A BRUTUS Logic for a Spi-Calculus Dialect

S. Gnesi[1], D. Latella[2], and G. Lenzini[1]

[1] IEI-CNR, Pisa (Italy), ({gnesi,lenzini}@iei.pi.cnr.it)
[2] CNUCE-CNR, Pisa (Italy), (d.latella@cnuce.pi.cnr.it)

Abstract. In the field of process algebras, the spi-calculus, a modified version of the π-calculus with encryption primitives, is indicated as an expressive specification language for cryptographic protocols. In spi-calculus basic security properties, such as secrecy and integrity can be formalized as may-testing equivalences which do not seem easily extendible to express other kinds of interesting properties such as, for example, anonymity. When, as a language for properties specification, temporal logics are used a more expressive power can be reached making possible to represent a wider class of properties. Recently, within the BRUTUS model checker, a first order temporal logic has been defined, making possible to express both basic and advanced properties, such as different kinds of authenticity and anonymity. In this work we define a spi-calculus dialect on which the BRUTUS logic can be interpreted with a double, in our opinion, potential advantage: to provide the spi-calculus like languages with a temporal logics as a flexible medium of security properties expression, and to enlarge the BRUTUS model checker with a widely used specification language for cryptographic protocols.

Key words: security protocols, security properties, spi-calculus, temporal logics, model checking.

ACM Computing Classification: D.2.4 Software/Program Verification - F.3.1 Specifying and Verifying and Reasoning about Programs

1 Introduction and Related Works

The wide diffusion of Internet as a commercial medium makes the guarantee of security a necessity for every distributed protocol running over it. In fact, experiences have shown that many cryptographic protocols considered secure for years, have been successively proved to be easily breakable. Many researchers have dedicated their efforts to propose new formal frameworks in which security properties of cryptographic protocols can be studied and analyzed (see for example [20, 26, 23, 27, 28, 11, 16]). Within the model checking approaches [7], we remind the studies, oriented to define an effective theoretical framework in [18, 15, 14], and the recent BRUTUS model checker [19].

This paper focuses on spi-calculus [4], a process algebra derived from the π-calculus [21, 22] with operators to encrypt and decrypt messages. The spi-calculus is expressive and flexible enough to easily allow the description of a wide class of cryptographic protocols. Basic properties, such as secrecy and integrity (a weak form of authenticity), can be expressed in it [2, 3] as may-testing [24] or barbed equivalences. Despite rigorous and intuitive, the definition of these properties suffers of quantification over infinite contexts which make an automatic checking impossible. To face this weakness Boreale, De Nicola and Pugliese in [6] have proposed environment-sensitive labeled transition systems on which equivalence and a weak bisimulation relations, not requiring quantification over any context, have been defined and proved to be sufficient for may-testing equivalence. By the way the authors limit the used of trace equivalence and weak bisimulation to express secrecy and integrity. Following a different approach, Abadi in [1] defines a set of typing rules for achieving secrecy properties. Based on traditional concepts of classification and control flows, these rules assure that if a protocol type-checks than it does not leak secrecy.

Recently Clarke, Jha and Marrero in the model checking framework, has developed an automatic model checker, BRUTUS [9] and defined a first order linear temporal logic to express security properties [8]. One of the main advantages of BRUTUS logic is its expressiveness able to to express not only secrecy and authenticity properties but also other, more complex, properties such as, for example, anonymity [10]. Within BRUTUS a protocol can be described using an embedded language whose semantics is based on labeled transition systems.

In this work we propose a spi-calculus dialect and its BRUTUS logic. The spi-calculus dialect is provided with an operational semantics based on labeled transition systems, on which the BRUTUS logic satisfiability relation has been defined. Indeed, our main goal is to provide an automatic framework for verifying, on spi-calculus protocols, logic properties in a model checking approach. In this sense this work can be considered as a first step in direction. To our knowledge, no such approach is available so far for the spi-calculus .

In Section 2 we outline some simplifying hypothesis assumed in this work. In Section 3 we illustrate the Denning-Sacco Protocol, which will be used as a running example in the paper. In Section 4, 5 and 6, we define the spi-calculus dialect. In Section 7 we report the BRUTUS temporal logics syntax, while in

Section 8 we define its satisfiability relation for our spi-calculus dialect. Finally in Section 9 we outline how to make model checking feasible and conclude.

2 Assumptions

In this section we make explicit the assumptions under which we have developed our work.

Instead of describing the attacker, the malicious entity supposed to play against honest agents in order to break security, as an additional principal explicitly interacting with the other agents, we assume (see for example [27, 19, 6, 13, 5]), the presence of an *environment* that controls all communication media. It implies that: (a) all the channels are public and considered both in input and in output to the environment; (b) all the communication events require a synchronization with the environment, that is whenever an agent performs an output action the environment performs an input, and whenever an agent performs an input action the environment performs an output; (c) the environment can modify or replace messages, or it can generate new messages starting from the ones it already had. The remaining assumptions regard on the cryptosystem used to encrypt or decrypt messages.

In order to maintain separate correctness and security issues related to the protocol itself, from those of the cryptosystem, we suppose to work under the *perfect encryption* assumption [19].

Assumption 1 (Perfect Encryption) *Given a ciphertext $\{T\}_K$, it can be decrypted only using the decryption key K^{-1}. Given two ciphertexts, $\{T_1\}_{K_1}$ and $\{T_2\}_{K_2}$ it is assumed that $\{T_1\}_{K_1} = \{T_2\}_{K_2}$ if and only if $T_1 = T_2$ and $K_1 = K_2$.*

Moreover we suppose that only *atomic* keys, instead of whole messages, could be used as encryption keys. In fact, the use of more complicated keys would increase the complexity of the message structure without helping in the issues we want to address in this present paper. Finally we suppose, without loss of generality, that the cryptosystem used is *symmetric*, that is the relative decryption key k^{-1} of an encryption key k, is the key k itself.

3 A Running Example: the Denning-Sacco Protocol

In this section we describe the Denning-Sacco protocol as described in [17]. It will be referred later in the paper. The Denning-Sacco is a conventional-key authentication protocol which involves two agents, A and B, and an authentication server, S. Its aim is to establish a session key K_{ab} that will be used by the agents as secure encryption key in successively communications. Each of the two agents, A and B, shares with the authentication server S a key, respectively indicated with K_{as} and K_{bs}, that will be used for encrypted communications with the server. The Denning-Sacco protocol can be informally described with

the following sequence of actions:

$$\begin{aligned}
&\text{step 1. } A \to S : A, B \\
&\text{step 2. } S \to A : \{B, K_{ab}, T_s, \{K_{ab}, A, T_s\}_{K_{bs}}\}_{K_{as}} \\
&\text{step 3. } A \to B : \{K_{ab}, A, T_s\}_{K_{bs}}
\end{aligned}$$

In this informal description $A \to B : m$ indicates that A sends a message m to B, which consequently receives it. In step 1, A requires to the server S a session key to be used in communications with B. In step 2, S generates a new name T_s, and sends back to A a message containing the session key K_{ab} and an encrypted message addressed to B. In step 3 A forwards to B the part of the message reserved for B.

The protocol is known to be prone to the following attack, occurring when a second session of the protocol is running. We report the sequence of action which represents the attack: in particular in step 3' of the second run, an intruder who has stored a previous message from A, acting as A makes B erroneously believe to start a new communication.

$$\begin{aligned}
&\text{step 1.} \quad A \quad \to S : A, B \\
&\text{step 2.} \quad S \quad \to A : \{B, K_{ab}, T_s, \{K_{ab}, A, T_s\}_{K_{bs}}\}_{K_{as}} \\
&\text{step 3.} \quad A \quad \to B : \{K_{ab}, A, T_s\}_{K_{bs}} \\
&\text{step 3}'. \; I(A) \to B : \{K_{ab}, A, T_s\}_{K_{bs}}
\end{aligned}$$

4 Language Syntax

In this section we describe the syntax our spi-calculus dialect. It is a modified version of the spi-calculus of Abadi and Gordon [4], without mobility and where the "let" and "case" constructs have been embedded into, respectively, the output and input primitives. Mobility is a characteristic that in cryptographic protocols seems to have a minor interest than in other protocols, at least at the abstract level we intend to follow (indeed most of the works on protocol languages disregard it, see for example [27, 13, 6]), while the different implementation of "let" and "case" is required to apply strategies to make the semantics model finite states in model checking.

In the language are supposed (1) a set $\mathcal{A}$ of *labels*, used to distinguish among communication events; (2) a set $\mathcal{K}$ of *atomic keys*, used as encryption keys; (3) a set $\mathcal{N}$ of *atomic names* which can be agent names, atomic messages, nounces etc.; (4) a set $\mathcal{V}$ of *variables*; (5) a finite set $\mathcal{I}$ of *agent identifiers*. Without loss of generality we can assume that the set of identifiers is the finite set $I_n = \{1, \ldots n\}$ of positive integers. We let a range over labels, k over atomic keys, $\mathtt{n}$ over atomic names, m over atomic names and keys when no distinction is required, x, y over variables, and i over agent identifiers.

Atomic names and keys are used to build the set $\mathcal{M}$ of *messages*, via encryption and pairing. As encryption keys we assume to use only atomic keys. The set

$\mathcal{T}$ of *message terms* is built, still via encryption and paring mechanisms, from atomic names or keys, and variables.

A *protocol* Z is a parallel composition of agent instances (i, P). Each agent instance is a couple composed by the unique agent identifier $i \in I_n$, and the agent P. Public names or keys m can be defined using (**public** m). A public name is known by the environment and everywhere the same public definition appears.

The *agents* syntax can be informally described as in the following: **0** is the agent that does nothing; the generation of a new atomic name or key m used in P, is indicated with (**new** m)P. In addition to new names, a public or shared name or key m used within P, can be defined respectively with (**public** m)P or (**shared** m)P. A public name has to be defined also at protocol level. A shared name is the same within all the agents which share the same definition. An output action, $\overline{a}(T).P$, allows a message to be sent on the channel labeled a. An input action, $a(T).P$, allows a message $M \in \mathcal{M}$ to be received via the channel a. The message M needs to be structurally unified with the message term T so that the action to be successful. The parallel execution of P and Q is written $P \parallel Q$, while $P + Q$ is the non-deterministic choice between P or Q. An equality test on message terms $[S{=}T]$, can be put as guard on actions.

In Figure 1 we have summarized the syntax, while in Figure 2 we have shown how to specify two runs of the Denning-Sacco protocol, introduced in Section 3. In the specification, for sake of clarity, we have omitted redundant parenthesis in nested pairs. Uppercase has been used for all the atomic messages involved in communications. Finally, the agent names A and B, and the server name S have been declared public in the protocol to point out that they are known by the environment.

5 Knowledge Functions

In this section we explain how to formalize the capabilities the environment has in manipulating messages. The environment can *synthesize* new messages by encryption or by pairing, it can *analyze* compound messages by decryption using the relative decryption key, or by splitting tuples. Synthesis and analysis, first introduced by Paulson in [25], can be formalized using the inference system described in Figure 3, which is composed by two different set of rules: expanding and shrinking, depending on the fact that the messages inferred are longer or not than the relative messages in the premise. The rule $\vdash_{dec}$ is used to obtain a message m from a cipher message $\{m\}_k$, supposing that the relative encryption key k is known. Other shrinking rules are $\vdash_{fst}$ and $\vdash_{snd}$ used to split a pair $\langle m, n\rangle$ respectively in its first and second component. The rule $\vdash_{cry}$ is used to generate an encrypted messages $\{m\}_k$ from the message m and an encryption key k, while the rule $\vdash_{pair}$ is used to compose pair of messages.

Given an initial set of messages W it is said that a message m is *generated* from W, written $W \vdash m$, if there exists a proof of m whose premises are contained in W, while it is said that a message m is *finitely generated* from W, written

$M, N ::= m \mid \{M\}_k \mid \langle M, N\rangle$	*messages* $\mathcal{M}$
$S, T ::= x \mid \{S\}_x \mid \langle S, T\rangle \mid M$	*message terms* $\mathcal{T}$
$Z ::=$	*protocol* $\mathcal{Z}$
$\quad (\textbf{public}\ m)Z$	public name
$\quad A \Vert Z$	agent list
$\quad A$	agent instance
$A ::=$	*agent instances* $\mathcal{P}$
$\quad$ (i,P)	
$P, Q ::=$	*agents*
$\quad \mathbf{0}$	nil
$\quad \mid a(T).P$	input
$\quad \mid \overline{a}(T).P$	output
$\quad \mid P \parallel Q$	parallel composition
$\quad \mid P + Q$	non-determinist choice
$\quad \mid (\textbf{new}\ m)P$	private name
$\quad \mid (\textbf{public}\ m)P$	public name
$\quad \mid (\textbf{shared}\ m)P$	shared name
$\quad \mid [S{=}T]P$	match
$\quad \mid p$	agent variable
$D ::= p \stackrel{def}{=} P$	*agent definitions*

The definition of free/bound names and variables are defined as expected.

Fig. 1. Syntax of the spi-calculus dialect.

$W \vdash_0 m$, if there exists a proof of m using only shrinking rules. Using the above definitions, it is possibible to formalize the notion of knowledge of the environment, starting from an initial set W of messages.

Definition 1 ([25]). *Let $W \subseteq \mathcal{M}$ be a finite set of messages. The* analysis *of W is the set $K(W) = W \cup \{m : W \vdash_0 m\}$.*

Definition 2 ([25]). *Let $W \subseteq \mathcal{M}$ be a finite set of messages. The* synthesis *of W, is the set $\overline{K}(W) = W \cup \{m : W \vdash m\}$.*

We want to underline that, given a finite set W, $K(W)$ is finite while $\overline{K}(W)$, is usually infinite. Anyway, testing if $m \in \overline{K}(W)$ is known to be decidable (see [9, 12]). In particular in [9] it is shown how the test can be answered in time $O(|m|)(|K(W)|)$.

$$
\begin{aligned}
DS \stackrel{def}{=}\ & (\mathbf{public}\ A)(\mathbf{public}\ B)(\mathbf{public}\ S) \\
& (1, p_s) \| (2, p_a) \| (3, p_b) \\
& \| (4, p_s) \| (5, p_a) \| (6, p_b) \\
p_a \stackrel{def}{=}\ & (\mathbf{public}\ A)(\mathbf{public}\ B)(\mathbf{public}\ S)(\mathbf{shared}\ K_{as}) \\
& \overline{c_{as}}(\{\langle A,\ B\rangle\}_{K_{as}}).c_{as}(\{\langle B, x_k, x_t, y\rangle\}_{K_{as}}).\overline{c_{ab}}(y).\mathbf{0} \\
p_b \stackrel{def}{=}\ & (\mathbf{public}\ A)(\mathbf{public}\ B)(\mathbf{public}\ S)(\mathbf{shared}\ K_{bs}) \\
& c_{ab}(\{\langle x_k, x_a, x_t\rangle\}_{K_{bs}}).\mathbf{0} \\
p_s \stackrel{def}{=}\ & (\mathbf{public}\ A)(\mathbf{public}\ B)(\mathbf{public}\ S)(\mathbf{shared}\ K_{as}) \\
& (\mathbf{shared}\ K_{sb})(\mathbf{new}\ T_s)(\mathbf{new}\ K_{ab})c_{as}(\{\langle A,\ B\rangle\}_{K_{as}}). \\
& \overline{c_{as}}(\{\langle B, K_{ab}, T_s, \{\langle K_{ab}, A, T_s\rangle\}_{K_{bs}}\rangle\}_{K_{as}}).\mathbf{0}
\end{aligned}
$$

Fig. 2. Specification of two runs of Denning-Sacco protocol.

Expanding rules

$$\frac{m \quad k}{\{m\}_k}\ (\vdash_{cry}) \qquad \frac{m \quad n}{\langle m, n\rangle}\ (\vdash_{pair})$$

Shrinking rules

$$\frac{\{m\}_k \quad k}{M}\ (\vdash_{dec}) \qquad \frac{\langle m, n\rangle}{m}\ (\vdash_{fst}) \qquad \frac{\langle m, n\rangle}{n}\ (\vdash_{snd})$$

Fig. 3. Inference system for message manipulation.

6 Language Semantics

In this section we define the semantics of our language. Since it is our intention to define a satisfiability relation for the BRUTUS logic [10], we will develop a semantics based on labeled transition systems (LTS) that are the semantic model for such a logic (see [8] for details). Let us start with some definitions.

Definition 3 (Local State). *A* local state *is a tuple* $(\mathtt{n}, w, \sigma)$*, where:*

- $\mathtt{n} \in \mathcal{N}$ *is a name;*
- $w \subseteq \mathcal{M}$ *is a set of messages;*
- $\sigma : \mathcal{V} \longrightarrow \mathcal{M}$ *is a function which binds variables to messages.*

With *Loc* we indicate the set of all possible local states. Moreover, given a local state $(\mathtt{n}, w, \sigma) \in Loc$, we indicate with $(\mathtt{n}, w, \sigma)\downarrow_1$, $(\mathtt{n}, w, \sigma)\downarrow_2$, $(\mathtt{n}, w, \sigma)\downarrow_3$ respectively the first component $\mathtt{n}$, the second component w, and the third component

σ. Finally, talking about binding functions, with the symbol $-$ we indicate the function undefined everywhere, while we write $\sigma' \sqsupseteq \sigma$ whenever the function σ' coincides with σ in every values of the domain where σ is defined.

Definition 4 (Extended Bindings). *Given a binding σ, we define the* extended binding $\widehat{\sigma}$*, as the extension of σ to the set of message terms, that is the function $\widehat{\sigma} : \mathcal{T} \longrightarrow \mathcal{M}$ so defined:*

$$\begin{array}{lll} \widehat{\sigma}(M) & = M, & \text{where } M \in \mathcal{M} \\ \widehat{\sigma}(x) & = \sigma(x), & \text{where } x \in \mathcal{V} \\ \widehat{\sigma}(\{T\}_x) & = \{\widehat{\sigma}(T)\}_{\sigma(x)} & \\ \widehat{\sigma}(\langle S, T\rangle) & = \langle \widehat{\sigma}(S), \widehat{\sigma}(T)\rangle & \end{array}$$

Definition 5 (Global State). *Given protocol $Z = (1, p_1) \| \ldots \| (n, p_n)$, consisting of n agent instances each identified by a different positive integer $i \in I_n$, a* global state *is a function $\mathcal{G} : I_n \cup \{0\} \to Loc$, such that:*

- *$\mathcal{G}(0) = (\Omega, w_\Omega, -)$ is the local state of the environment. It is composed by the environment name Ω, by the set of messages w_Ω passed through the network and by an empty bindings.*
- *$\forall i \in I_n$, $\mathcal{G}(i) = (\mathtt{n}, w, \sigma)$ is the local state of the agent instance whose identifier is i. It is composed by the agent's name $\mathtt{n}$, by the set of messages w explicitly received or created by the instance and by the function σ whose domain is the set of variable appearing in the agent instance specification.*

With *Glob* we indicate the set of all possible global states. The global state function describes the local state of the environment and the local state of each of the agent instance running the protocol.

Definition 6 (Sort). *Given a set $\mathcal{A}$ of labels. We define* sort *of the set $\mathcal{A}$, the set $\mathcal{L} = \mathcal{A} \cup \overline{\mathcal{A}}$ where $\overline{\mathcal{A}} = \{\overline{a} | a \in \mathcal{A}\}$. We let λ range over sorts.*

Definition 7 (Actions). *An* action *is either the term $i.\lambda(M)$ or the term τ where $i \in I_n$ is an agent identifier, $\lambda \in \mathcal{L}$ is a sort and $M \in \mathcal{M}$ is a ground message.*

In particular actions can be: (a) output actions $i.\overline{a}\langle M\rangle$, (b) input actions $i.a\langle M\rangle$, or the silent action τ. We indicate with $Act \cup \{\tau\}$ the set of actions, and we let α range over it.

Definition 8 (Name Function). *Given a protocol $Z = (1, p_1) \| \ldots \| (n, p_n)$ a* name function $\mathbf{name} : I_n \to \mathcal{M}$ *is a function from the set of agent identifier to messages, such that $\mathbf{name}(i) = \mathbf{name}(j)$ whenever i and j are different instances of the same agent, that is whenever $p_i = p_j$.*

The operational semantics is based on the LTS formally explained in Definition 9. An atomic transition of the LTS has form $\langle \mathcal{G}, Z\rangle \stackrel{\alpha}{\longmapsto} \langle \mathcal{G}', Z'\rangle$ and describes the atomic evolution of a protocol Z and a global environment $\mathcal{G}$.

Definition 9. *A* LTS *is a tuple $(Q_{\mathcal{Z}}, \mathcal{G}_0, Act \cup \{\tau\}, \mathcal{R}_{\mathcal{Z}})$, where:*

(P-PUB) $\langle \mathcal{G},\ (\mathbf{public}\ m)Z\rangle \stackrel{\tau}{\longmapsto} \langle \mathcal{G}',\ Z\rangle$ where $\mathcal{G}'(0)\downarrow_1 = \mathcal{G}(0)\downarrow_1 \cup \{m\}$

(P-PAR1) $\dfrac{\langle \mathcal{G},\ Z_1\rangle \stackrel{\alpha}{\longmapsto} \langle \mathcal{G}',\ Z_1'\rangle}{\langle \mathcal{G},\ Z_1 \| Z_2\rangle \stackrel{\alpha}{\longmapsto} \langle \mathcal{G}',\ Z_1' \| Z_2\rangle}$ (P-PAR2) $\dfrac{\langle \mathcal{G},\ Z_2\rangle \stackrel{\alpha}{\longmapsto} \langle \mathcal{G}',\ Z_2'\rangle}{\langle \mathcal{G},\ Z_1 \| Z_2\rangle \stackrel{\alpha}{\longmapsto} \langle \mathcal{G}',\ Z_1 \| Z_2'\rangle}$

(P-AGE) $\dfrac{\langle \mathcal{G},\ (i,P)\rangle \stackrel{\alpha}{\rightsquigarrow} \langle \mathcal{G}',\ (i,P')\rangle}{\langle \mathcal{G},\ (i,P)\rangle \stackrel{\alpha}{\longmapsto} \langle \mathcal{G}',\ (i,P')\rangle}$

Fig. 4. Transition rules for protocols

- $Q_{\mathcal{Z}} \subseteq Glob \times \mathcal{Z}$ *is the set of states;*
- $\mathcal{G}_0$ *is the initial state so defined:* $\mathcal{G}_0(0) = (\Omega, \emptyset, -)$, $\mathcal{G}_0(i) = (\mathbf{name}(i), \emptyset, -)$, *for all* $i \in I_n$.
- $Act \cup \{\tau\}$ *is the set of actions.*
- $\mathcal{R}_{\mathcal{Z}} \subseteq Q_{\mathcal{Z}} \times (Act \cup \{\tau\}) \times Q_{\mathcal{Z}}$ *is the transition relation defined in Figure 4. Whenever* $(Q, \alpha, Q') \in \mathcal{R}_{\mathcal{Z}}$ *we write* $Q \stackrel{\alpha}{\longmapsto} Q'$.

Referring to Figure 4, the (P-PUB) rule explains how the set of messages w_Ω of the environment is changed when a public name or key is defined using (**public** m). The (P-PAR1) and (P-PAR2) define the behavior of a parallel composition of two protocols when a single protocols evolves. Finally the (P-AGE) rules makes a single agent transition to rise up at protocol transition level. The definition of relation $\longmapsto$ is based on another deduction system defined by the relation $\rightsquigarrow$ whose transition rules are reported and described in Appendix A. The model is developed in such a way each possible execution corresponds to a finite *trace* $\pi = \mathcal{G}_0 \cdot \alpha_1 \cdot \mathcal{G}_1 \cdot \ldots \cdot \alpha_n \cdot \mathcal{G}_n$, of states $\mathcal{G}_i$ and actions α_i, where $\mathcal{G}_i \stackrel{\alpha_{i+1}}{\longmapsto} \mathcal{G}_{i+1}$. Anyway the resulted transition system could be potentially infinite. The source of infinity is rule the input rule of $\rightsquigarrow$ which involves the synthesis of the environment.

In next sections we show how the temporal logic used in BRUTUS can be interpreted over the transition system presented as semantics domain.

7 The BRUTUS Logics Syntax

We start with giving the syntax of BRUTUS logics [10]. As alphabet, we assume to have: (a) a set $\mathcal{M}$ of *messages*, (b) a set $\mathcal{L}$ of sort constructed from a set $\mathcal{A}$ of labels; (c) a finite set $\mathcal{I}$ of agents' identifier. All the previous sets coincide with the ones defined in Section 4. In addition we have: (d) a set $\mathcal{V}_m$ of message variables; (e) a set $\mathcal{V}_s$ of agent instance variables; (f) a set of predicate symbols; (g) the symbols $\neg, \wedge, \exists, \Diamond_P$.

Alphabet symbols are used to built *terms*, *atomic propositions* and *formulae* that constitute the syntax of the BRUTUS logics, as summarized in Figure 5.

$f ::=$	$p \mid \neg f \mid f_1 \wedge f_2 \mid \exists s.f \mid \Diamond_P f$	formulae
$p ::=$	$\mathbf{Knows}(s,t) \mid \mathbf{Acts}_\lambda(s,m) \mid t_1 = t_2$	atomic propositions
$s ::=$	$i \mid \mathbf{name}(s) \mid v_s$	agent-instance terms
$t ::=$	$M \mid \langle t_1, t_2 \rangle \mid \{t_1\}_{t_2} \mid v_t \mid s.t$	message terms

Fig. 5. The syntax of the BRUTUS logic

Terms are splitted in two different sets: *agent-instance* terms $\mathcal{T}_s$ and *message* term $\mathcal{T}_m$ defined as in the following. Informally, an agent-instance term can either be an agent identifier, or an agent variable. A message term can be either an atomic message, or a message variable, or a pair of message term, or an encryption of message terms. The message term $\mathbf{name}(s)$ is the message indicating the name of the agent identifier s. The message term $s.t$ is the message term t interpreted in the local state of the agent instance s.

Atomic propositions can be: $\mathbf{Knows}(s,t)$, which is a predicate on the knowledge of the agent instance s of the message t, $\mathbf{Acts}_\lambda(s,t)$ which is a predicate on the action λ, performed by s, involving the message t, and $t_1 = t_2$, is an equality test over message terms, $t_1 = t_2$.

A formula, can be any propositional logic formula, or the modal formula $\Diamond_P f$, where the symbol $\Diamond_P$ is the modal operator *eventually* in its past interpretation. The formula $\exists s.f$ binds the agent-instance term s within the formula f.

8 Logics Semantics

It is our intention to define an interpretation of formulae over finite *traces* $\pi = \mathcal{G}_0 \cdot \alpha_1 \cdot \mathcal{G}_1 \cdot \ldots \cdot \alpha_n \cdot \mathcal{G}_n$, where $\mathcal{G}_i \overset{\alpha_{i+1}}{\longmapsto} \mathcal{G}_{i+1}$ is a possible protocol transition. We start defining the interpretation of terms.

Definition 10 (Agent-instance Term Interpretation). *An* agent-instance term interpretation, $\mathbb{A}$, *is a function from agent-instance terms* $\mathcal{T}_s$ *to* $\mathcal{I}$, *that is* $\mathbb{A} : \mathcal{T}_s \to \mathcal{I}$, *such that* $\mathbb{A}(i) = i$

Message terms are interpreted on global states *Glob*.

Definition 11 (Message Term Interpretation). *Given a global state* $\mathcal{G} \in$ *Glob and an message term* t, *a* message term interpretation, *is the function*

$\mathbb{M} : \mathit{Glob} \to \mathcal{T}_m \to \mathcal{M}$ *given below:*

$$\begin{array}{lll}
\mathbb{M}(\mathcal{G})(m) & = m, & \textit{where: } m \in \mathcal{M} \\
\mathbb{M}(\mathcal{G})(\langle t_1,\, t_2\rangle) & = \langle \mathbb{M}(\mathcal{G})(t_1),\, \mathbb{M}(\mathcal{G})(t_2)\rangle & \\
\mathbb{M}(\mathcal{G})(\{t_1\}_{t_2}) & = \{\mathbb{M}(\mathcal{G})(t_1)\}_{\mathbb{M}(\mathcal{G})(t_2)} & \\
\mathbb{M}(\mathcal{G})(j.t) & = \widehat{\sigma}(t), & \textit{where: } \sigma = \mathcal{G}(j)\downarrow_3 \\
\mathbb{M}(\mathcal{G})(\mathbf{name}(j)) & = n, & \textit{where: } n = \mathcal{G}(j)\downarrow_1
\end{array}$$

Given a global state $\mathcal{G} \in \mathit{Glob}$, the interpretation of an atomic message m is the message m itself; the interpretation of a pair $\langle t_1,\, t_2\rangle$ or an encryption $\{t_1\}_{t_2}$ is defined recursively on the interpretation of message terms t_1 and t_2; the interpretation of $i.t$ is the message obtained instantiating all the variables appearing in t, using the set of bindings of the agent instance whose identifier is i; the interpretation of $\mathbf{name}(i)$ is the message which represents the name of the agent instance whose identifier is i.

Formulae of the logic are interpreted over finite traces of a protocol model. We first define interpretation of atomic formulae on a particular state. We will write $\langle \pi,\, i\rangle \models_A p$ to mean that the atomic proposition p is satisfied on the state $\mathcal{G}_i$ of the trace π. Then we define interpretation of formulae over a state. We will write $\langle \pi,\, i\rangle \models f$ to mean that the formula f is satisfied on the state $\mathcal{G}_i$.

Definition 12 (Atomic Proposition Interpretation). *Given a trace* $\pi = \mathcal{G}_0 \cdot \alpha_1 \cdot \mathcal{G}_1 \cdot \ldots \cdot \alpha_n \cdot \mathcal{G}_n$*, we have that:*

$$\begin{array}{lll}
\langle \pi,\, i\rangle \models_A t_1 = t_2 & \textit{iff} & \mathbb{M}(\mathcal{G}_i)(t_1) = \mathbb{M}(\mathcal{G}_i)(t_2) \\
\langle \pi,\, i\rangle \models_A \mathbf{Knows}(j,t) & \textit{iff} & \mathbb{M}(\mathcal{G}_i)(t) \in \overline{K}(\mathcal{G}_i(j)\downarrow_1) \\
\langle \pi,\, i\rangle \models_A \mathbf{Acts}_\lambda(j,t) & \textit{iff} & \alpha_{i+1} = j.\lambda(m),\ \textit{where: } m = \mathbb{M}(\mathcal{G}_i)(t)
\end{array}$$

Informally, the proposition $t_1 = t_2$ is true if and only if the interpretation, over the global state $\mathcal{G}_i$, of the two message terms is equal. The proposition $\mathbf{Knows}(j,t)$ is true if and only if the message m which is the interpretation of the message term t over the global state $\mathcal{G}_i$, belongs to $\overline{K}(w_j)$, where w_j is the set of messages in the local state of the agent whose identifier is j. Finally the proposition $\mathbf{Acts}_\lambda(j,t)$ is true if and only if the action α_{i+1} is $j.\lambda(m)$ where m is the message term t interpreted over the state $\mathcal{G}_i$.

Definition 13 (Formulae Interpretation). *Given a formula* f *and a trace* $\pi = \mathcal{G}_0 \cdot \alpha_1 \cdot \ldots \cdot \alpha_n \cdot \mathcal{G}_n$*, we have that:*

$$\begin{array}{lll}
\langle \pi,\, i\rangle \models p & \textit{iff} & \langle \pi,\, i\rangle \models_A p \\
\langle \pi,\, i\rangle \models \neg f & \textit{iff} & \langle \pi,\, i\rangle \not\models f \\
\langle \pi,\, i\rangle \models f_1 \wedge f_2 & \textit{iff} & \langle \pi,\, i\rangle \models f_1 \ \textit{and}\ \langle \pi,\, i\rangle \models f_2 \\
\langle \pi,\, i\rangle \models \exists s.f & \textit{iff} & \textit{there exists } s_0 \in I_n : \langle \pi,\, i\rangle \models f[s_0/s] \\
\langle \pi,\, i\rangle \models \Diamond_P f & \textit{iff} & \textit{there exists } j, 0 \le j \le i : \langle \pi,\, j\rangle \models f
\end{array}$$

The obvious extension of satisfiability over a trace is defined as follows $\pi \models f$ iff $\langle \pi,\, i\rangle \models f, \forall i : 0 \le i \le \mathrm{length}(\pi)$.

8.1 A Property Specification Example

In this section we briefly show how to express a properties over the Denning-Sacco protocol. In particular let us suppose that we want to specify the authenticity property that whenever the agent B receives a message $\{\langle x_k, x_a, x_t\rangle\}_{K_{bs}}$ apparently from the agent A, indeed that message has been previously sent by A. The properties can be expressed with the following formula:

$$\begin{array}{l} \forall b.\mathbf{name}(b) = B \ \wedge \mathbf{Acts}_{c_{ab}}(b, b.\{\langle x_k, x_a, x_t\rangle\}_{K_{bs}}\rangle) \Rightarrow \\ (\ \Diamond_P(\exists a.\mathbf{name}(a) = A \ \wedge \mathbf{Acts}_{\overline{c_{as}}}(a, a.y)) \wedge \\ \neg\ \Diamond_P(\exists b'.\mathbf{name}(b') = B \ \wedge b' \neq b \ \wedge \mathbf{Acts}_{c_{ab}}(b, b.\{\langle x_k, x_a, x_t\rangle\}_{K_{bs}}\rangle)) \wedge \\ (a.y = b.\{\langle x_k, x_a, x_t\}_{K_{bs}}) \ \wedge (b.\{\langle x_k, x_a, x_t\}_{K_{bs}} = b'.\{\langle x_k, x_a, x_t\}_{K_{bs}})\ ) \end{array}$$

In the previous formula it is required that whenever in a state the agent whose name is B performs a receive action on the channel c_{ab} getting the message $\widehat{\sigma_b}(\{\langle x_k, x_a, x_t\rangle\}_{K_{bs}}\rangle)$, where σ_b is the set of bindings of the agent B, then in a previous state $\mathcal{G}_j$, $j \leq i$, an agent whose name is A has sent that message on the channel c_{ab}. In addition, it is required that in no other state $\mathcal{G}_{j'}$ a different instance of B had received the same message since, in such a case, the message cannot be considered authentic.

It is not difficult to prove that there exists a trace where the formula is false. In particular, let us consider the trace corresponding to the attack illustrated in Section 3, and suppose to evaluate the formula over the state corresponding to step 3': indeed, in that state an instance of B has just received a message previously sent by an instance of A, but there exists a previous state where the same message is received by another instance of B.

9 Model Checking Issues and Conclusions

In this paper we have shown how the BRUTUS logics [10], a first order logics used within the BRUTUS model checker [9], can be used to specify security properties of a spi-calculus protocol, through the definition of a possible infinite labeled transition systems. The source of infinity can be found in the input rule of $\rightsquigarrow$ which involves a matching from a pattern and messages in the synthesis of the environment, which is an infinite set. In the defining the semantics of the BRUTUS embedded language, the authors found a similar problems, in [9]. To make the model checking feasible they observes that: (a) in the majority of the case the matching against a pattern is restrictive enough to limit the derivation of an infinite matching; (b) different methods can be found to restrict the model to a finite state space, without weakening the model, such the ones used in [14, 23] that allow message types.

In order to make our transition system finite state we intend to resort to similar strategies. In particular, input action semantics can be modified in such a way that only right typed messages are be accepted, making the pattern matching to be finite. Typed messages in a input action, has been recently studied by Heather, Lowe and Schneider in [14], where the authors show an efficient implementation which is independent of the protocol model used.

Interesting future work can be trying provide the original spi-calculus with a suitable logic, in such a way to perform automatic verification of properties expressed in the logic. Moreover, adequacy of the logic with respect to equivalences already existing in spi-calculus works, needed to be studied.

References

1. M. Abadi. Secrecy by Typing in Security Protocols. *Journal of the ACM*, 46(5):749–786, 1999.
2. M. Abadi and A. D. Gordon. Reasoning about Cryptographic Protocols in the Spi Calculus. In *Proc. of CONCUR '97: Concurrency Theory, 8th International Conference*, volume 1243 of *Lecture Notes in Computer Science*, pages 59–73. Springer-Verlag, 1997.
3. M. Abadi and A. D. Gordon. A Bisimulation Method for Cryptographic Protocols. *Nordic Journal of Computing*, 5(4):267–303, 1998.
4. M. Abadi and A. D. Gordon. A Calculus for Cryptographic Protocols. The Spi Calculus. Technical Report 149, Digital Equipment Corporation Systems Research Center, Palo Alto, California, 1998.
5. R. M. Amadio and D. Lugiez. On the Reachability Problem in Cryptographic Protocols. In *Proc. of 11th International Conference on Concurrency Theory (CONCUR 2000)*, volume 1877 of *Lecture Notes in Computer Science*, pages 380–394, 2000.
6. M. Boreale, R. De Nicola, and R. Pugliese. Proof Techiques of Crytpographic Protocols. In *Proc. of Annual IEEE Symposium on Logic in Computer Science – LICS*, 1999.
7. E. M. Clarke, E. A. Emerson, and A. P. Sistla. Automatic Verification of Finite-State Concurrent Systems Using Temporal Logic Specification. *ACM Transaction on Programming Languages and Systems*, 8(2):244–263, 1986.
8. E. M. Clarke, S. Jha, and W. Marrero. A Machine Checkable Logic of Knowledge for Protocols. In *Proc. of Workshop on Formal Methods and Security Protocols*, 1998.
9. E. M. Clarke, S. Jha, and W. Marrero. Using state space exploration and a natural deduction style message derivation engine to verify security protocols. In *Proc. of IFIP Working Conference on Programming Concepts and Methods (PROCOMET)*, 1998.
10. E. M. Clarke, S. Jha, and W. Marrero. Partial Order Reductions for Security Protocol Verification. In *Proc. of the International Conference for the Construction and Analysis of Systems – TACAS 2000*, volume 1785 of *Lecture Notes in Computer Science*, pages 503–518, 2000.
11. B. Donovan, P. Norris, and G. Lowe. Analyzing Library of Security Protocols using Casper and FDR. In *Proc. of the Workshop on Formal Methods and Security Protocols*, 1999.
12. L. Durante, R. Sisto, and A. Valenzano. A State-Exploration Technique for Spi-Calculus Testing-Equivalence Verification. In *Proc. of the IFIP TC6 WG6.1 Joint International Conference on Formal Description Techniques for Distributed Systems and Communication Protocols (FORTE XIII) and Protocol Specification, Testing and Verification (PSTV XX)*, pages 155–170. Kluwer Academic Publishers, 2000.

13. R. Focardi and F. Martinelli. A Uniform Approch for the Definition of Security Properties. In *Proc. of Congress on Formal Methods (FM'99)*, volume 1708 of *Lecture Notes in Computer Science*, pages 794–813. Springer-Verlag, 1999.
14. J. Heather, G. Lowe, and S. Schneider. How to Prevent Type Flaw Attacks on Security Protocols. In *Proc. of 13th IEEE Computer Security Foundations Workshop (CSFW'00)*, pages 32–43. IEEE Computer Society Press, 2000.
15. M. L. Hui and G. Lowe. Symplifing Transformations for Security Protocols. In *Proc. of 12th IEEE Computer Security Foundations Workshop (CSFW'99)*, pages 32–43. IEEE Computer Society Press, 1999.
16. G. Lowe. Breaking and fixing the Needham-Schroeder public-key protocol using FDR. In *Proc. of 19th IEEE Computer Security Foundations Workshop (CSFW'96)*, volume 1055 of *Lecture Notes in Computer Science*, pages 147–166. Springer-Verlag, 1996.
17. G. Lowe. A Family of Attacks upon Authentication Protocols. Technical Report 1997/5, Departement of Mathematics and Computer Science, Univ. of Leicester, 1997.
18. G. Lowe. Towards a completeness result for model checking for secutity protocols. In *Proc. of the 11th Computer Security Foundations Workshop (CSFW'98)*. IEEE Computer Society Press, 1998.
19. W. Marrero, E. Clarke, and S. Jha. Model Checking for Security Protocols. In *Proc. of the DIMACS Workshop on Design and Formal Verification of Secutity Protocols*, 1997.
20. C. A. Meadows. The NRL protocol analyzer: an overview. In *Proc. of the 2nd International Conference on the Practical Application of PROLOG*, 1994.
21. R. Milner, J. Parrow, and D. Walker. A Calculus of Mobile Processes, I. *Information and Computation*, 100(1):1–40, 1992.
22. R. Milner, J. Parrow, and D. Walker. A Calculus of Mobile Processes, II. *Information and Computation*, 100(1):41–77, 1992.
23. J. C. Mitchell, M. Mitchell, and U. Stern. Automated Analysis of Cryptographic Protocols Using Murϕ. In *Proceeding of the IEEE Symposium on Security and Privacy*, pages 141–151, 1997.
24. R. De Nicola and M. C. B. Hennessy. Testing Equivalences for Processes. *Theoretical Computer Science*, 34:83–133, 1984.
25. L. C. Paulson. Proving Properties of Security Protocols by Induction. Technical Report 409, Computer Laboratory, Univ. of Cambridge, 1996.
26. L. C. Paulson. Proving Properties of Security Protocols by Induction. In *Proc. of The 10th Computer Security Foundations Workshop*. IEEE Computer Society Press, 1997.
27. S. Schneider. Security properties and CSP. In *Proc. of the IEEE Symposium on Research in Security and Privacy*, pages 174–187, 1996.
28. S. Schneider. Verifying Authentication Protocols in CSP. *IEEE Transaction on Sofware Engineering*, 24(8):743–758, 1998.

A Appendix

In this appendix we describe into the details the transition rules defining the transition $\rightsquigarrow$, reported in Figure 6. The (A-NEW) rule explains how to update the set of messages w_i of an agent instance (i, P) when a new (i.e., not appearing anywhere in the protocol specification) name or key m, is defined using (**new** m).

The (A-SHA) is used to define a shared name or key m. A shared name may also appear in other agent instances specification. All the names coming from the same shared definition must be considered equal. The (A-PUB) makes the public name or key m, already defined at protocol level, to be added to the agent's set of messages w_i. As in the share definition all the public names coming from the same definition (**public** m) must be considered equal. The (A-INP) describes what happens when an agent instance is ready to perform an input from a public channel a. The input action is performed only if it is possible to extend the binding $\sigma_i = \mathcal{G}(i)\downarrow_3$ with a new binding $\sigma' \sqsupseteq \sigma$ such that the input term T can be instantiated to a message M. In addition M must belong to the environment's synthesis. If the extension is possible, the received message is added to the set of messages w_i of the agent instance (i, P), while its binding σ_i is substituted with σ'. The (A-OUT) rule describes what happens when an agent instance is ready to perform an output action on a public channel. The output is successfully executed only if the message term can be instantiated, using the local binding, to a ground message M. In such a case the message sent in output becomes part of the set of messages w_Ω of the environment. Rules (A-PLUS$_1$) and (A-PLUS$_2$) describe the behavior in the case of non-deterministic choice. Rules (A-PAR$_1$) and (A-PAR$_2$) describe the behavior in case of parallel composition.

The resulted transition system could be potentially infinite. One source of infinity is rule (A-INP) which involve the synthesis of the environment. It can cause infinity because there infinite number of new bindings $\sigma' \sqsupseteq \sigma_i$ could exist in such a way $\widehat{\sigma}'(T) = M$, for some $M \in \overline{K}(w_\Omega)$.

(A-NEW) $\langle \mathcal{G},\ (i, (\mathbf{new}\ m)P)\rangle \overset{\tau}{\rightsquigarrow} \langle \mathcal{G}',\ (i,P)\rangle$ where: $\begin{cases} \forall j,\ m \notin \mathcal{G}(j)\downarrow_2 \\ \mathcal{G}'(i)\downarrow_2 = \mathcal{G}(i)\downarrow_2 \cup \{m\} \end{cases}$

(A-SHA) $\langle \mathcal{G},\ (i, (\mathbf{shared}\ m)P)\rangle \overset{\tau}{\rightsquigarrow} \langle \mathcal{G}',\ (i,P)\rangle$ where: $\mathcal{G}'(i)\downarrow_2 = \mathcal{G}(i)\downarrow_2 \cup \{m\}$

(A-PUB) $\langle \mathcal{G},\ (i, (\mathbf{public}\ m)P)\rangle \overset{\tau}{\rightsquigarrow} \langle \mathcal{G}',\ (i,P)\rangle$ where: $\begin{cases} m \in \mathcal{G}(0)\downarrow_2 \\ \mathcal{G}'(i)\downarrow_2 = \mathcal{G}(i)\downarrow_2 \cup \{m\} \end{cases}$

(A-INP) $\dfrac{M \in \overline{K}(\mathcal{G}(0)\downarrow_2)\ ,\ \exists \sigma' \sqsupseteq \mathcal{G}(i)\downarrow_3 : \widehat{\sigma'}(T) = M}{\langle \mathcal{G},\ (i, a(T).P)\rangle \overset{i.a\langle M\rangle}{\rightsquigarrow} \langle \mathcal{G}',\ P\rangle}$ where: $\begin{cases} \mathcal{G}'(i)\downarrow_2 = \mathcal{G}(i)\downarrow_2 \cup \{M\} \\ \mathcal{G}'(i)\downarrow_3 = \sigma' \end{cases}$

(A-OUT) $\dfrac{\widehat{\mathcal{G}(i)\downarrow_3}(T) = M}{\langle \mathcal{G},\ (i, \overline{a}(T).P)\rangle \overset{i.\overline{a}\langle M\rangle}{\rightsquigarrow} \langle \mathcal{G}',\ (i,P)\rangle}$ where: $\mathcal{G}'(0)\downarrow_2 = \mathcal{G}(0)\downarrow_2 \cup \{M\}$

(A-PLU$_1$) $\dfrac{\langle \mathcal{G},\ (i,P)\rangle \overset{\alpha}{\rightsquigarrow} \langle \mathcal{G}',\ (i,P')\rangle}{\langle \mathcal{G},\ (i,P+Q)\rangle \overset{\alpha}{\rightsquigarrow} \langle \mathcal{G}',\ (i,P')\rangle}$ (A-PLU$_2$) $\dfrac{\langle \mathcal{G},\ (i,Q)\rangle \overset{\alpha}{\rightsquigarrow} \langle \mathcal{G}',\ (i,Q')\rangle}{\langle \mathcal{G},\ (i,P+Q)\rangle \overset{\alpha}{\rightsquigarrow} \langle \mathcal{G}',\ (i,Q')\rangle}$

(A-PAR$_1$) $\dfrac{\langle \mathcal{G},\ (i,P)\rangle \overset{\alpha}{\rightsquigarrow} \langle \mathcal{G}',\ (i,P')\rangle}{\langle \mathcal{G},\ (i,P \parallel Q)\rangle \overset{\alpha}{\rightsquigarrow} \langle \mathcal{G}',\ (i,P' \parallel Q)\rangle}$

(A-PAR$_2$) $\dfrac{\langle \mathcal{G},\ (i,Q)\rangle \overset{\alpha}{\rightsquigarrow} \langle \mathcal{G}',\ (i,Q')\rangle}{\langle \mathcal{G},\ (i,P \parallel Q)\rangle \overset{\alpha}{\rightsquigarrow} \langle \mathcal{G}',\ (i,P \parallel Q')\rangle}$

(A-MAT) $\dfrac{\mathcal{G}(i)\downarrow_3 T = \mathcal{G}(i)\downarrow_3 S}{\langle \mathcal{G},\ (i,[S{=}T].P)\rangle \overset{\tau}{\rightsquigarrow} \langle \mathcal{G}',\ (i,P)\rangle}$

Fig. 6. Transition rules for agents