number of participants.

The symbolic approach to the analysis of security protocols has been explored by other authors, including Amadio and Lugiez [4] and Huima [12]. Discussion with these and other related work can be found in the concluding section.

The rest of the paper is organized a follows. The language, a variant of the spi-calculus with shared-key cryptography, is introduced in Section 2, along with its conventional operational semantics. Section 3 introduces trace analysis. The symbolic operational semantics, and its agreement with the conventional semantics, are discussed in Section 4. Section 5 presents the symbolic method for trace analysis, at the core of which is the concept of *refinement*. For the sake of presentation, the language of Section 2 does not include the restriction operator, whose treatment is postponed to Section 6. Section 7 contains discussion on related work and a few concluding remarks. A separate appendix contains the proof of a major theorem.

2 The language

Syntax (Table 1) We presuppose three countable disjoint sets: $\mathcal{L}$, $\mathcal{N}$ and $\mathcal{V}$. The set $\mathcal{L}$ of *labels* is ranged over by $\mathsf{a}, \mathsf{b}, \ldots$. The set of $\mathcal{N}$ of *names* is partitioned into two countable sets, a set $\mathcal{LN}$ of *local names* $a, b, \ldots$ and a set $\mathcal{EN}$ of *environmental names* $\underline{a}, \underline{b}, \ldots$: these sets represent the basic data (keys, nonces,...) initially known to the process and to the environment, respectively. The set $\mathcal{V}$ of *variables* is ranged over by $x, y, \ldots$. The set $\mathcal{N} \cup \mathcal{V}$ is ranged over by letters $u, v, \ldots$. Names and variables can be used to build compound *messages*, in $\mathcal{M}$, via shared-key encryption and pairing. In particular, $\{M\}_k$ represents the message obtained by encrypting M (the *argument*) under name k (the *key*), using a shared-key encryption system. We allow pairing and encryptions to be arbitrarily nested, but only permit atomic keys. *Terms* are obtained by closing messages under substitutions (they are just 'garbage' that can be generated at run-time, with no semantical significance).

The syntax of *agent expressions*, in $\mathcal{A}$, is taken essentially from the spi-calculus [1]. Agent $\mathsf{case}\,\{M\}_h\,\mathsf{of}\,\{y\}_k\,\mathsf{in}\,A$ tries decryption of $\{M\}_h$ using k as a key: if this is possible (that is, if $k = h$), the result of the decryption, M, is bound to y, and the agent proceeds like A, otherwise, the whole expression is stuck. Similarly, $\mathsf{pair}\,M\,\mathsf{of}\,\langle x, y\rangle\,\mathsf{in}\,A$ tries to split M into two components and

call them x and y. A difference from spi/π-calculus is that, in our case, input and output labels ($\mathsf{a}, \mathsf{b}, \ldots$) must not be regarded as a channels (as already noted, we assume just one public network), but rather as 'tags' attached to process actions for ease of reference. Also, the only useful cases for output and decryption are when ζ is a message and η is a name; otherwise the whole agent is stuck.

Given the presence of binders for variables, notions of *free variables*, $\mathrm{v}(A) \subseteq \mathcal{V}$ and *alpha-equivalence* arise as expected. We shall identify alpha-equivalent agent expressions. For any M and u, $[M/u]$ denotes the operation of substituting the free occurrences of u by M. An agent expression A is said to be *closed* or a *process* if $\mathrm{v}(A) = \emptyset$; the set of processes $\mathcal{P}$ is ranged over by $P, Q, \ldots$. Local names and environmental names occurring in A are denoted by $\mathrm{ln}(A)$ and $\mathrm{en}(A)$, respectively. A process P is *initial* if $\mathrm{en}(P) = \emptyset$. These notations are extended to terms, messages, and tuples/string/sets of such objects, component-wise. We shall also use such abbreviations as $\mathrm{ln}(M, P, Q)$ to mean $\mathrm{ln}(M) \cup \mathrm{ln}(P) \cup \mathrm{ln}(Q)$.

$m, n, \ldots$	names $\mathcal{N}$	$x, y, \ldots$ variables $\mathcal{V}$
$a, b, \ldots, h, k, \ldots$	local names $\mathcal{LN}$	
$\underline{a}, \underline{b}, \ldots, \underline{h}, \underline{k}, \ldots$	environmental names $\mathcal{EN}$	
$u, v, \ldots$	variables or names $\mathcal{V} \cup \mathcal{N}$	$\mathsf{a}, \mathsf{b}, \ldots$ labels $\mathcal{L}$

$M, N ::= u \mid \{M\}_u \mid \langle M, N\rangle$	messages $\mathcal{M}$	
$\eta, \zeta ::= u \mid \{\zeta\}_\eta \mid \langle \zeta, \eta\rangle$	terms $\mathcal{Z}$	

$A, B ::=$	agents $\mathcal{A}$	
$\mathbf{0}$		(null)
$\mid \mathsf{a}(x).\, A$		(input)
$\mid \overline{\mathsf{a}}\langle\zeta\rangle.\, A$		(output)
$\mid \mathsf{case}\, \zeta\, \mathsf{of}\, \{y\}_\eta\, \mathsf{in}\, A$		(decryption)
$\mid \mathsf{pair}\, \zeta\, \mathsf{of}\, \langle x, y\rangle\, \mathsf{in}\, A$		(selection)
$\mid [\zeta = \eta]A$		(matching)
$\mid A \parallel B$		(parallel composition)

The occurrences of variables x and y in (input), (decryption) and (selection) are bound.

Table 1. Syntax

Example 1 (the wide-mouthed frog protocol, WMF [8]). Principals A and B share two secret keys, k_{AS} and k_{BS} respectively, with a server S. The purpose of the protocol is that of establishing a new secret key k between A and B, which A may use to send a confidential datum d to B. For the sake of simplicity, we suppose that the protocol is always started by A. The protocol and its translation in spi-calculus, WMF, are described below:

1. $A \longrightarrow S : \{k\}_{k_{AS}}$
2. $S \longrightarrow B : \{k\}_{k_{BS}}$
3. $A \longrightarrow B : \{d\}_k.$

$$A \stackrel{\mathrm{def}}{=} \overline{\mathsf{a1}}\langle\{k\}_{k_{AS}}\rangle.\, \overline{\mathsf{a2}}\langle\{d\}_k\rangle.\, A'$$

$$S \stackrel{\mathrm{def}}{=} \mathsf{s1}(x).\, \mathsf{case}\, x\, \mathsf{of}\, \{x'\}_{k_{AS}}\, \mathsf{in}\, \overline{\mathsf{s2}}\langle\{x'\}_{k_{BS}}\rangle.\, \mathbf{0}$$

$$B \stackrel{\mathrm{def}}{=} \mathsf{b1}(y).\, \mathsf{case}\, y\, \mathsf{of}\, \{y'\}_{k_{BS}}\, \mathsf{in}\, \mathsf{b2}(z).\, \mathsf{case}\, z\, \mathsf{of}\, \{z'\}_{y'}\, \mathsf{in}\, B'$$

$$WMF \stackrel{\mathrm{def}}{=} A \parallel S \parallel B\,.$$

Agents A' and B' represent the behaviour of A and B, respectively, after the protocol has been completed. The above description just accounts for a single instance of the protocol. The case of $n > 1$ instances is described by composing n copies of A, S and B in parallel and then appropriately renaming the instance-dependent quantities (k, d, n_A and n_B above).

Operational semantics The semantics of the calculus is given in terms of a transition relation $\longrightarrow$, which we will sometimes refer to as 'concrete' (as opposed to the 'symbolic' one we shall introduce later on). We will find it convenient to model a state of the system as a pair (s, P), where, s records the current environment's knowledge (i.e. the sequence of messages the environment has "seen" travelling on the network up to that moment) and P is a process. Similarly to [3, 4, 9], we characterize the messages that the environment can *produce* (or deduce) at a given moment, starting from the current knowledge s, via a deductive system. These concepts are formalized below.

Definition 1 (the deductive system). Let $S \subseteq_{\text{fin}} \mathcal{M}$ and M a message. We let $\vdash$ be the least binary relation generated by the deductive system in Table 2. If $S \vdash M$ we say that S *can produce* M. $\diamond$

$$(\textsc{Ax})\ \frac{}{S \vdash M}\ M \in S \qquad (\textsc{Env})\ \frac{}{S \vdash \underline{a}}\ \underline{a} \in \mathcal{EN}$$

$$(\textsc{Proj}_1)\ \frac{S \vdash \langle M, N\rangle}{S \vdash M} \qquad (\textsc{Proj}_2)\ \frac{S \vdash \langle M, N\rangle}{S \vdash N} \qquad (\textsc{Pair})\ \frac{S \vdash M \quad S \vdash N}{S \vdash \langle M, N\rangle}$$

$$(\textsc{Dec})\ \frac{S \vdash \{M\}_u \quad S \vdash u}{S \vdash M} \qquad (\textsc{Enc})\ \frac{S \vdash M \quad S \vdash u}{S \vdash \{M\}_u}$$

Table 2. Deductive system ($\vdash$)

Note that, whatever S, the set of messages that S can produce is infinite, due to rules Env, Pair and Enc. An *action* is a term of the form $\mathsf{a}\langle M\rangle$ (*input* action) or $\overline{\mathsf{a}}\langle M\rangle$ (*output* action), for a a label and M a message. The set of actions Act is ranged over by $\alpha, \beta, \ldots$, while the set Act^* of strings of actions is ranged over by $s, s', \ldots$. String concatenation is written '$\cdot$'. We denote by $\text{act}(s)$ and $\text{msg}(s)$ the set of actions and messages, respectively, appearing in s. A string s is *closed* if $\text{v}(s) = \emptyset$ (note that s does not contain binders) and *initial* if $\text{en}(s) = \emptyset$. In what follows, we write $s \vdash M$ for $\text{msg}(s) \vdash M$. We are now set to define *traces*, that is sequences of actions that may result from the interaction between a process and its environment. In traces, each message received by the process (input message) is deducible from the knowledge the environment has previously acquired. In *configurations*, the latter is explicitly recorded.

Definition 2 (traces and configurations). A *trace* is a closed string $s \in Act^*$ such that for each s_1, s_2 and $\mathsf{a}\langle M\rangle$, if $s = s_1 \cdot \mathsf{a}\langle M\rangle \cdot s_2$ then $s_1 \vdash M$.

A *configuration*, written as $\langle s,\ P\rangle$, is a pair consisting of a trace s and a process P. A configuration is *initial* if $\mathrm{en}(s, P) = \emptyset$. Configurations are ranged over by $\mathcal{C}, \mathcal{C}', \ldots$. ◇

$$\text{(Inp)}\ \langle s,\ \mathsf{a}(x).\,P\rangle \longrightarrow \langle s \cdot \mathsf{a}\langle M\rangle,\ P[^{M}/_{x}]\rangle \qquad s \vdash M,\ M \text{ closed}$$

$$\text{(Out)}\ \langle s,\ \overline{\mathsf{a}}\langle M\rangle.\,P\rangle \longrightarrow \langle s \cdot \overline{\mathsf{a}}\langle M\rangle,\ P\rangle$$

$$\text{(Case)}\ \langle s,\ \mathsf{case}\,\{\zeta\}_k\,\mathsf{of}\,\{y\}_k\,\mathsf{in}\,P\rangle \longrightarrow \langle s,\ P[\zeta/y]\rangle$$

$$\text{(Select)}\ \langle s,\ \mathsf{pair}\,\langle\zeta,\eta\rangle\,\mathsf{of}\,\langle x,y\rangle\,\mathsf{in}\,P\rangle \longrightarrow \langle s,\ P[\zeta/x,\ \eta/y]\rangle$$

$$\text{(Match)}\ \langle s,\ [\zeta = \zeta]P\rangle \longrightarrow \langle s,\ P\rangle$$

$$\text{(Par)}\ \frac{\langle s,\ P\rangle \longrightarrow \langle s',\ P'\rangle}{\langle s,\ P \parallel Q\rangle \longrightarrow \langle s',\ P' \parallel Q\rangle}$$

plus symmetric version of (Par).

Table 3. Transition relation ($\longrightarrow$).

The concrete transition relation on configurations is defined by the rules in Table 3. Each action taken by the process is recorded in the first component of the configuration (thus a 'labelled' transition relation is not needed). Rule (Inp) makes the transition relation infinite-branching, as M ranges over the infinite set $\{M : s \vdash M,\ M \text{ closed}\}$. Another point worth to notice is that decryption with key k is achieved by matching the term to decrypt against a pattern $\{y\}_k$, where y is a fresh variable (rule (Case)). Finally, no handshake communication is provided (rule (Par)): all messages go through the environment.

3 Trace analysis

Given a configuration $\langle s,\ P\rangle$ and a trace s', we say that $\langle s,\ P\rangle$ *generates* s', written $\langle s,\ P\rangle \searrow s'$, if $\langle s,\ P\rangle \longrightarrow^* \langle s',\ P'\rangle$ for some P'. Given a string of actions $s \in Act^*$, and actions α and β, we say that α *occurs prior to* β *in* s if whenever $s = s' \cdot \beta \cdot s''$ then $\alpha \in \mathrm{act}(s')$. We let ρ range over ground substitutions, i.e. finite maps from a set $\mathrm{dom}(\rho) \subseteq \mathcal{V}$ to closed messages; $t\rho$ denotes the result of replacing each $x \in \mathrm{v}(t) \cap \mathrm{dom}(\rho)$ by $\rho(x)$. The properties of configurations we are interested in are defined below.

Definition 3 (properties). Let α and β be actions, with $\mathrm{v}(\alpha) \subseteq \mathrm{v}(\beta)$, and let s be a trace. We write $s \models \alpha \hookleftarrow \beta$, or s *satisfies* $\alpha \hookleftarrow \beta$, if for each ground substitution ρ it holds that $\alpha\rho$ occurs prior to $\beta\rho$ in s. We say that a configuration

$\mathcal{C}$ *satisfies* $\alpha \hookleftarrow \beta$, and write $\mathcal{C} \models \alpha \hookleftarrow \beta$, if all traces generated by $\mathcal{C}$ satisfy $\alpha \hookleftarrow \beta$. $\diamond$

Note that the variables in α and β can be thought of as being universally quantified (so any consistent renaming of these variables does not change the set of traces and configurations that satisfy $\alpha \hookleftarrow \beta$). In practice, the scheme $\alpha \hookleftarrow \beta$ permits formalizing all forms of authentication in Lowe's hierarchy [14], except the most demanding one (but can be easily modified to include this one as well). As we shall see, the scheme also permits expressing secrecy as a reachability property, in the style of [3]. To this purpose, it is convenient to assume a fixed 'absurd' action $\bot$ that is nowhere used in agent expressions. Thus the formula $\bot \hookleftarrow \alpha$ expresses that action α should never take place, and can be used to encode reachability.

Example 2 (authentication and secrecy in WMF). We discuss the use of properties on the simple protocol WMF (Example 1), but our considerations are indeed quite general.

A property of WMF that one would like to check is the following: if B accepts as 'good' a datum d encrypted under key k (step 3), then this message has actually been sent by A. This is a form of *authentication*. In order to formalize this particular property, we make B explicitly declare if, at the end of the protocol, a particular message has been accepted. That is, we consider the process $B_{auth} \stackrel{\text{def}}{=} \mathsf{b1}(y).\,\mathsf{case}\,y\,\mathsf{of}\,\{y'\}_{k_{BS}}\,\mathsf{in}\,\mathsf{b2}(z).\,\mathsf{case}\,z\,\mathsf{of}\,\{z'\}_{y'}\,\mathsf{in}\,(B' \parallel \overline{\mathsf{accept}}\langle z\rangle.\,\mathbf{0})$ instead of B, and $WMF_{auth} \stackrel{\text{def}}{=} A \parallel S \parallel B_{auth}$ instead of WMF. We have to check that, in every trace of WMF_{auth}, every $\overline{\mathsf{accept}}$ is preceded by the corresponding $\overline{\mathsf{a2}}$. More formally, we have to check that $\langle \epsilon,\ WMF_{auth}\rangle \models \overline{\mathsf{a2}}\langle t\rangle \hookleftarrow \overline{\mathsf{accept}}\langle t\rangle$ (where t is any variable).

Another important property is *secrecy*: the environment should never learn the confidential datum d. Following [3], we can formalize this property by considering a version of WMF that also includes a 'guardian' listening to the public network: $WMF_{secr} \stackrel{\text{def}}{=} WMF \parallel \mathsf{guard}(x).\,\mathbf{0}$. Evidently, WMF generates a trace s s.t. $s \vdash d$ (i.e. the environment learns d) if and only if $\langle \epsilon,\ WMF_{secr}\rangle$ generates a trace containing action $\mathsf{guard}\langle d\rangle$. Thus, we have to check that action $\mathsf{guard}\langle d\rangle$ never takes place, i.e. that $\langle \epsilon,\ WMF_{secr}\rangle \models \bot \hookleftarrow \mathsf{guard}\langle d\rangle$.

4 Symbolic semantics

'Concrete' traces and configurations can be given a symbolic counterpart, which may contain free variables.

Definition 4 symbolic traces and configurations. A *symbolic trace* is string $\sigma \in Act^*$ such that: (a) $\mathrm{en}(\sigma) = \emptyset$, and (b) for each σ_1, σ_2, α and x, if $\sigma = \sigma_1 \cdot \alpha \cdot \sigma_2$ and $x \in \mathrm{v}(\alpha) - \mathrm{v}(\sigma_1)$ then α is an input action. Symbolic traces are ranged over by $\sigma, \sigma', \ldots$.

A *symbolic configuration*, written $\langle \sigma,\ A\rangle_{\mathrm{s}}$, is a pair composed by a symbolic trace σ and an agent A, such that $\mathrm{en}(A) = \emptyset$ and $\mathrm{v}(A) \subseteq \mathrm{v}(\sigma)$. $\diamond$

Note that, due to condition (b) in the definition, e.g. $\overline{\mathsf{a}}\langle x\rangle \cdot \mathsf{a}\langle\{h\}_x\rangle$ is not a symbolic trace, while $\mathsf{a}\langle\{h\}_x\rangle \cdot \overline{\mathsf{a}}\langle x\rangle$ is. Let us now recall some standard terminology about substitutions. A substitution θ is a finite partial map from $\mathcal{V}$ to $\mathcal{M}$ and, for any object (i.e. variable, message, process, trace,...) t, we denote by $t\theta$ the result of applying θ to t. A substitution θ is a *unifier of* t_1 *and* t_2 if $t_1\theta = t_2\theta$. We denote by $\mathrm{mgu}(t_1, t_2)$ a chosen *most general unifier (mgu)* of t_1 and t_2, that is a unifier θ such that any other unifier can be written as a composition of substitutions $\theta\theta'$ for some θ'.

The transition relation on symbolic configurations, $\longrightarrow_{\mathrm{s}}$, is defined by the rules in Table 4. There, a function $\mathrm{new}_{\mathcal{V}}(\cdot)$ is assumed such that, for any given $V \subseteq_{\mathrm{fin}} \mathcal{V}$, $\mathrm{new}_{\mathcal{V}}(V)$ is a variable not in V. Note that, differently from the concrete semantics, input variables are not instantiated immediately in the input rule ($\textsc{Inp}_{\mathrm{s}}$). Rather, constraints on these variables are added as soon as they are needed, and recorded via mgu's. This may occur due to rules ($\textsc{Case}_{\mathrm{s}}$), ($\textsc{Select}_{\mathrm{s}}$) and ($\textsc{Match}_{\mathrm{s}}$). In the following example, after the first step, variable x gets instantiated to name b due to a ($\textsc{Match}_{\mathrm{s}}$)-reduction:

$$\langle \epsilon,\ \mathsf{a}(x).\,[x = b]P\rangle_{\mathrm{s}} \ \longrightarrow_{\mathrm{s}}\ \langle \mathsf{a}\langle x\rangle,\ [x = b]P\rangle_{\mathrm{s}} \ \longrightarrow_{\mathrm{s}}\ \langle \mathsf{a}\langle b\rangle,\ P[b/x]\rangle_{\mathrm{s}}\ .$$

The side condition on B' in ($\textsc{Par}_{\mathrm{s}}$) ensures that constraints are propagated across parallel components sharing variables, like in the following ($\textsc{Match}_{\mathrm{s}}$)-reduction: $\langle \sigma,\ [x = M]A \parallel \overline{\mathsf{a}}\langle x\rangle.\,B)\rangle_{\mathrm{s}} \ \longrightarrow_{\mathrm{s}}\ \langle \sigma[M/x],\ A[M/x] \parallel \overline{\mathsf{a}}\langle M\rangle.\,B[M/x]\rangle_{\mathrm{s}}$.

Whenever $\langle \sigma,\ A\rangle_{\mathrm{s}} \longrightarrow_{\mathrm{s}}^{*} \langle \sigma',\ A'\rangle_{\mathrm{s}}$ for some A', we say that $\langle \sigma,\ A\rangle_{\mathrm{s}}$ *symbolically generates* σ', and write $\langle \sigma,\ A\rangle_{\mathrm{s}} \searrow_{\mathrm{s}} \sigma'$. The relation $\longrightarrow_{\mathrm{s}}$ is finite-branching. This implies that each configuration generates a finite number of symbolic traces. It is important to stress that many symbolic traces are in fact 'garbage' – jumbled sequences of actions that cannot be instantiated to give a concrete trace. This is the case, e.g., for the trace $\mathsf{a}\langle\{x\}_k\rangle \cdot \overline{\mathsf{a}}\langle x\rangle$, which is symbolically generated by $\langle \epsilon,\ P\rangle_{\mathrm{s}}$, where $P \stackrel{\mathrm{def}}{=} \mathsf{a}(y).\,\mathsf{case}\ y\ \mathsf{of}\ \{x\}_k\ \mathsf{in}\ \overline{\mathsf{a}}\langle x\rangle.\,\mathbf{0}$. To state soundness and completeness of $\longrightarrow_{\mathrm{s}}$ w.r.t. $\longrightarrow$, we need a notion of consistency for symbolic traces, given below.

Definition 5 (solutions of symbolic traces). Given a symbolic trace σ and a ground substitution ρ, we say that ρ *satisfies* σ if $\sigma\rho$ is a trace. In this case, we also say that $\sigma\rho$ is a *solution* of σ, and that σ is *consistent*. ◇

Theorem 6 (soundness and completeness). *Let $\mathcal{C}$ be an initial configuration and s a trace. Then $\mathcal{C} \searrow s$ if and only if there is σ s.t. $\mathcal{C} \searrow_{\mathrm{s}} \sigma$ and s is a solution of σ.*

Proof: By transition induction on $\longrightarrow$ and $\longrightarrow_{\mathrm{s}}$, and then by induction on the length of traces. □

Any given configuration generates only finitely many symbolic traces. Thus, by the previous theorem, the task of checking $\mathcal{C} \models \alpha \leftrightarrows \beta$ is reduced to analysing each of these symbolic traces in turn. To do this, we need at least a method to tell whether any given symbolic trace is consistent or not. More precisely, the previous theorem reduces the problem $\mathcal{C} \models \alpha \leftrightarrows \beta$ to the following, that will be faced in the next section.

$$(\textsc{Inp}_\mathrm{s})\ \langle \sigma,\ \mathsf{a}(x).\,A\rangle_\mathrm{s} \longrightarrow_\mathrm{s} \langle \sigma \cdot \mathsf{a}\langle x\rangle,\ A\rangle_\mathrm{s}$$

$$(\textsc{Out}_\mathrm{s})\ \langle \sigma,\ \overline{\mathsf{a}}\langle M\rangle.\,A\rangle_\mathrm{s} \longrightarrow_\mathrm{s} \langle \sigma \cdot \overline{\mathsf{a}}\langle M\rangle,\ A\rangle_\mathrm{s}$$

$$(\textsc{Case}_\mathrm{s})\ \langle \sigma,\ \mathsf{case}\,\{\zeta\}_v\,\mathsf{of}\,\{x\}_u\,\mathsf{in}\,A\rangle_\mathrm{s} \longrightarrow_\mathrm{s} \langle \sigma\theta,\ A\theta\rangle_\mathrm{s} \quad \theta = \mathrm{mgu}(\{\zeta\}_v, \{x\}_u)$$

$$(\textsc{Select}_\mathrm{s})\ \langle \sigma,\ \mathsf{pair}\,\zeta\,\mathsf{of}\,\langle x, y\rangle\,\mathsf{in}\,A\rangle_\mathrm{s} \longrightarrow_\mathrm{s} \langle \sigma\theta,\ A\theta\rangle_\mathrm{s} \quad \theta = \mathrm{mgu}(\zeta, \langle x, y\rangle)$$

$$(\textsc{Match}_\mathrm{s})\ \langle \sigma,\ [\zeta = \eta]A\rangle_\mathrm{s} \longrightarrow_\mathrm{s} \langle \sigma\theta,\ A\theta\rangle_\mathrm{s} \quad \theta = \mathrm{mgu}(\zeta, \eta)$$

$$(\textsc{Par}_\mathrm{s})\ \frac{\langle \sigma,\ A\rangle_\mathrm{s} \longrightarrow_\mathrm{s} \langle \sigma',\ A'\rangle_\mathrm{s}}{\langle \sigma,\ A \parallel B\rangle_\mathrm{s} \longrightarrow_\mathrm{s} \langle \sigma',\ A' \parallel B\theta\rangle_\mathrm{s}} \quad \text{where } \sigma' = \sigma\theta, \text{ for some } \theta$$

plus symmetric version of ($\textsc{Par}_\mathrm{s}$). In the rules, it is assumed that:
(i) $x = \mathrm{new}_{\mathcal{V}}(V)$ – where V is the set of free variables in the source configuration,
(ii) $y = \mathrm{new}_{\mathcal{V}}(V \cup \{x\})$ and
(iii) $\mathrm{msg}(\sigma)\theta \subseteq \mathcal{M}$.

Table 4. Symbolic transition relation ($\longrightarrow_\mathrm{s}$)

Symbolic trace analysis problem (STAP) Given actions α, β ($\mathrm{v}(\alpha) \subseteq \mathrm{v}(\beta)$) and a symbolic trace σ, check whether or not each solution s of σ satisfies $\alpha \hookleftarrow \beta$.

We write $\sigma \models \alpha \hookleftarrow \beta$ if the answer to STAP with σ, α and β is 'yes'. Note that STAP is a non-trivial problem: one has to consider *every* solution of σ, and there may be infinitely many of them.

5 Refinement

As a first step towards devising a method for STAP, let us consider the simpler problem of checking consistency of a symbolic trace. Existence of solutions (i.e. ground instances that are traces) of a symbolic trace σ depends on the form of input actions in σ. For example, the symbolic trace $\sigma_0 = \overline{\mathsf{a}}\langle h\rangle \cdot \mathsf{b}\langle\{x\}_k\rangle$ ($h \neq k$) has no solution, because no matter which m is substituted for x, we have $\{h\} \not\vdash \{m\}_k$. On the contrary, in $\sigma_1 = \overline{\mathsf{c}}\langle k\rangle \cdot \mathsf{c}\langle\{x\}_k\rangle$, instantiating x with any environmental name $\underline{a} \in \mathcal{EN}$ will give a solution, because $\{k\} \vdash \{\underline{a}\}_k$. Yet a different case is $\sigma_2 = \overline{\mathsf{c}}\langle\{a\}_k\rangle \cdot \overline{\mathsf{c}}\langle\{b\}_k\rangle \cdot \mathsf{c}\langle\{x\}_k\rangle$. Since $\{\{a\}_k, \{b\}_k\} \not\vdash k$, there are only two ways of getting a solution of σ_2: to unify $\{x\}_k$ with either $\{a\}_k$ or $\{b\}_k$. These examples suggest that it should be possible to check consistency of a symbolic trace by gradually instantiating it until a trace is obtained. We will call this process *refinement*. In order to formalize this concept, we first need to lift the definition of 'trace' to the non-ground case. This requires a few more notations and concepts.

In refinement, we shall consider both ordinary variables and *marked* variables; roughly, the latters can only be instantiated to messages that the environment

can produce. This is made precise in the sequel. We consider a new set $\widehat{\mathcal{V}}$ of marked variables, which is in bijection with $\mathcal{V}$ via a mapping $\hat{\cdot}$: thus variables $x, y, z, \ldots$ have marked versions $\hat{x}, \hat{y}, \hat{z}, \ldots$. Marked messages are messages that may also contain marked variables, and marked symbolic traces are defined similarly. The deduction relation $\vdash$ is extended to marked messages by adding the new axiom

$$(\textsc{Mvar})\ \frac{}{S \vdash \hat{x}} \quad \hat{x} \in \widehat{\mathcal{V}}$$

to the system of Table 2. For any $\hat{x}$ and any sequence σ, we denote by $\sigma\backslash\hat{x}$ the longest prefix of σ not containing $\hat{x}$. The satisfaction relation is extended to marked symbolic traces as follows:

Definition 7. Let σ be a marked symbolic trace and ρ be a ground substitution. We say that ρ *satisfies* σ if $\sigma\rho$ is a trace and, for each $\hat{x} \in \mathrm{v}(\sigma)$, it holds that $(\sigma\backslash\hat{x})\rho \vdash \rho(\hat{x})$. We also say that $\sigma\rho$ is a *solution* of σ, and that σ is *consistent*. ◇

The terminology introduced above agrees with Definition 5 when σ does not contain marked variables. We can give now the definition of *solved form*, that lifts the definition of trace to the non-ground case (note that this definition is formally the same as Def. 2)

Definition 8 solved forms. Let σ be a marked symbolic trace. We say σ is in *solved form (sf)* if for every σ_1, $\mathsf{a}\langle M\rangle$ and σ_2 s.t. $\sigma = \sigma_1 \cdot \mathsf{a}\langle M\rangle \cdot \sigma_2$ it holds that $\sigma_1 \vdash M$. ◇

Solved forms are consistent: the next lemma gives us a specific way to instantiate a solved form so as to get a trace.

Lemma 9. *Let σ be in solved form and let ρ be any substitution from $\mathrm{v}(\sigma)$ to $\mathcal{EN}$. Then ρ satisfies σ.*

A key concept of refinement is that of decomposing a message into its irreducible components, those that cannot be further split or decrypted using the knowledge of a given σ.

Definition 10 (decomposition of messages). Let σ be a marked symbolic trace. We define the sets

- $\mathbf{I}(\sigma) \stackrel{\text{def}}{=}$
  $\{M \mid \sigma \vdash M$ and either $M \in \mathcal{LN} \cup \mathcal{V}$ or $M = \{N\}_u$ for some u s.t. $\sigma \nvdash u\}$.
- $[M]_\sigma$ by induction on M as follows:

$$[u]_\sigma = \{u\} - (\widehat{\mathcal{V}} \cup \mathcal{EN})$$
$$[\langle M, N\rangle]_\sigma = [M]_\sigma \cup [N]_\sigma$$
$$[\{M\}_u]_\sigma = \begin{cases} \{\,\{M\}_u\,\} & \text{if } \sigma \nvdash u \\ [M]_\sigma & \text{if } \sigma \vdash u \end{cases}$$

◇

The irreducible components of σ, $\mathbf{I}(\sigma)$, are the building blocks of messages that can be produced by σ. This is the content of the next proposition, that makes the relationship among $\mathbf{I}(\sigma)$, $[M]_\sigma$ and $\vdash$ precise.

Proposition 11. *Let σ be a marked symbolic trace. Then $\sigma \vdash M$ if and only if $[M]_\sigma \subseteq \mathbf{I}(\sigma)$.*

There are two points worth noting with respect to the above proposition. First, $\mathbf{I}(\sigma)$ is finite and can be easily computed by an iterative procedure, thus the proposition gives us an effective method to decide $\sigma \vdash M$; this also implies that the set of solved forms is decidable. Second, the proposition suggests a strategy for refining a generic σ to a solved form: for any input message M in σ, one tries to make the condition $[M]_{\sigma'} \subseteq \mathbf{I}(\sigma')$ true, for the appropriate prefix σ' of σ. We are now ready to define refinement formally. In the sequel, we shall use the following notations. We write '$t \in_\theta S$' for: there is $t' \in S$ s.t. $\theta = \mathrm{mgu}(t, t')$; when $\widetilde{y}$ is a set of variables, we denote by $[\widetilde{y}/\widetilde{\hat{y}}]$ the substitution that for each $x \in \widetilde{y}$ maps $\hat{x}$ to x.

Let σ be a marked symbolic trace, and assume $\sigma = \sigma' \cdot \mathsf{a}\langle M\rangle \cdot \sigma''$, where σ' is the longest prefix of σ that is in solved form. Assume $N \in [M]_{\sigma'} - \mathbf{I}(\sigma')$.

$$(\textsc{Ref}_1)\ \frac{N \notin \mathcal{V},\ N \in_\theta \mathbf{I}(\sigma'),\ \widetilde{y} = \{x \mid \hat{x} \in \mathrm{v}(\sigma) \text{ and } (\sigma\theta)\backslash\hat{x} \text{ is shorter than } \sigma\backslash\hat{x}\}}{\sigma \ \succ\ \sigma\theta[\widetilde{y}/\widetilde{\hat{y}}]}$$

$$(\textsc{Ref}_2)\ \frac{N = x \text{ or } N = \{N'\}_x}{\sigma \ \succ\ \sigma[\hat{x}/x]}$$

Table 5. Refinement ($\succ$)

Definition 12 (refinement). We let *refinement*, written $\succ$, be the least binary relation over marked symbolic traces generated by the two rules in Table 5. $\diamond$

Rule ($\textsc{Ref}_1$) implements the basic step: an element N in the decomposition of M gets instantiated, via θ, to an irreducible component of some past message (to be found in $\mathbf{I}(\sigma')$). E.g., consider again the above $\sigma_2 = \overline{\mathsf{c}}\langle\{a\}_k\rangle \cdot \overline{\mathsf{c}}\langle\{b\}_k\rangle \cdot \mathsf{c}\langle\{x\}_k\rangle$: its possible refinements are $\sigma_2 \succ \sigma_2[a/x]$ and $\sigma_2 \succ \sigma_2[b/x]$, and the refined traces are in sf. By rule ($\textsc{Ref}_2$), one may choose to mark a variable x, that will be considered as part of the environment's knowledge in subsequent refinement. Sometimes marked variables need to be 'unmarked' back to variables, and this is achieved via the renaming $[\widetilde{y}/\widetilde{\hat{y}}]$ in ($\textsc{Ref}_1$).[2]

[2] Unmarking of $\hat{x}$ occurs in a ($\textsc{Ref}_1$)-step if the prefix $\sigma\backslash\hat{x}$ gets shorter, like in: $\overline{\mathsf{a}}\langle\{a\}_k\rangle \cdot \mathsf{a}\langle\hat{z}\rangle \cdot \overline{\mathsf{a}}\langle\{\hat{z}\}_h\rangle \cdot \overline{\mathsf{a}}\langle k\rangle \cdot \mathsf{a}\langle\{\hat{x}\}_{\hat{y}}\rangle \cdot \mathsf{a}\langle\{\{\hat{x}\}_{\hat{y}}\}_h\rangle \ \succ\ \overline{\mathsf{a}}\langle\{a\}_k\rangle \cdot \mathsf{a}\langle\{x\}_y\rangle \cdot \overline{\mathsf{a}}\langle\{\{x\}_y\}_h\rangle \cdot \overline{\mathsf{a}}\langle k\rangle \cdot \mathsf{a}\langle\{x\}_y\rangle \cdot \mathsf{a}\langle\{\{x\}_y\}_h\rangle$, where $\{\{\hat{x}\}_{\hat{y}}\}_h$ gets unified with $\{\hat{z}\}_h$.

Refinement is repeatdly applied until some solved form is reached. It is important to realize that the reflexive and transitive closure $(\succ)^*$ is a non-deterministic relation, and that not all sequences of refinement lead to a solved form. However, the set of possible solved forms reachable from σ completely characterizes the set of solutions of σ. Formally, for any symbolic trace σ, we let $\mathbf{SF}(\sigma) \stackrel{\text{def}}{=} \{\sigma' \mid \sigma\ (\succ)^*\ \sigma'$ and σ' is in sf$\}$. Then we have the following theorem

Theorem 13 (characterization of solutions). *Let σ be a symbolic trace and s a trace. Then s is a solution of σ if and only if s is a solution of some $\sigma' \in \mathbf{SF}(\sigma)$.*

By the above theorem and Lemma 9, we obtain:

Corollary 14. *A symbolic trace σ is consistent if and only if $\mathbf{SF}(\sigma) \neq \emptyset$.*

Note that $\mathbf{SF}(\sigma)$ can be effectively computed, and is always finite, as: (a) $\succ$ is finitely-branching relation, and (b) infinite sequences of refinement steps cannot arise. As to the latter point, note that, since each (REF_1)-step eliminates at least one variable, any sequence of refinement steps can contain only finitely many (REF_1)-steps, after the last of which rule (REF_2) can only be applied a finite number of times. Thus, computing $\mathbf{SF}(\sigma)$ gives a method to decide consistency of σ. This also suggests a method for solving STAP. As an example, suppose that we want to check the property $\sigma \models \bot \hookleftarrow \alpha$, that is, no solution of σ contains an instance of α. Then we can proceed as follows: for each action γ in σ, we check whether there is a mgu θ that unifies γ and α; if such a θ does not exist, or if it exists but $\sigma\theta$ is not consistent (i.e. $\mathbf{SF}(\sigma\theta) = \emptyset$, Corollary 14), then the property is true, otherwise it is not. Considering the general case $\alpha \hookleftarrow \beta$ leads us to the next theorem, which gives us an effective method to check $\sigma \models \alpha \hookleftarrow \beta$. Its proof relies on Theorem 13 and on Lemma 9, plus routine calculations on mgu's.

Theorem 15 (a method for STAP). *Let σ be a symbolic trace and let $pr = \alpha \hookleftarrow \beta$, where $\mathrm{v}(\alpha) \subseteq \mathrm{v}(\beta)$ and $\mathrm{v}(\beta) \cap \mathrm{v}(\sigma) = \emptyset$. Then $\sigma \models pr$, if and only if the following is true: for each θ such that $\alpha \in_\theta \mathrm{act}(\sigma)$ and for each $\sigma' \in \mathbf{SF}(\sigma\theta)$, say $\sigma' = \sigma\theta\theta'$, it holds that $\alpha\theta\theta'$ occurs prior to $\beta\theta\theta'$ in σ'.*

The above theorem immediately yields decidability of STAP, because there are finitely many mgu's θ to consider (at most one for each action in σ), and $\mathbf{SF}(\sigma)$ can be effectively computed. This result lifts of course to configurations.

Corollary 16 (decidability). *Let $\mathcal{C}$ be an initial configuration and $\alpha \hookleftarrow \beta$ be a property. It is decidable whether $\mathcal{C} \models \alpha \hookleftarrow \beta$ or not.*

PROOF: Compute $\{\sigma \mid \mathcal{C}$ symbolically generates $\sigma\}$, which is finite, and then check whether or not for each σ in this set it is the case that $\sigma \models \alpha \hookleftarrow \beta$, which can be effectively done. The thesis is a consequence of Theorem 6. □

In a practical implementation, rather than generating the whole set of symbolic traces of a given configuration and then check the property, it is more convenient to check the single symbolic traces as soon as they are generated in an 'on-the-fly' way.

6 Restriction

We consider extending the base language via the *restriction* operator $(\mathsf{new}\, a)A$, where $a \in \mathcal{LN}$ and A an agent; $(\mathsf{new}\, a)$ is binder for name a. The intended meaning of $(\mathsf{new}\, a)A$ is that a new name a is created, which is private to A. The concrete and symbolic rules for restriction are given below. A function $\mathrm{new}_{\mathcal{LN}}(\cdot)$ is assumed that, for any set of names $V \subseteq_{\mathrm{fin}} \mathcal{LN}$, yields a local name $a \notin V$.

$$(\textsc{New})\ \langle s,\ (\mathsf{new}\, a)P\rangle \longrightarrow \langle s,\ P\rangle \qquad a = \mathrm{new}_{\mathcal{LN}}(V)$$
$$(\textsc{New}_{\mathrm{s}})\ \langle \sigma,\ (\mathsf{new}\, a)A\rangle_{\mathrm{s}} \longrightarrow_{\mathrm{s}} \langle \sigma,\ A\rangle_{\mathrm{s}} \quad a = \mathrm{new}_{\mathcal{LN}}(V)$$

In both rules, V is the set of local names occurring free in the source configuration. Note that the side-condition on name a is always met modulo alpha-renaming. A change is required in the rules for parallel composition $A \parallel B$, both in the concrete and in the symbolic case: in the conclusion, an additional renaming $[b/a]$ (where $a = \mathrm{new}_{\mathcal{LN}}(\mathrm{ln}(\sigma, A))$ and $b = \mathrm{new}_{\mathcal{LN}}(\mathrm{ln}(\sigma, A \parallel B))$) is applied onto the target configuration: this prevents a new name a possibly created by a (New)-transition of A from clashing with free occurences of a in B (this is just the side-condition of the rule (Par) of the π-calculus [17] rephrased in our language).

7 Conclusions

We have presented a symbolic method for analysing cryptographic protocols. The method is well suited for an efficient mechanization. The word 'efficient' should be taken with a grain of salt here. The trace analysis problem is obviouvsly NP- hard [4, 10], thus pathological examples are unavoidable: formal statements on 'efficiency' are therefore hard to formulate. However, we expect the method to perform well in practical cases; this is further discussed below. Experiments conducted with a preliminary, non optimized implementation have given encouraging results [6]. Developments in the near future include an optimized implementation of the method and extension of the present results to other cryptographic primitives, like public key and hashing: this should not present conceptual difficulties, though we have not checked the details yet.

Approaches based on symbolic analysis have also been explored by Huima in [12] and Amadio and Lugiez [4]. In [12], Huima presents a symbolic semantics by which the execution of a protocol generates a set of equational constraints; only an informal description is given of the kind of equational rewriting needed to solve these constraints. Amadio and Lugiez in [4] consider a variant of the spi-calculus equipped with a symbolic semantics. Similarly to Huima's, their symbolic semantics generates equational constraints of a special form, rather than unifiers. The (rather complex) constraint-solving procedure is embedded into symbolic execution, and uses a brute-force method to resolve variables in key position (all possible instantiations of variables to names that are around are tried). These factors have a relevant impact on the size of the symbolic model. On the contrary, in our case symbolic execution and consistency check are kept separate, and this permits to keep the size of the model to a minimum. The

consistency check procedure (refinement) is invoked only when necessary, and, most important, does not use brute-force instantiation. Finally, Amadio and Lugiez encode authentication via reachability: this may add to the complexity of their method.

Model checking [9, 13, 19, 21] and theorem proving [18] seem to be among the most successful approaches to the formal analysis of security protocols. As Paulson has pointed out [18], theorem proving is intuitive, but, within it, verification is not fully automated and general completeness results are difficult to establish. On the contrary, model checking is automatic, but suffers from the state explosion problem, which requires the model to be cut down to a convenient finite size. Our paper might be regarded as an attempt at bridging the two approaches. We extract the unification mechanism underlying theorem proving and bring it on the ground of a process language (a variant of the spi-calculus) that naturally supports a notion of variable binding. This allows us to obtain precise completeness results for trace analysis.

Acknowledgements I have benefitted from stimulating discussions with Martin Abadi, Roberto Amadio, Rocco De Nicola, Marcelo Fiore and Rosario Pugliese.

References

1. M. Abadi, A.D. Gordon. A calculus for cryptographic protocols: The spi calculus. *Information and Computation*, 148(1):1-70, 1999.
2. M. Abadi, A.D. Gordon. A Bisimulation Method for Cryptographic Protocols. *Nordic Journal of Computing*, 5(4):267-303, 1998.
3. R. Amadio, S. Prasad. The game of the name in cryptographic tables. RR 3733 INRIA Sophia Antipolis. In *Proc. of Asian'00*, LNCS, 2000.
4. R.M. Amadio, S. Lugiez. On the reachability problem in cryptographic protocols. In *Proc. of Concur'00*, LNCS, 2000. Full version: RR 3915 Inria Sophia Antipolis.
5. M. Boreale. Symbolic analysis of cryptographic protocols in the spi-calculus. Manuscript, 2000. Available at `http://www.dsi.unifi.it/~boreale/papers.html`.
6. M. Boreale. STA: a tool for trace analysis of cryptographic protocols. ML object code and examples, 2001. Available at `http://www.dsi.unifi.it/~boreale/tool.html`.
7. M. Boreale, R. De Nicola, R. Pugliese. Proof Techniques for Cryptographic Processes. In *Proc. of LICS'99*, IEEE Computer Society Press, 1999. Full version to appear in *SIAM Journal on Computing.*
8. M. Burrows, M. Abadi, R.M. Needham. A logic of authentication. *Proc. of the Royal Society of London*, 426:233–271, 1989.
9. E.M. Clarke, S. Jha, W. Marrero. Using state exploration and a natural deduction style message derivation engine to verify security protocols. In *Proc. of the IFIP Working Conference on Programming Concepts and Methods (PROCOMET)*, 1998.
10. N. Durgin, P. Lincoln, J. Mitchell, A. Scedrov. Undecidability of bounded security protocols. In *Proc. of Workshop on Formal Methods and Security Protocols*, Trento, 1999.
11. D. Dolev, A.C. Yao. On the security of public key protocols. In *IEEE Transactions on Information Theory* 29(2):198–208, 1983.

12. A. Huima. Efficient infinite-state analysis of security protocols. In *Proc. of Workshop on Formal Methods and Security Protocols*, Trento, 1999.
13. G. Lowe. Breaking and Fixing the Needham-Schroeder Public-Key Protocol Using FDR. In *Proceedings of TACAS'96*, (T. Margaria, B. Steffen, Eds.), *LNCS* 1055, pp. 147-166, Springer-Verlag, 1996.
14. G. Lowe. A Hierarchy of Authentication Specifications. In *10th IEEE Computer Security Foundations Workshop*, IEEE Computer Society Press, 1997.
15. G. Lowe. Towards a completeness result for model checking of security protocols. In *11th Computer Security Foundations Workshop*, IEEE Computer Society Press, 1998.
16. D. Marchignoli, F. Martinelli. Automatic verification of cryptographic protocols through compositional analysis techniques. In *Proc. of TACAS99*, LNCS 1579:148–163, 1999.
17. R. Milner, J. Parrow, D. Walker. A calculus of mobile processes, (Part I and II). *Information and Computation*, 100:1-77, 1992.
18. L.C. Paulson. Proving Security Protocols Correct. In *Proc. of LICS'99*, IEEE Computer Society Press, 1999.
19. A.W. Roscoe. Modelling and verifying key-exchange using CSP and FDR. In *8th Computer Security Foundations Workshop*, IEEE Computer Society Press, 1995.
20. A.W. Roscoe. Proving security protocols with model checkers by data independent techniques. In *11th Computer Security Foundations Workshop*, IEEE Computer Society Press, 1998.
21. S. Schneider. Verifying Authentication Protocols in CSP. *IEEE Transactions on Software Engineering*, 24(8):743-758, 1998.
22. S. Stoller. A reduction for automated verification of security protocols. In *Proc. of Workshop on Formal Methods and Security Protocols*, Trento, 1999.

This article was processed using the LaTeX macro package with LLNCS style