A Framework for the Analysis of Security Protocols

Michele Boreale[1] and Maria Grazia Buscemi[2]

[1] Dipartimento di Sistemi e Informatica, Università di Firenze, Italy.
[2] Dipartimento di Matematica e Informatica, Università di Catania, Italy.
boreale@dsi.unifi.it, buscemi@dmi.unict.it

Abstract. Properties of security protocols such as authentication and secrecy are often verified by explictly generating an operational model of the protocol and then seeking for insecure states. However, message exchange between the intruder and the honest participants induces a form of state explosion that makes the model infinite in principle. Building on previous work on symbolic semantics, we propose a general framework for automatic analysis of security protocols that make use of a variety of crypto-functions. We start from a base language akin to the spi-calculus, equipped with a set of *generic* cryptographic primitives. We propose a symbolic operational semantics that relies on unification and provides finite and effective protocol models. Next, we give a method to carry out trace analysis directly on the symbolic model. Under certain conditions on the given cryptographic primitives, our method is proven complete for the considered class of properties.

1 Introduction

Automatic methods for verifying properties of security protocols are very often based on explicit generation of the protocol model. The latter is then explored in order to check whether any insecure state is reachable. In particular, most methods based on finite state model-checking (see, e.g., [17, 21, 23, 26]) follow a Dolev-Yao intruder model [12], which implies that a (hostile) environment has total control over the communication network. The assumption is that the environment can store, duplicate, hide or replace any message traveling on the network, and synthesize new messages by pairing, encryption and decryption of those already known.

To make standard finite-state model checking applicable to protocol analysis, two simplifying requirements are necessary: (a) there is a bound on the number of protocol runs, and (b) there is a bound on the number of possible messages the intruder can generate and send to honest participants at any moment. Discarding one of these two requirements leads to infinite models. Also, these bounds have to be chosen very carefully: due to the combinatorics of message generation,

This work has been partially supported by EU within the FET - Global Computing initiative, projects MIKADO and PROFUNDIS and by MIUR project NAPOLI.

the size of the model tends to explode as multiple principals and data values are considered. It is known that discarding requirement (a) leads to undecidability of protocol analysis, unless severe restrictions are imposed on the analyzed protocols (see e.g.[5, 11, 13]). Rather, it is less clear to what extent requirement (b) can be relaxed, while preserving generality of protocol format, decidability and effectiveness.

For the case of shared-key encryption, approaches alternative to finite-state model checking have been pursued, based on notions of symbolic execution [4, 7]. In this paper, we extend the symbolic semantics given in [7] to a general framework for the treatment of *generic* cryptographic primitives. We then spell out sufficient conditions on these primitives, under which verification can be performed symbolically and, hence, effectively. As an application, we consider an instance of this framework whose primitives correspond to the most common cryptographic funtions (shared and public key, digital signature and hashing).

As a base language, we consider a dialect of the spi-calculus [3]. Unlike the original spi-calculus and the languages in [4, 7], where cryptographic functions are modelled as process operators, we adopt a uniform process syntax, much as that of [2], and consider a generic signature Σ of cryptographic functions. The latter may include constructors and destructors for various cryptographic operations, akin to [1]. The meaning of Σ-terms is provided by an evaluation relation $\downarrow$ that maps terms to values (or *messages*). On top of the evaluation relation, we introduce a deduction relation $\vdash$ that describes how the environment synthesize new messages from known ones. Protocol properties are formalised as correspondence assertions of the kind "every execution of action α must be preceded by some execution of action β", for given α and β.

The ("concrete") operational semantics of this language is, as expected, infinitary, because each input action gives rise to infinitely many transitions. To overcome this problem, we also define a symbolic operational semantics that is finitely branching and yields finite models of protocols with a fixed number of participants. Then, we give a method to carry out trace analysis directly on the symbolic traces.

Next, we provide some reasonable conditions on the given cryptosystem under which we can prove that the method is sound and complete with respect to the concrete semantics. In other words, every attack detected in the symbolic model corresponds to some attack in the concrete one, and vice-versa. Thus our symbolic method makes no approximation with respect to the infinitary, concrete model. For instance, type-dependent flaws (see e.g. [15]), which usually escape finite-state analysis, with our approach naturally emerge when present. The method is rather efficient in practice, because in the symbolic model there is no state-explosion induced by message exchange: every input action gives rise exactly to one symbolic transition. Experimentation with STA [27], a prototype tool based on this method, has given encouraging results [8].

A summary of our paper follows. The general framework is introduced in Section 2. Symbolic semantics, and its relationship to the concrete one, are the subject of Section 3. The verification method is introduced and discussed in Section 4 and applied to a specific protocol in Section 5. Throughout the paper, we shall consider public-key cryptography as a running example; other common crypto-primitives are discussed in Section 6. A few concluding remarks and directions for future work are in Section 7.

2 A general framework

In this section, we present the main ingredients of our framework. We introduce the concept of *frame*, i.e., a structure consisting of a signature Σ, a set of (legal) messages and a function that evaluates terms to messages. Then, we generalize the notions of traces, configurations and security property, introduced in [7].

2.1 Frames

We consider two countable disjoint sets of *names* $m, n, \ldots \in \mathcal{N}$ and variables $x, y, \ldots \in \mathcal{V}$. The set $\mathcal{N}$ is in turn partitioned into a countable set of *local names* $a, b, \ldots \in \mathcal{LN}$ and a countable set of *environmental names* $\underline{a}, \underline{b}, \ldots \in \mathcal{EN}$: these two sets represent the basic data (keys, nonces,...) initially known to a process and to the environment, respectively. The set $\mathcal{N} \cup \mathcal{V}$ is ranged over by letters $u, v, \ldots$. Given a signature Σ of function symbols $f, g, \ldots$, each coming with its arity (constants have arity 0), we denote by $\mathcal{E}_\Sigma$ the algebra of terms (or *expressions*) on $\mathcal{N} \cup \mathcal{V} \cup \Sigma$, given by the grammar:

$$\zeta, \eta \quad ::= \quad u \quad | \quad f(\widetilde{\zeta})$$

where $\widetilde{\zeta}$ is a tuple of terms of the expected length. A *term context* $C[\cdot]$ is a term with a hole that can be filled with any expression ζ, thus yielding an expression $C[\zeta]$.

Definition 1 (frame). *A* frame $\mathcal{F}$ *is a triple* $(\Sigma, \mathcal{M}, \downarrow)$*, where:*

- Σ *is a signature;*
- $\mathcal{M} \subseteq \mathcal{E}_\Sigma$ *is a set of* messages $M, N, \ldots$*;*
- $\downarrow \subseteq \mathcal{E}_\Sigma \times \mathcal{M}$ *is an* evaluation relation.

In the sequel, we write $\zeta \downarrow M$ for $(\zeta, M) \in \downarrow$ and say that ζ *evaluates to* M. An evaluation relation evaluates expressions to messages. In typical frame instances the relation $\downarrow$ will be both a function and a congruence with respect to the operations in Σ, but we need not to assume these facts in the general framework.

Below, we define a deduction relation which expresses how the environment can generate new messages starting from an initial set of messages S. Unlike other approaches, our definition of deduction relation is not given by a set of deductive rules. Rather, we make use of the set $\mathcal{H}(S)$, which consists of all the expressions inductively built by applying functions of Σ to elements of S and of $\mathcal{EN}$. We denote by $\mathcal{P}_f(X)$ the set of finite subsets of X.

Definition 2 (deduction relation). *For $\mathcal{F} = (\Sigma, \mathcal{M}, \downarrow)$ a frame and $S \subseteq \mathcal{M}$, the set $\mathcal{H}_{\mathcal{F}}(S)$ is inductively defined by the following rules:*

$$\begin{aligned} \mathcal{H}^0_{\mathcal{F}}(S) &\triangleq S \cup \mathcal{EN} \\ \mathcal{H}^{i+1}_{\mathcal{F}}(S) &\triangleq \mathcal{H}^i_{\mathcal{F}}(S) \cup \{ f(\widetilde{\zeta}) : \ f \in \Sigma,\ \widetilde{\zeta} \subseteq \mathcal{H}^i_{\mathcal{F}}(S) \} \\ \mathcal{H}_{\mathcal{F}}(S) &\triangleq \bigcup_{i \geq 0} \mathcal{H}^i_{\mathcal{F}}(S) \end{aligned}$$

The deduction relation $\vdash_{\mathcal{F}} \subseteq \mathcal{P}_f(\mathcal{M}) \times \mathcal{M}$ *is defined by:*

$$S \vdash_{\mathcal{F}} M \quad \overset{\triangle}{\Leftrightarrow} \quad \exists\, \zeta \in \mathcal{H}_{\mathcal{F}}(S) : \zeta \downarrow M$$

A message M is deducible *from S if $S \vdash_{\mathcal{F}} M$.*

When no confusion arises, we simply write $\mathcal{H}(S)$ for $\mathcal{H}_{\mathcal{F}}(S)$ and $\vdash$ for $\vdash_{\mathcal{F}}$.
Throughout the paper, we study the case of public key encryption, which serves as a running example of application of our method.

Example: Public Key Encryption (1) Let us consider a system where primitives for pairing and public key encryption of messages can be arbitrarily nested, but non-atomic keys are forbidden. The frame $\mathcal{F}_{pk} = (\Sigma, \mathcal{M}, \downarrow)$ is defined in Table 1. The functions of Σ are: generation of public $((\cdot)^+)$ and private $((\cdot)^-)$ keys, encryption with a public key ($\{\![\cdot]\!\}_{(\cdot)}$), decryption using a private key ($\mathsf{dec}^{\mathsf{pk}}_{(\cdot)}(\cdot)$), pairing ($\langle\cdot,\cdot\rangle$) and selection ($\pi_i(\cdot)$). Public and private keys are represented by u^+ and u^-, respectively. Names and variables can be used to build compound messages via public-key encryption and pairing. In particular, $\{\![M]\!\}_{m^+}$ represents the message obtained by encrypting M under m^+. The definition of evaluation relation makes use of an auxiliary relation $\rightsquigarrow$, that models the mechanisms of public key encryption under the perfect cryptography assumption. As an example of deducible message in $\mathcal{F}_{pk}$, if $S = \{\ \{\![\langle a, b\rangle]\!\}_{k^+}, k^-\ \}$ then $S \vdash a$, since $\zeta = \pi_1(\mathsf{dec}^{\mathsf{pk}}_{k^-}(\{\![\langle a, b\rangle]\!\}_{k^+})) \in \mathcal{H}(S)$ and $\zeta \downarrow a$. Note that, whatever S, the set of messages deducible from S is infinite.

2.2 Processes

Syntax As a base language, we consider a variant of the spi-calculus [3], parametrized by an arbitrary frame $\mathcal{F}$ (for readability, in the notation we omit explicit reference to $\mathcal{F}$). The syntax of *agent expressions*, in $\mathcal{A}$, is reported in Table 2. We consider a set $\mathcal{L}$ of *labels* which is ranged over by $\mathsf{a}, \mathsf{b}, \ldots$. The main difference from the spi-calculus is that, here, input and output labels $(\mathsf{a}, \mathsf{b}, \ldots)$ must not be regarded as channels – following the Dolev-Yao model, we assume just one public network – but, rather, as 'tags' attached to process actions for ease of reference. Also, we have a single construct (let) for evaluating expressions that replaces the ad-hoc constructs found in the spi-calculus for encryption, decryption and other cryptographic operations. We do not consider restriction: it

$$\Sigma \quad = \{(\cdot)^+,\quad (\cdot)^-,\quad \{\![\cdot]\!\}_{(\cdot)},\quad \langle\cdot,\cdot\rangle,\quad \pi_i(\cdot)\quad (i=1,2),\quad \mathsf{dec}^{\mathsf{pk}}_{(\cdot)}(\cdot)\}$$

Signature

$$M, N \quad ::= u \;\mid\; u^+ \;\mid\; u^- \;\mid\; \{\![M]\!\}_{u^+} \;\mid\; \langle M, N\rangle$$

Messages

$$\pi_i(\langle M_1, M_2\rangle) \rightsquigarrow M_i \quad (i=1,2)$$

$$\mathsf{dec}^{\mathsf{pk}}_{u^-}(\{\![M]\!\}_{u^+}) \rightsquigarrow M$$

$$\frac{\zeta \rightsquigarrow \zeta'}{C[\zeta] \rightsquigarrow C[\zeta']} \qquad \zeta \downarrow M \quad \stackrel{\triangle}{\Leftrightarrow} \quad \zeta \rightsquigarrow^* M$$

Evaluation

Table 1. $\mathcal{F}_{pk}$, a frame for public key encryption

might be easily accommodated but it has little semantical significance, in the absence of replication/recursion.

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 $\ln(A)$ and $\mathrm{en}(A)$, respectively. A process P is *initial* if $\mathrm{en}(P) = \emptyset$.

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

The occurrences of variables x and y are bound.

Table 2. Syntax for agents

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 model the state of the system as a pair $\langle s,\ P\rangle$, where s records the current environment's knowledge (i.e., the sequence of messages the environment has "seen" on the network up to a given moment) and P is a process term. 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 $\mathrm{act}(s)$ and $\mathrm{msg}(s)$ the set of actions and messages, respectively, appearing in s. A string s is *closed* if $\mathrm{v}(s) = \emptyset$ and *initial* if $\mathrm{en}(s) = \emptyset$. In what follows, we write '$s \vdash M$' for $\mathrm{msg}(s) \vdash M$.

We, now, 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 a process (input message) can be synthesized from the knowledge the environment has previously acquired. In *configurations*, the latter is explicitly recorded.

Definition 3 (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 $\mathcal{C}, \mathcal{C}', \ldots$.*

(Inp)	$\langle s,\ \mathsf{a}(x).\,P\rangle \longrightarrow \langle s \cdot \mathsf{a}\langle M\rangle,\ P[{}^{M}\!/x]\rangle$	$s \vdash M,\ M$ closed
(Out)	$\langle s,\ \overline{\mathsf{a}}\langle\zeta\rangle.\,P\rangle \longrightarrow \langle s \cdot \overline{\mathsf{a}}\langle M\rangle,\ P\rangle$	$\zeta \downarrow M,\ M$ closed
(Let)	$\langle s,\ \mathsf{let}\,y = \zeta\,\mathsf{in}\,P\rangle \longrightarrow \langle s,\ P[{}^{M}\!/y]\rangle$	$\zeta \downarrow M,\ M$ closed
(Match)	$\langle s,\ [\zeta = \eta]P\rangle \longrightarrow \langle s,\ P\rangle$	$\zeta \downarrow M,\ \eta \downarrow N,\ M = N$
(Par)	$\dfrac{\langle s,\ P\rangle \longrightarrow \langle s',\ P'\rangle}{\langle s,\ P \parallel Q\rangle \longrightarrow \langle s',\ P' \parallel Q\rangle}$	
plus symmetric version of (Par).		

Table 3. Rules for the 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 configuration's first component. Rule (Inp) makes the transition relation infinitely-branching, as M ranges over the infinite set $\{M : s \vdash M,\ M \text{ closed}\}$. In rule (Out), ζ is evaluated before the action takes place. By rule (Let), the evaluation of ζ replaces any occurrence of y in P. No handshake communication is provided : all messages go through the environment (rule (Par)).

2.3 Properties

We express security properties of a protocol in terms of the traces it generates. In particular, we focus on correspondence assertions of the kind 'for every generated trace, whenever action β occurs in the trace, then action α must have occurred at some previous point in the trace'. 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'. A substitution θ in a frame $\mathcal{F}$ is a finite partial map from $\mathcal{V}$ to the set of messages $\mathcal{M}$ of frame $\mathcal{F}$ such that $\theta(x) \neq x$, for each variable x. For any object t (i.e. variable, message, process, trace,...), we denote by $t\theta$ the result of simultaneously replacing each $x \in \mathrm{v}(t) \cap \mathrm{dom}(\theta)$ by $\theta(x)$. We let ρ range over ground substitutions, i.e. substitutions that map variables to closed messages.

Definition 4 (satisfaction relation). *Let α and β be actions and s be a trace. We say that α* occurs prior to β in s *if whenever $s = s' \cdot \beta \cdot s''$ then $\alpha \in \mathrm{act}(s')$. For $\mathrm{v}(\alpha) \subseteq \mathrm{v}(\beta)$, we write $s \models \alpha \hookleftarrow \beta$, and say 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 $\alpha \hookleftarrow \beta$.*

Assertions $\alpha \hookleftarrow \beta$ can express interesting secrecy and authentication properties. As an example, in the final step of many key-establishment protocols, a principal A sends a message of the form $\{N\}_k$ to a responder B, where $\{N\}_k$ is obtained by encrypting some authentication information N under a newly established shared-key k. Our scheme permits expressing that *every* message encrypted with k that is accepted by B during the execution of the protocol indeed originates from A, i.e. that B is really talking to A, and that k is authentic. If we denote by $\mathsf{final}_\mathsf{A}$ and $\mathsf{final}_\mathsf{B}$ the labels attached to A's and B's final action, respectively, then the above property might be formalized as an assertion $\mathsf{final}_\mathsf{A}\langle\{x\}_k\rangle \hookleftarrow \mathsf{final}_\mathsf{B}\langle\{x\}_k\rangle$, for x a variable. An extended example is given in Section 5. In practice, all forms of authentication in Lowe's hierarchy [18] are captured by this scheme, except the most demanding one that requires one-to-one bijection between α's and β's. However, we expect our scheme can be easily adjusted to include this stronger form, by requiring that each β is preceded by *exactly* one occurrence of α.

Another property that can be set within our frame is *secrecy* in the style of [6]. In this case, it is convenient to fix a conventional 'absurd' action $\perp$ that is nowhere used in agent expressions. Thus, the formula $\perp \hookleftarrow \alpha$ means that action α should never take place. Now, the fact that a protocol, say P, does not leak a sensible datum, say d, can be expressed also by saying that the adversary will never be capable of synthesizing d. This can be formalized by extending the protocol to include a 'guardian' that at any time picks up one message from the network, $P \parallel \mathsf{g}(x).\,\mathbf{0}$, and then requiring that this guardian will never receive d, that is, $\langle \epsilon,\ P \parallel \mathsf{g}(x).\,\mathbf{0}\rangle \models \perp \hookleftarrow \mathsf{g}\langle d\rangle$.

3 Symbolic semantics

The symbolic semantics we present in this section is based on the notion of symbolic frame. The latter is essentially a frame equipped with an additional symbolic evaluation relation, which is in agreement with its concrete counterpart.

Formally, let us denote by Subst the set of all substitutions in a given frame. 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 of t_1 and t_2 such that any other unifier is a composition of substitutions θ and θ', written $\theta\theta'$, for some θ'. Also, for t_1, t_1', t_2, t_2' terms, $\mathrm{mgu}(t_1 = t_1', t_2 = t_2')$ stands for $\mathrm{mgu}(t_2\theta, t_2'\theta)$, where $\theta = \mathrm{mgu}(t_1, t_1')$, if such mgu's exist.

Definition 5 (symbolic frame). *A* symbolic frame *is a pair* $\mathcal{F}^s = (\mathcal{F}, \downarrow_s)$*, where* $\mathcal{F}$ *is a frame, and* $\downarrow_s \subseteq \mathcal{E}_\Sigma \times \mathsf{Subst} \times \mathcal{M}$ *is a* symbolic evaluation relation *(we write* $\zeta \downarrow_\theta M$ *for* $(\zeta, \theta, M) \in \downarrow_s$*) such that, for any expression* ζ *and ground substitution* ρ *with* $\mathrm{v}(\zeta) \subseteq \mathrm{dom}(\rho)$*:*

(a) If $\zeta\rho \downarrow M$*, then there exist* N, θ, ρ_0 *such that* $\zeta \downarrow_\theta N$*,* $\rho = \theta\rho_0$ *and* $M = N\rho_0$*, and*
(b) If $\zeta \downarrow_\theta N$*,* $\rho = \theta\rho_0$*, for some* ρ_0*, and* $N\rho_0 \in \mathcal{M}$*, then* $\zeta\rho \downarrow N\rho_0$*.*

Example: Public Key Encryption (2) The symbolic frame $\mathcal{F}^s_{pk}$ is defined as the pair $(\mathcal{F}_{pk}, \downarrow_s)$, where $\downarrow_s$ is the reflexive and transitive closure of the relation $(\rightsquigarrow_s)$, as given in Table 4. $\mathcal{F}^s_{pk}$ is indeed a symbolic frame: Conditions (a) and (b) in Definition 5 are proven by straightforward induction on ζ.

$$\mathsf{dec}^{\mathsf{pk}}_N(M) \overset{\theta}{\rightsquigarrow}_s x_1\,\theta \qquad\qquad \theta = \mathrm{mgu}(M = \{\![x_1]\!\}_{x_2^+}, N = x_2^-)$$

$$\pi_i(M) \overset{\theta}{\rightsquigarrow}_s x_i\,\theta \qquad (i = 1, 2) \qquad \theta = \mathrm{mgu}(M, \langle x_1, x_2 \rangle)$$

$$\{\![M]\!\}_x \overset{\theta}{\rightsquigarrow}_s \{\![M\theta]\!\}_{x^+} \qquad\qquad \theta = [{}^{x^+}\!/x]$$

$$\frac{\zeta \overset{\theta}{\rightsquigarrow}_s \zeta'}{C[\zeta] \overset{\theta}{\rightsquigarrow}_s C\theta[\zeta']}$$

$$\zeta \downarrow_\theta M \overset{\triangle}{\Leftrightarrow} \zeta \overset{\theta_1}{\rightsquigarrow}_s \cdots \overset{\theta_n}{\rightsquigarrow}_s M \text{ and } \theta = \theta_1 \cdots \theta_n$$

Variables x_1 and x_2 are chosen fresh according to some arbitrary but fixed rule.

Table 4. Symbolic Evaluation Relation $(\downarrow_s)$ for $\mathcal{F}^s_{pk}$

We now come to symbolic counterparts of traces and configurations. Note that Condition (b) below states that only the environment can introduce variables into symbolic traces.

Definition 6 (symbolic traces and configurations). *A* symbolic trace *is a string $s \in Act^*$ such that: (a) $\mathrm{en}(s) = \emptyset$, and (b) for each s_1, s_2, α and x, if $s = s_1 \cdot \alpha \cdot s_2$ and $x \in \mathrm{v}(\alpha) - \mathrm{v}(s_1)$ then α is an input action. Symbolic traces are ranged over by $\sigma, \sigma', \ldots$. A* symbolic configuration*, written $\langle \sigma,\ A\rangle_{\mathsf{s}}$, is a pair composed by a symbolic trace σ and an agent A, such that* $\mathrm{en}(A) = \emptyset$ *and* $\mathrm{v}(A) \subseteq \mathrm{v}(\sigma)$.

Note that, due to Condition (b) in the Definition 6, e.g. $\bar{\mathsf{a}}\langle x^+\rangle \cdot \mathsf{a}\langle\{[h]\}_{x^+}\rangle$ is not a symbolic trace, while $\mathsf{a}\langle\{[h]\}_{x^+}\rangle \cdot \bar{\mathsf{a}}\langle x^+\rangle$ is so. Once a symbolic frame $\mathcal{F}^s$ is fixed, configurations can be equipped with a symbolic transition relation, $\longrightarrow_{\mathsf{s}}$, as defined by the rules in Table 5 (for the sake of readability we omit any explicit reference to $\mathcal{F}^s$). There, a function $\mathrm{new}_{\mathcal{V}}(\cdot)$ is assumed such that, for any given $V \subseteq_{\mathsf{fin}} \mathcal{V}$, $\mathrm{new}_{\mathcal{V}}(V)$ is a variable not in V. Note that, differently from the concrete semantics, input variables are *not* instantiated immediately (rule (Inp$_{\mathsf{s}}$)). Rather, constraints on these variables are added as soon as they are needed, and recorded via mgu's. This may occur due to rules (Out$_{\mathsf{s}}$), (Let$_{\mathsf{s}}$) and (Match$_{\mathsf{s}}$). In the following example, after the first step, variable x gets instantiated to name b by a (Match$_{\mathsf{s}}$)-reduction:

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

Whenever $\langle \sigma,\ A\rangle_{\mathsf{s}} \longrightarrow^*_{\mathsf{s}} \langle \sigma',\ A'\rangle_{\mathsf{s}}$ for some A', we say that $\langle \sigma,\ A\rangle_{\mathsf{s}}$ *symbolically generates* σ', and write $\langle \sigma,\ A\rangle_{\mathsf{s}} \searrow_{\mathsf{s}} \sigma'$. The relation $\longrightarrow_{\mathsf{s}}$ is finitely-branching if so is $\downarrow_s$ (this is the case, e.g., for the public key frame $\mathcal{F}^s_{pk}$). In this case, 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 any concrete trace. For instance, consider process $P \triangleq \mathsf{a}(y).\,\mathsf{let}\, x = \mathsf{dec}^{\mathsf{pk}}_{k^-}(y)\,\mathsf{in}\,\bar{\mathsf{a}}\langle x\rangle.\,\mathbf{0}$. The initial configuration $\langle \epsilon,\ P\rangle_{\mathsf{s}}$ symbolically generates the trace $\mathsf{a}\langle\{[z]\}_{k^+}\rangle \cdot \bar{\mathsf{a}}\langle z\rangle$, which is inconsistent, because the environment cannot provide k^+. The problem of detecting these inconsistent traces, that might give rise to 'false positives' when checking protocol properties, will be faced in the next section.

We define below the notion of consistency. Based on it, we state a theorem that lifts the concrete-symbolic correspondence given in Definition 5 to the transition relations. Its proof is an easy transition induction on $\longrightarrow_{\mathsf{s}}$ and $\longrightarrow$.

Definition 7 (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. If this is this case, we also say that $\sigma\rho$ is a* solution *of σ, and that σ is* consistent.

Theorem 1 (concrete vs. symbolic semantics). *Let $\mathcal{F}^s$ be a symbolic frame, $\mathcal{C}$ an initial configuration and s a trace of $\mathcal{F}$. Then $\mathcal{C} \searrow s$ if and only if there is σ s.t. $\mathcal{C} \searrow_{\mathsf{s}} \sigma$ and s is a solution of σ.*

4 A verification method

In this section, we first define *regular frames*, i.e., frames for which it is possible to determine a finite *basis* for the synthesis of messages. Next, we introduce

($\textsc{Inp}_s$)	$\langle\sigma,\ \mathsf{a}(x).\,A\rangle_s \longrightarrow_s \langle\sigma\cdot\mathsf{a}\langle x\rangle,\ A\rangle_s$		
($\textsc{Out}_s$)	$\langle\sigma,\ \bar{\mathsf{a}}\langle\zeta\rangle.\,A\rangle_s \longrightarrow_s \langle\sigma\theta\cdot\bar{\mathsf{a}}\langle M\rangle,\ A\theta\rangle_s$	$\zeta\downarrow_\theta M$	
($\textsc{Let}_s$)	$\langle\sigma,\ \mathsf{let}\,y=\zeta\,\mathsf{in}\,A\rangle_s \longrightarrow_s \langle\sigma\theta,\ A\theta[^M/_y]\rangle_s$	$\zeta\downarrow_\theta M$	
($\textsc{Match}_s$)	$\langle\sigma,\ [\zeta=\eta]A\rangle_s \longrightarrow_s \langle\sigma\theta,\ A\theta\rangle_s$	$\zeta\downarrow_{\theta_1} M,\ \eta\,\theta_1\downarrow_{\theta_2} N,$ $\theta_3=\mathrm{mgu}(M\theta_2,N),$ $N\theta_3\in\mathcal{M},\ \theta=\theta_1\theta_2\theta_3$	
($\textsc{Par}_s$)	$\dfrac{\langle\sigma,\ A\rangle_s \longrightarrow_s \langle\sigma',\ A'\rangle_s}{\langle\sigma,\ A\parallel B\rangle_s \longrightarrow_s \langle\sigma',\ A'\parallel B'\rangle_s}$		

plus symmetric version of ($\textsc{Par}_s$). In the above 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 $\mathrm{msg}(\sigma)\theta\subseteq\mathcal{M}$,
(iii) in rule ($\textsc{Par}_s$), $B'=B\theta$ where $\sigma'=\sigma\theta\cdot\alpha$ or $\sigma'=\sigma\theta$.

Table 5. Rules for symbolic transition relation ($\longrightarrow_s$)

a *refinement* procedure that checks consistency of symbolic traces. Finally, we present a verification method which is based on refinement and applies to regular frames.

Regular frames It is convenient to extend the syntax of messages with a new class of variables to be used as placeholders for generic messages known to the environment. Formally, we consider a new set $\widehat{\mathcal{V}}$ of *marked* variables, in bijection with $\mathcal{V}$ via a mapping $\hat{\cdot}$; thus, variables $x,y,z,\ldots$ have marked counterparts $\hat{x},\hat{y},\hat{z},\ldots$. Marked messages (resp., traces) are messages (resp., traces) that may also contain marked variables. Also, for $S\subseteq\mathcal{M}$, the set $\mathcal{H}(S)$ in Definition 2 is extended to include marked variables, i.e., we re-define $\mathcal{H}^0_{\mathcal{F}}(S)$ as follows:

$$\mathcal{H}^0_{\mathcal{F}}(S)\stackrel{\triangle}{=}S\cup\mathcal{EN}\cup\widehat{\mathcal{V}}.$$

The deduction relation ($S\ \vdash\ M$) remains formally unchanged. Note that in case S and M do not contain marked variables, this definition conservatively extends Definition 2. In practice, marked variables are treated as constants which are known to the environment. The satisfaction relation is extended to marked symbolic traces according to this intuition. For any $\hat{x}$ and any trace σ, we denote by $\sigma\backslash\hat{x}$ the longest prefix of σ not containing $\hat{x}$.

Definition 8. *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 7 when σ does not contain marked variables. We give now the definition of solved form, that lifts the concept of trace to the non-ground case (note that this definition is formally the same as Definition 3).

Definition 9 (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 $\sigma_1 \vdash M$.*

Regular frames enjoy a "finite-basis" property. Basically, this property states the existence of a finite set containing the building blocks of all messages that the attacker can synthesise out of a given σ. This requirement is stated by Condition 1, below. Condition 2 is a technical requirement about substitutions.

Definition 10 (regular frames). *A symbolic frame $\mathcal{F}^s$ is* regular *if there exists a function $\mathbf{b} : Act^* \longrightarrow \mathcal{P}_f(\mathcal{M})$ such that, for each solved form σ of $\mathcal{F}^s$ and for all ρ that satisfy σ:*

1. *$\sigma\rho \vdash M$ if and only if $M \in \mathcal{H}(\mathbf{b}(\sigma\rho))$;*
2. *$\mathbf{b}(\sigma\rho) \subseteq \mathbf{b}(\sigma)\rho$.*

For each σ, $\mathbf{b}(\sigma)$ is said a basis *of σ.*

Example: Public Key Encryption (3) Let us consider the frame $\mathcal{F}^s_{pk}$ defined in the previous sections. A basis function for this frame is defined by

$$\begin{aligned}\mathbf{b}_{pk}(\sigma) \stackrel{\triangle}{=} \{\, M \mid \sigma \vdash M \text{ and either: } M = u, \text{ or } M = u^+, \text{ or } M = u^-,\\ \text{for } u \in \mathcal{LN} \cup \mathcal{V}, \text{ or } M = \{\![N]\!\}_{u^+} \text{ for some } N \text{ and } u \text{ s.t. } \sigma \nvdash \langle N, u^+\rangle \,\}\end{aligned}$$

Condition 1 and Condition 2 of Definition 10 are proven by induction on M and σ, respectively. In practice, for a given σ, the set $\mathbf{b}_{pk}(\sigma)$ can be effectively computed by an iterative procedure, which repeatedly applies destructors ($\mathsf{dec}^{\mathsf{pk}}_{(\cdot)}(\cdot)$ and $\pi_i(\cdot)$) to messages in σ, until some fixed point is reached. This procedure always terminates. We omit the details.

Refinement In the refinement procedure, input messages in a symbolic trace are tentatively unified to messages that can be synthesized from the basis. By iterating this procedure, one can check whether a given symbolic trace can eventually be instantiated to a trace in the concrete model. In particular, given any symbolic trace σ, and a basis for it, we can compute the set of the most general instances of σ that are in solved form, denoted by $\mathbf{SF}(\sigma)$.

Definition 11 (refinement and $\mathbf{SF}(\sigma)$). *We let* refinement, *written $\succ$, be the least binary relation over marked symbolic traces of a regular frame given by the following rules. In* ($\textsc{Ref}_1$), *σ' is the longest prefix of σ that is in solved form and $\sigma = \sigma' \cdot \mathsf{a}\langle M\rangle \cdot \sigma''$, for some σ'. Assume $N, N' \notin \mathcal{V} \cup \widehat{\mathcal{V}}$.*

$$(\textsc{Ref}_1)\ \frac{M = C[N] \quad N' \in \mathbf{b}(\sigma') \quad \theta = \mathrm{mgu}(N, N')}{\sigma\ \succ\ \sigma\theta\theta_0}$$

where $\theta \neq \epsilon$, $\theta_0 \triangleq \{\, {}^{x}\!/\!_{\hat{x}} \mid \hat{x} \in \mathrm{v}(\sigma)$ *and* $|(\sigma\theta)\backslash\hat{x}| < |\sigma\backslash\hat{x}|\,\}$;

$$(\textsc{Ref}_2)\ \frac{x \in \mathrm{v}(M)}{\sigma\ \succ\ \sigma[{}^{\hat{x}}\!/\!_{x}]}$$

For any symbolic trace σ*, we let* $\mathbf{SF}(\sigma) \triangleq \{\, \sigma' \mid \sigma\ (\succ)^*\ \sigma'$ *and* σ' *is in sf* $\}$.

Rule $(\textsc{Ref}_1)$ implements the basic step of refinement: the subcomponent N of M gets instantiated, via θ, to an element of $\mathbf{b}(\sigma')$. E.g., consider $\sigma = \bar{\mathsf{c}}\langle\{[a]\}_{k^+}\rangle \cdot \bar{\mathsf{c}}\langle\{[b]\}_{k^+}\rangle \cdot \mathsf{c}\langle\{[x]\}_{k^+}\rangle$: its possible refinements are $\sigma\ \succ\ \sigma[{}^{a}\!/\!_{x}]$ and $\sigma\ \succ\ \sigma[{}^{b}\!/\!_{x}]$, and the refined traces are in sf. By rule $(\textsc{Ref}_2)$ a variable can be marked: this amounts to constraining its possible values to be messages known by the environment. (For technical reasons, marked variables sometimes need to be 'unmarked' back to plain variables, and this is achieved in $(\textsc{Ref}_1)$ via the renaming θ_0.)

The proposition below states that solutions of a symbolic trace σ can be completely characterized in terms of $\mathbf{SF}(\sigma)$.

Proposition 1. *Let* $\mathcal{F}$ *be a regular frame,* 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* $\sigma' \in \mathbf{SF}(\sigma)$*.*

Note that, since solved forms always have a solution (just map each variable to any name in $\mathcal{EN}$), the above proposition implies that σ is consistent if and only if $\mathbf{SF}(\sigma) \neq \emptyset$.

The verification method The method $\mathbf{M}(\mathcal{C}, \alpha \hookleftarrow \beta)$ described in Table 6 can be used to verify if $\mathcal{C} \models \alpha \hookleftarrow \beta$ or not. If the property is not satisfied, the method computes a trace violating the property, that is, an attack on $\mathcal{C}$. To understand

$\mathbf{M}(\mathcal{C}, \alpha \hookleftarrow \beta)$
1. compute $\mathbf{Mod}_{\mathcal{C}} = \{\sigma \mid \mathcal{C} \searrow_{\mathsf{s}} \sigma\}$;
2. **foreach** $\sigma \in \mathbf{Mod}_{\mathcal{C}}$ **do**
foreach action γ in σ **do**
if $\exists\, \theta = \mathrm{mgu}(\beta, \gamma)$ **and** $\exists\, \sigma' = (\sigma\theta\theta') \in \mathbf{SF}(\sigma\theta)$ s.t.
$\alpha\theta\theta'$ does not occur prior to $\beta\theta\theta'$ in σ'
then return(No, σ');
3. **return**(Yes);

Table 6. The verification method

how the method works, it is better to consider the simple case $\alpha = \bot$, i.e. verification of $\mathcal{C} \models \bot \hookleftarrow \beta$. This means verifying that, in the concrete semantics, no instance of action β is ever executed starting from $\mathcal{C}$. By the correspondence between symbolic and concrete semantics (Theorem 1), this amounts to checking that for each σ symbolically generated by $\mathcal{C}$, no solution of σ contains an instance of β. To verify this, the method proceeds as follows: for each such σ, and for each action γ in σ, it is checked whether there is a mgu θ for γ and β. If, for each σ, such a θ does not exist, or if it exists but $\sigma\theta$ is not consistent (i.e. $\mathbf{SF}(\sigma\theta) = \emptyset$, by the considerations following Proposition 1), then the property is true, otherwise it is not. The correctness of the method in the general case is stated in the following theorem. Its proof relies on Theorem 1 and on Proposition 1, plus routine calculations on unifiers.

Theorem 2 (correctness and completeness). *Let $\mathcal{F}^s$ be a regular frame, $\mathcal{C}$ be an initial configuration of $\mathcal{F}^s$ and α and β be actions such that* $\mathrm{v}(\alpha) \subseteq \mathrm{v}(\beta)$ *and* $\mathrm{v}(\beta) \cap \mathrm{v}(\mathcal{C}) = \emptyset$.

- *If* $\mathbf{M}(\mathcal{C}, \alpha \hookleftarrow \beta)$ *returns* Yes *then* $\mathcal{C} \models \alpha \hookleftarrow \beta$.
- *If* $\mathbf{M}(\mathcal{C}, \alpha \hookleftarrow \beta)$ *returns* $(\mathsf{No},\ \sigma)$ *then* $\mathcal{C} \not\models \alpha \hookleftarrow \beta$. *In particular, for any injective ground substitution* $\rho : \mathrm{v}(\sigma) \longrightarrow \mathcal{EN}$, *we have that* $\mathcal{C} \searrow (\sigma\rho)$ *and that* $(\sigma\rho) \not\models \alpha \hookleftarrow \beta$.

In practice, rather than generating the whole set of symbolic traces at once (step 1) and then checking the property, it is more convenient to work 'on-the-fly' and comparing every last symbolic action γ taken by the configuration against action β of the property $\alpha \hookleftarrow \beta$; the refinement procedure $\mathbf{SF}(\cdot)$ is invoked only when β and γ are unifiable.

5 An example: the Needham-Schroeder protocol

In this section, we analyze the Needham-Schroeder protocol within our framework. First, we give an informal description of the protocol (see, e.g., [17], for further details). Here, A is the initiator and B is the responder.

$$\begin{array}{ll}
1.\ A \longrightarrow B : \{NA, A\}_{kB^+} & (NA \text{ fresh nonce}) \\
2.\ B \longrightarrow A : \{NA, NB\}_{kA^+} & (NB \text{ fresh nonce}) \\
3.\ A \longrightarrow B : \{NB\}_{kB^+} &
\end{array}$$

As reported in [17], there is a well-known attack on this protocol, in case A may also run the protocol with a dishonest participant I. The following is a formalization of the protocol according to our method. For readability, we adopt some abbreviations; for instance, we write '$\mathsf{a}(\{\![M, N]\!\}_{k^+}).\, A$' for '$\mathsf{a}(x).\ \mathsf{let}\ y = \mathsf{dec}^{\mathsf{pk}}_{k^-}(x)\ \mathsf{in}\ (\mathsf{let}\ z = \pi_1(y)\ \mathsf{in}\ (\mathsf{let}\ w = \pi_2(y)\ \mathsf{in}\ ([z = M][w = N]A)))$.

$$\begin{aligned}
A \triangleq\ & \overline{\mathsf{a1}}\langle\{\![NA, \mathtt{id_A}]\!\}_{kB^+}\rangle.\, \mathsf{a2}(\{\![NA, xNB]\!\}_{kA^+}).\, \overline{\mathsf{a3}}\langle\{\![xNB]\!\}_{kB^+}\rangle.\, \mathbf{0} \\
& ||\quad \overline{\mathsf{a'1}}\langle\{\![N'A, \mathtt{id_A}]\!\}_{kI^+}\rangle.\, \mathsf{a'2}(\{\![N'A, xNI]\!\}_{kA^+}).\, \overline{\mathsf{a'3}}\langle\{\![xNI]\!\}_{kI^+}\rangle.\, \mathbf{0}
\end{aligned}$$

$$B \triangleq \mathsf{b1}(\{\![yNA, \mathtt{id_A}]\!\}_{kB^+}).\, \overline{\mathsf{b2}}\langle\{\![yNA, NB]\!\}_{kA^+}\rangle.\, \mathsf{b3}(\{\![NB]\!\}_{kB^+}).\, \mathbf{0}$$

$$NS \triangleq \langle \overline{\mathsf{disclose}}\langle kI, kA^+, kB^+, \mathtt{id_A}, \mathtt{id_B}, \mathtt{id_I}\rangle,\ (A \,||\, B)\rangle$$

$$AuthAtoB \quad \triangleq \quad \overline{\mathsf{a3}}\langle\{\![z]\!\}_{k^+}\rangle \hookleftarrow \mathsf{b3}\langle\{\![z]\!\}_{k^+}\rangle$$

Our verification method finds a symbolic trace that refines to the following concrete trace, which violates the property $AuthAtoB$:

$$\overline{\mathsf{disclose}}\langle kI, kA^+, kB^+, \mathtt{id_A}, \mathtt{id_B}, \mathtt{id_I}\rangle \cdot \overline{\mathsf{a'1}}\langle\{\![N'A, A]\!\}_{kI^+}\rangle \cdot \mathsf{b1}\langle\{\![N'A, A]\!\}_{kB^+}\rangle\cdot$$
$$\overline{\mathsf{b2}}\langle\{\![N'A, NB]\!\}_{kA^+}\rangle \cdot \mathsf{a'2}\langle\{\![N'A, NB]\!\}_{kA^+}\rangle \cdot \overline{\mathsf{a'3}}\langle\{\![NB]\!\}_{kI^+}\rangle\cdot\mathsf{b3}\langle\{\![NB]\!\}_{kB^+}\rangle.$$

When fed with the above example, our prototype STA detects this attack in a fraction of second.

6 Other cryptographic primitives

We consider extending the symbolic frame $\mathcal{F}^s_{pk}$, that served as a running example in the previous sections, to deal with some of the most common cryptographic operations.

The set Σ is enriched by means of appropriate operators for shared-key encryption $\{\cdot\}_{(\cdot)}$ and decryption $\mathsf{dec}^{\mathsf{sk}}_{(\cdot)}(\cdot)$, digital signing $[\![\cdot]\!]_{(\cdot)}$ and verifying $\mathsf{dec}^{\mathsf{ds}}_{(\cdot)}(\cdot)$ and hashing $H(\cdot)$. The syntax of messages is extended via the following additional clauses:

$$\begin{array}{llll} M, N ::= & \ldots & & \text{as in Table 2} \\ & \mid \ \{M\}_u & \mid \ [\![M]\!]_{u^-} & \Big| \ H(M)\,. \end{array}$$

The symbolic and concrete evaluations are given in terms of an auxiliary relation $\rightsquigarrow$, defined as expected. In particular, hashing has no rules, digital signature rules are just the same as for public key, but the roles of u^+ and u^- are swapped. For shared key, the concrete and symbolic rules are given below:

$$\mathsf{dec}^{\mathsf{sk}}_u(\{M\}_u) \rightsquigarrow M \qquad\qquad \mathsf{dec}^{\mathsf{sk}}_v(\{M\}_u) \overset{\theta}{\rightsquigarrow}_s M\theta \quad \text{where } \theta = \mathrm{mgu}(u, v)\,.$$

A finite basis for this frame can be given by including into the basis given for public key also all messages of the form $\{M\}_u$ (resp. $[\![M]\!]_{u^-}$, $H(M)$) s.t. $\sigma \nvdash u$ (resp. $\sigma \nvdash \langle M, u^-\rangle$, $\sigma \nvdash M$).

7 Conclusions

We have proposed a framework for the analysis of security protocols and provided some sufficient conditions under which verification can be performed via a symbolic method. In contrast to finite-state model checking, our method can analyze the whole infinite state space generated by a limited number of participants. The method is efficient in practice, because the symbolic model is compact, and the refinement procedure at its heart is only invoked on demand and on single symbolic traces. However, claims on efficiency should generally be taken with some care, given that the protocol analysis problem is NP-hard even under very mild hypotheses (see e.g. [25]).

Early work on symbolic analysis is due to Huima. In [16], the execution of a protocol generates a set of equational constraints. Only an informal description is provided of the kind of equational rewriting needed to solve these constraints. More recent approaches based on symbolic analysis are exploited in [5, 14]. The paper [5] extends the symbolic reachability analysis of [4] to hashing functions and public key cryptography and establishes some results on the complexity of the problem. Unlike our approach, symbolic execution and consistency check are not kept separate, and this may have a relevant impact on the size of the computed symbolic model. Another point worth noting is that, in [5], a brute-force method is needed to resolve variables in key position: such variables have to be instantiated to every possible name used by the participants; this fact may lead to state explosion, too. In [14], a procedure is provided to analyze the knowledge of the environment, based on a symbolic semantics akin to [7]. The approach applies to protocols with shared-key encryption and arbitrary messages as keys, but, like ours, it is proven complete only for atomic keys. Also, the method suffers from the same problem as [5] concerning brute-force instantiation.

Other recent developments of the symbolic approach are presented in [11] and [20]. Both of them do not rely on unification to build the symbolic model. The decision technique in [11] is based on a reduction to a set constraint problem which is in turn reduced to an automata-theoretic problem. Completeness is proven by assuming rather severe restrictions on protocol syntax. The technique in [20] focuses on reachability properties and is based on constraint solving. The symbolic reduction and the knowledge analysis are separated and the latter is performed by a procedure for solving a system of constraints.

In the future, we plan to focus on the verification of protocols that also exploit low-level cryptographic operations, such as modular exponentiation. As an example, we are confident that Diffie-Hellman key exchange can be smoothly set within our framework.

Acknowledgements. We thank the anonymous referees for helpful comments.

References

1. M. Abadi, B. Blanchet. Analyzing Security Protocols with Secrecy Types and Logic Programs. In *Conf. Rec. of POPL'02*, 2002.
2. M. Abadi, C. Fournet. Mobile Values, New Names, and Secure Communication. In *Conf. Rec. of POPL'01*, 2001.
3. M. Abadi, A.D. Gordon. A calculus for cryptographic protocols: The spi calculus. *Information and Computation*, 148(1):1-70, 1999.
4. R.M. Amadio, S. Lugiez. On the reachability problem in cryptographic protocols. In *Proc. of Concur'00*, LNCS 1877, 2000. Full version: RR 3915, INRIA Sophia Antipolis.
5. R.M. Amadio, S. Lugiez, V. Vanackère. On the symbolic reduction of processes with cryptographic functions. RR 4147, INRIA Sophia Antipolis, March 2001.
6. R. Amadio, S. Prasad. The game of the name in cryptographic tables. In *Proc. of Asian'00*, LNCS 1742, Springer-Verlag, 2000. RR 3733 INRIA Sophia Antipolis.

7. M. Boreale. Symbolic Trace Analysis of Cryptographic Protocols. In *Proc. of ICALP'01*, LNCS 2076, Springer-Verlag, 2001.
8. M. Boreale, M. Buscemi. Experimenting with STA, a Tool for Automatic Analysis of Security Protocols. In *Proc. of SAC'02*, ACM Press, 2002.
9. 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*.
10. E.M. Clarke, S. Jha, W. Marrero. Using State Space Exploration and a Natural Deduction Style Message Derivation Engine to Verify Security Protocols. In *Proc. of IFIP PROCOMET*, 1998.
11. H. Comon, V. Cortier, J. Mitchell. Tree automata with one memory, set constraints and ping-pong protocols. In *Proc. of ICALP'01*, LNCS 2076, Springer-Verlag, 2001.
12. D. Dolev, A. Yao. On the security of public-key protocols. *IEEE Transactions on Information Theory*, 2(29):198-208, 1983.
13. N. Durgin, P. Lincoln, J. Mitchell, A. Scedrov. Undecidability of bounded security protocols. In *Proc. of Workshop on Formal Methods and Security Protocols*, 1999.
14. M.P. Fiore and M. Abadi. Computing Symbolic Models for Verifying Cryptographic Protocols. In *Proc. of 14th Computer Security Foundations Workshop*, IEEE Computer Society Press, 2001.
15. J. Heather, G. Lowe, and S. Schneider. How to prevent type flaw attacks on security protocols. In *Proc. of 13th Computer Security Foundations Workshop*, IEEE Computer Society Press, 2000.
16. A. Huima. Efficient infinite-state analysis of security protocols. In *Proc. of Workshop on Formal Methods and Security Protocols*, Trento, 1999.
17. G. Lowe. Breaking and Fixing the Needham-Schroeder Public-Key Protocol Using FDR. In *Proc. of TACAS'96*, LNCS 1055, Springer-Verlag, 1996.
18. G. Lowe. A Hierarchy of Authentication Specifications. In *Proc. of 10th IEEE Computer Security Foundations Workshop*, IEEE Computer Society Press, 1997.
19. W. Marrero, E.M. Clarke, S. Jha. Model checking for security protocols. Technical Report TR-CMU-CS-97-139, Carnegie Mellon University, 1997.
20. J. Millen, V. Shmatikov. Constraint solving for bounded-process cryptographic protocol analysis. In *Proc. of 8th ACM Conference on Computer and Communication Security*, ACM Press, 2001.
21. J.C. Mitchell, M. Mitchell, U Stern. Automated Analysis of Cryptographic Protocols Using Murφ. In *Proc. of Symp. Security and Privacy*, IEEE Computer Society Press, 1997.
22. L.C. Paulson. The inductive approach to verifying cryptographic protocols. *Journal of Computer Security*, 6:85–128, 1998.
23. A.W. Roscoe. Modelling and verifying key-exchange using CSP and FDR. In *Proc. of 8th Computer Security Foundations Workshop*, IEEE Computer Society Press, 1995.
24. A.W. Roscoe. Proving security protocols with model checkers by data independent techniques. In *Proc. of 11th Computer Security Foundations Workshop*, IEEE Computer Society Press, 1998.
25. M. Rusinowitch, M Turuani. Protocol Insecurity with Finite Number of Sessions in NP-Complete. In *Proc. of 14th Computer Security Foundations Workshop*, IEEE Computer Society Press, 2001.
26. S. Schneider. Verifying Authentication Protocols in CSP. *IEEE Transactions on Software Engineering*, 24(8):743-758, 1998.
27. 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`.