A BRUTUS Logic for the Spi-Calculus*

S. Gnesi† D. Latella⋆
G. Lenzini†

† Istituto di Elaborazione dell'Informazione
Area della Ricerca CNR di Pisa, 56010 Pisa (Italy)
(e-mail:{gnesi,lenzini}@iei.pi.cnr.it)

⋆ Consiglio Nazionale Universitario per il Calcolo Elettronico
Area della Ricerca CNR di Pisa, 56010 Pisa (Italy)
(e-mail: d.latella@cnuce.cnr.it)

November 19, 2001

Abstract

A spi-calculus dialect and its BRUTUS logic is presented. The BRUTUS logic is a temporal first order logic defined within the BRUTUS model checker by Clarke, Jha and Marrero [10], and its use within spi-calculus like languages can help in expressing security properties. The spi-calculus dialect is then provided with an operational semantics based on labeled transition systems, that are also the model on which the satisfiability relation of BRUTUS logic formulas is defined. In this way we intend to perform a first step towards the definition of a theoretical framework for model checking logic properties on spi-calculus.

1 Introduction

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. As a classical example we can remind the Needham-Schroeder [24] public-key protocol, proposed in 1978, and found to be vulnerable the first time three years later by Denning and Sacco [11]. Security protocol problems appear to be well suited for the application of Formal Methods: in fact protocols specification is simple

*This work has been supported by the project "Coordinato CNR Agenzia 2000, *La prossima generazione di sistemi di trading on line: aspetti relativi alla sicurezza*".

enough to make their modeling tractable while finding the flaws is complex and counterintuitive enough to make an informal analysis too prone to errors to be reliable.

During the last years many researchers have dedicated their efforts to propose plenty of formal frameworks in which cryptographic protocols can be analyzed (e.g., see [5, 26, 30, 1, 17]). Among the different proposed approaches we are mainly interested in the *spi-calculus* [2] of Abadi and Gordon. The spi-calculus is a process algebra derived from the π-calculus [21, 22] with operators to encrypt and decrypt messages. Using the spi-calculus syntax is possible to express all the shared-key cryptographic protocols and, with light modifications [2], also other cryptographic paradigms such as public-keys, hash functions and digital signatures. In spi-calculus secrecy and integrity properties [3] can be naturally formalized as may-testing equivalences [25], and automatic strategies have been shown to be effective [4, 13] in verifying them. On the contrary other properties, such as anonymity [29] and non repudiation [31], seem not to be easily tractable [15, 29] as secrecy and authenticity are.

An alternative approach to the verification of security properties is the *model checking* [6] approach. Model checking has been successfully applied in security protocols analysis, and in the meaning known as *behavior-based*, where both the desired and the possible behavior of a system are given in the same notation (e.g., an automaton) and equivalence relations, or pre-orders, are used as correctness criterion, we remind:

- the numerous studies of Schneider (e.g., see [28]) and the ones of Lowe (e.g., see [18]), who independently provided a theoretical foundation for security properties definition within the framework of CSP process algebra, and who used FDR tool [19] model checker to find unknown flaws on many commercial protocols;
- the NRL compiler (e.g., see [20]) of Meadows, a special-purpose verification tool for the analysis of cryptographic protocols that checks security starting from an insecure state description and trying to reach an initial state to show that a flaw exists;
- the CVS/CoSeC Protocol Analyzer of Durante, Focardi and Gorrieri (e.g., see [12]) based on earlier studies about to use of Non Interference [16] to specify security properties (e.g., see [15, 14]) on protocols written in Crypto Security Process Algebra (CryptoSPA);
- the reachability analysis results of Mitchell et. al. over many famous protocols, with Murϕ, a general purpose state enumeration tool. (e.g., see [23]).

In this work we are prevalently interested in the *logic-based* model checking. The common strategy here consists in formulating properties as predicates on a formal model of the system, and trying to check their satisfiability. In particular using modal or temporal logics equipped with verification techniques, is possible to show that a system have o have not certain properties expressed in the logic.

Among logic-based approaches we remind the BRUTUS model checker of Clarke, Jha and Marrero [10]. Within BRUTUS, a first order linear temporal logic, we will call *BRUTUS logic*, it is used to formally formulate security properties. By using the BRUTUS logic it is possible to express not only secrecy and authenticity properties, but also others like anonymity [7]. As a consequence, although a wide study about its expressivity has not been presented yet by the authors, the use of BRUTUS logic appears to be promising.

In this work we propose a spi-calculus dialect[1] and its BRUTUS logic. The spi-calculus dialect is provided with an operational semantics, based on labeled transition systems, that are also the model on which we define the satisfiability relation of logic formulas. The present paper reports only on a part of a wider work whose main goal is to provide a theoretical framework for model checking logic properties on spi-calculus. To our knowledge, no such approach is available so far for the spi-calculus.

In Section 2 we will briefly describe the spi-calculus dialect syntax and the transition rules of the labeled transition systems used as semantics of the calculus. In Section 3 we introduce the BRUTUS logic syntax, and we explain how the satisfiability of BRUTUS logic formulas can be defined over the set of traces coming from the labeled transition systems, model of our calculus. Examples of protocol specification and of security properties expressed as BRUTUS logic formulas will be shown in Section 4, while in Section 5 we conclude.

2 A Spi Calculus Dialect

In this section we introduce the spi-calculus syntax, synthesized in Figure 1. Indeed our calculus has a similar syntax but a different semantics from the original spi-calculus. The most evident difference is in the communication paradigm. In particular:

- we do not provide our calculus with mobility, still possible in the spi-calculus, because all the channels are considered to be public;
- instead of a symmetric, synchronous communication via shared public channels, we have an asynchronous communication via the environment, a pool of messages to which anyone can access.

These choices are motivated by the fact that we want to simulate any possible unpredictable intruder's actions. For this reason any channel is an insecure medium, and it may be possible, in a communication between two principals, that a message transmitted is not the one received as a consequence of some unknown attack. Moreover other minor differences should be underlined:

- constructs for pair splitting and for decryption are embedded into the input primitive and they act as filters on the structure of messages coming from the environment pool.

[1]The use of a dialect, versus the original spi-calculus, simplifies most of the underneath theory, but an extension of our results to the whole spi-calculus is not excluded as a future work.

Here filtering out a message may be viewed as, in the original spi-calculus, receiving it and then going stuck because of a failure in decrypting or splitting.

- all processes, that are anonymous in the spi-calculus have a name; in addition more instances of a single process can exist and each instance is uniquely identified.

This last requirement strictly matches certain characteristics of the BRUTUS logic, where some predicate refers to instance numbers and to process names.

a		*labels* $\mathcal{A}$	$m,\ k,\ p$		*names* $\mathcal{N}$
x		*variables* $\mathcal{V}$	i		*proc id's* $\mathcal{I} = \{1, \dots, n\}$
$M,\ N ::=$	m	*messages* $\mathcal{M}$	$P,\ Q ::=$		*processes*
	$\mid \{M\}_k$			$\mathbf{0}$	nil
	$\mid \langle M,\ N\rangle$			$\mid a(T).P$	input
				$\mid \overline{a}(T).P$	output
$S,\ T ::=$	x	*message terms* $\mathcal{T}$		$\mid P \parallel Q$	parallel composition
	$\mid \{S\}_x$			$\mid P + Q$	non-determinist choice
	$\mid \langle S,\ T\rangle$			$\mid (\mathbf{new}\ m)P$	new local name
	$\mid M$			$\mid [S{=}T]P$	match
				$\mid p$	process name
$Z ::=$		*protocol* $\mathcal{Z}$			
	$A \Vert\!\vert Z$	process list	$p \stackrel{def}{=} P$		*name definition*
	$(\nu m)Z$	name restriction			
	$\mid A$	process instance			
$A ::=$	(i, p)	*process instances*			

Figure 1: Syntax of the spi-calculus dialect.

In our calculus a *protocol* Z is a parallel composition of n process instances, $(1, p_1) \Vert\!\vert \dots \Vert\!\vert (h, p_h)$, where $\{i, \dots, h\} \subseteq \{1, \dots, n\}$. A name m can be restricted, over the whole protocol, with (νm) in order to make it initially invisible either to the environment or to the other process instances. Each process instance (i, p) is identified by a name p, defining a spi-calculus process, and by an unique instance identifier i. The presence of more instances is to be intended, as usual in cryptographic analysis, as describing more runs of the protocol.

The operational semantics of our calculus is based on *two* labeled transition systems, describing respectively transitions at protocol level (the outermost) and transitions at process instance level (the innermost). The outermost system (see Figure 2) has form $\langle \mathcal{G},\ Z\rangle \stackrel{\alpha}{\longmapsto} \langle \mathcal{G}',\ Z'\rangle$ and it represents atomic transitions of the protocol Z and the relative changes in the global state $\mathcal{G}$. The innermost system (see Figure 3) has form $\langle \mathcal{G},\ (i, P)\rangle \stackrel{\alpha}{\leadsto} \langle \mathcal{G}',\ (i, P')\rangle$ and it describes how a process instances (i, P) evolves and how the global state changes. In fact, in the operational semantics each process instance has its own local state, and the global state $\mathcal{G}$ of the entire system is the composition of these local states with the local state of the environment. Supposing that at most n process instances are involved in a protocol, the global state $\mathcal{G}$ can be thought as an array of $n+1$ local states. Precisely:

$$\text{(P-PAR1)}\ \frac{\langle \mathcal{G},\ Z_1\rangle \overset{\alpha}{\longmapsto} \langle \mathcal{G}',\ Z_1'\rangle}{\langle \mathcal{G},\ Z_1 \| Z_2\rangle \overset{\alpha}{\longmapsto} \langle \mathcal{G}',\ Z_1' \| Z_2\rangle} \qquad \text{(P-PAR2)}\ \frac{\langle \mathcal{G},\ Z_2\rangle \overset{\alpha}{\longmapsto} \langle \mathcal{G}',\ Z_2'\rangle}{\langle \mathcal{G},\ Z_1 \| Z_2\rangle \overset{\alpha}{\longmapsto} \langle \mathcal{G}',\ Z_1 \| Z_2'\rangle}$$

$$\text{(P-AGE)}\ \frac{\langle \mathcal{G},\ (i,P)\rangle \overset{\alpha}{\leadsto} \langle \mathcal{G}',\ (i,P')\rangle}{\langle \mathcal{G},\ (i,P)\rangle \overset{\alpha}{\longmapsto} \langle \mathcal{G}',\ (i,P')\rangle} \qquad \text{(P-RES)}\ \frac{\langle \mathcal{G},\ Z\rangle \overset{\alpha}{\longmapsto} \langle \mathcal{G}',\ Z'\rangle}{\langle \mathcal{G},\ (\nu m)Z\rangle \overset{\alpha}{\longmapsto} \langle \mathcal{G}',\ Z'\rangle}$$

Figure 2: Transition rules for protocols

(a) $\forall i \in \mathcal{I}$, $\mathcal{G}(i) = (p, W, \sigma)$ is the local state of the i-th process instance, the one whose identifier is i, where $p \in \mathcal{N}$ 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[2];

(b) $\mathcal{G}(0) = (\Omega, W_\Omega, \perp)$ is the local state of the environment, which is like a process instance state with an undefined binding function.

Given a local state (p, W, σ), we indicate with $\mathsf{Name}(p, W, \sigma)$, $\mathsf{Know}(p, W, \sigma)$, $\mathsf{Bind}(p, W, \sigma)$ respectively the first component p, the second component W, and the third component σ. Finally with $\sigma(T)$ we indicate the message term obtained substituting each variable x in T with the corresponding value $\sigma(x)$. Given a protocol specification Z, the state initial state of the transition system is $q_0 = \langle \mathcal{G}_0, Z_0\rangle$ so defined:

(a) $Z_0 = Z$;

(b) $\mathcal{G}_0(0) = (\Omega, W_\Omega, \perp)$, is the initial local state of the environment, where Ω is the name of the environment, W_Ω is the initial set of messages known by the environment composed by all the free names[3] in Z, and $\perp$ is the empty set of bindings of the environment;

(c) $\mathcal{G}_0(h) = (p_h, W_h, \perp)$ for all $h \in \mathcal{I}$, is the initial local state of the process instance h. More precisely, p_h is the name of the process P if $p_h \stackrel{def}{=} P$, W_h is the set of messages it knows, composed by all the free names in Z and by all the bound names[4] in P_h, and $\perp$ is its initial set of bindings.

We briefly explain the most important transition rules, reported in Figures 2 and Figure 3. The (P-AGE) rule makes a single process transition rise up at protocol transition level. The (A-NEW) rule is used to define a new local name. The (A-INP) transition describes what happens when a process instance (i, P) is ready to perform an input action. The receiving is performed only if it is possible to extend the binding $\sigma = \mathsf{Bind}(\mathcal{G}(i))$, local to the process instance, with a new binding $\sigma' \sqsupseteq \sigma$ such that the input term T can be instantiated to a message M. The message M must belong to the set of messages, $KS(W_\Omega)$, that

[2]Talking about binding functions, with the symbol $\perp$ 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.

[3]Free and bound names are supposed defined as usual. In addition we suppose that the scope of $(\nu m)Z$, which binds m, is Z.

[4]Here we suppose that the scope of $(\textbf{new}\ m)P$, which binds m, is P.

$$\text{(A-NEW)}\ \frac{}{\langle \mathcal{G},\ (i, (\mathbf{new}\ m)P)\rangle \overset{\tau}{\rightsquigarrow} \langle \mathcal{G}',\ (i, P)\rangle} \quad \text{where} \begin{cases} \mathsf{Know}(\mathcal{G}'(i)) = \mathsf{Know}(\mathcal{G}(i)) \cup \{m\}, \\ \mathsf{Know}(\mathcal{G}'(j)) = \mathsf{Know}(\mathcal{G}(j)), \forall j \neq i; \end{cases}$$

$$\text{(A-INP)}\ \frac{M \in KS(W_\Omega)\ ,\ \exists \sigma' \sqsupseteq \mathsf{Bind}(\mathcal{G}(i)) : \sigma'(T) = M}{\langle \mathcal{G},\ (i, a(T).P)\rangle \overset{i.a\langle M\rangle}{\rightsquigarrow} \langle \mathcal{G}',\ P\rangle} \quad \text{where} \begin{cases} W_\Omega = \mathsf{Know}(\mathcal{G}(0)); \\ \mathsf{Know}(\mathcal{G}'(i)) = \mathsf{Know}(\mathcal{G}(i)) \cup \{M\}, \\ \mathsf{Know}(\mathcal{G}'(j)) = \mathsf{Know}(\mathcal{G}(j)), \forall j \neq i; \\ \mathsf{Bind}(\mathcal{G}'(i)) = \sigma' \end{cases}$$

$$\text{(A-OUT)}\ \frac{\mathsf{Bind}(\mathcal{G}(i))(T) = M}{\langle \mathcal{G},\ (i, \overline{a}(T).P)\rangle \overset{i.\overline{a}\langle M\rangle}{\rightsquigarrow} \langle \mathcal{G}',\ (i, P)\rangle} \quad \text{where } \mathsf{Know}(\mathcal{G}'(0)) = \mathsf{Know}(\mathcal{G}(0)) \cup \{M\}$$

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

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

$$\text{(A-PAR}_2\text{)}\ \frac{\langle \mathcal{G},\ (i, Q)\rangle \overset{\alpha}{\rightsquigarrow} \langle \mathcal{G}',\ (i, Q')\rangle}{\langle \mathcal{G},\ (i, P \parallel Q)\rangle \overset{\alpha}{\rightsquigarrow} \langle \mathcal{G}',\ (i, P \parallel Q')\rangle} \qquad \text{(A-MAT)}\ \frac{\mathsf{Bind}(\mathcal{G}(i))(T) = \mathsf{Bind}(\mathcal{G}(i))(S)}{\langle \mathcal{G},\ (i, [s{=}T].P)\rangle \overset{\tau}{\rightsquigarrow} \langle \mathcal{G}',\ (i, P)\rangle}$$

$$\text{(A-CON)}\ \frac{\langle \mathcal{G},\ (i, P)\rangle \overset{\alpha}{\rightsquigarrow} \langle \mathcal{G}',\ (i, P')\rangle}{\langle \mathcal{G},\ (i, p)\rangle \overset{\alpha}{\rightsquigarrow} \langle \mathcal{G}',\ (i, P')\rangle} \quad \text{where } p \overset{def}{=} P$$

Figure 3: Transition rules for process instances.

the environment can generate[5] starting from the messages W_Ω it knows locally. If the transition is successful the received message M is added to the set W_i of messages locally known by the process instance (i, P), while its local binding σ is substituted with σ'. The (A-OUT) rule describes what happens when a process instance is ready to perform an output action on a public channel. The output is successfully executed only if the output message term T can be instantiated, using the binding function σ local of the instance, 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.

3 BRUTUS Logic

The syntax of BRUTUS logic [9], is presented in Figure 4. Its alphabet symbols are used to built *messages*, *terms*, *atomic propositions* and *formulae* that constitute the syntax of the BRUTUS logic. Terms are split in two different sets: *process-instance* terms $\mathcal{T}_s$ and *message* terms $\mathcal{T}_m$. Informally, a process-instance term can either be a process identifier $j \in \mathcal{I}$, or a process variable $s \in \mathcal{V}_s$. A

[5] Here we suppose to have some reasonable rules for message manipulation, such as encryption, decryption, pairing and splitting as the one in [15]. With $KS(W)$ we intend all the messages in the set W or the ones that can be generated from it.

$\mathcal{N}$ is a set of *names*; $\mathcal{L}$ is a set of *labels*;
$\mathcal{I}$ is a finite set of *process id's*; $\mathcal{V}$ is a set of *msg variables.*
$\mathcal{V}_s$ is a set of *proc instance vars*;

$f ::= q \mid \neg f \mid f_1 \wedge f_2$ $\mid \exists s.f \mid \Diamond_P f$		formulae
$\rho ::= \mathbf{Knows}(mp, mt)$ $\mid \mathbf{Acts}_\lambda(mp, mt)$ $\mid mt_1 = mt_2$		atomic propositions
$mp ::= j \mid s$	$s \in \mathcal{V}_s, j \in \mathcal{I}$	process-instance terms $\mathcal{T}_s$
$mt ::= M \mid x \mid mp.mt$ $\mid \mathbf{name}(mp)$ $\mid \langle mt_1,\, mt_2 \rangle \mid \{mt_1\}_{mt_2}$	$x \in \mathcal{V}, M \in \mathcal{M}$	msg terms $\mathcal{T}_m$

Figure 4: The Syntax of the BRUTUS Logic

message term can be either a message M, or a message variable $x \in \mathcal{V}$, or a pair of message terms, or an encryption of a message term. The message term $\mathbf{name}(s)$ is the name of the process-instance term s. The message term $mp.mt$ is the message term mt interpreted in the local state of the process-instance mp. Atomic propositions can be:

(a) $\mathbf{Knows}(mp, mt)$, which is a predicate on the knowledge of the process instance mp of the message mt;

(b) $\mathbf{Acts}_\lambda(mp, mt)$ which is a predicate on the action λ, performed by mp, involving the message mt;

(c) $mt_1 = mt_2$, which is an equality test over message terms.

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 process-instance variable s within the formula f.

The semantics is defined over our transition systems. The interpretation of BRUTUS formulae is defined over *traces* $\pi = q_0 \cdot \alpha_1 \cdot q_1 \cdot \ldots \cdot \alpha_n \cdot q_n$, where $q_i = \langle \mathcal{G}_i,\, Z_i \rangle$, and $q_i \overset{\alpha_{i+1}}{\longmapsto} q_{i+1}$ is a possible transition from q_i. Process instance variables s ranges over the set $\mathcal{I}$ of process identifiers, while message variables x over the set $\mathcal{M}$ of messages. Message terms are interpreted on *Glob*, the set of all global states, as in the following:

Definition 1 (Message Term Interpretation) *Given a global state $\mathcal{G} \in Glob$ and a message term mt,* message term interpretation*, is the function* $\mathbb{M}$:

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

$$
\begin{array}{lcll}
\mathbb{M}(\mathcal{G})(M) & = & M, & \textit{where } M \in \mathcal{M} \\
\mathbb{M}(\mathcal{G})(\langle mt_1, \, mt_2 \rangle) & = & \langle \mathbb{M}(\mathcal{G})(mt_1), \, \mathbb{M}(\mathcal{G})(mt_2) \rangle & \\
\mathbb{M}(\mathcal{G})(\{mt_1\}_{mt_2}) & = & \{\mathbb{M}(\mathcal{G})(mt_1)\}_{\mathbb{M}(\mathcal{G})(mt_2)} & \\
\mathbb{M}(\mathcal{G})(j.mt) & = & \sigma(mt), & \textit{where } \sigma = \mathsf{Bind}(\mathcal{G}(j)) \\
\mathbb{M}(\mathcal{G})(\mathbf{name}(j)) & = & p, & \textit{where } p = \mathsf{Name}(\mathcal{G}(j))
\end{array}
$$

The interpretation of $j.mt$ is the message obtained instantiating all the variables appearing in mt, using the set of bindings of the process instance whose identifier is j; the interpretation of $\mathbf{name}(j)$ is the message which represents the name of the process instance whose identifier is j. Formulae of the logic are interpreted over traces of a protocol model. We first define interpretation of atomic proposition on a particular state. We will write $\langle \pi, \, i \rangle \models_A \rho$ to mean that the atomic proposition ρ is satisfied on the state q_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 $q_i = \langle \mathcal{G}_i, \, Z_i \rangle$.

Definition 2 (Atomic Proposition Interpretation) *Given a trace* $\pi = q_0 \cdot \alpha_1 \cdot q_1 \cdot \ldots \cdot \alpha_n \cdot q_n$, *we have that for* $1 \leq i \leq n$*:*

$$
\begin{array}{lll}
\langle \pi, \, i \rangle \models_A mt_1 = mt_2 & \textit{iff} & \mathbb{M}(\mathcal{G}_i)(mt_1) = \mathbb{M}(\mathcal{G}_i)(mt_2) \\
\langle \pi, \, i \rangle \models_A \mathbf{Knows}(j, mt) & \textit{iff} & \mathbb{M}(\mathcal{G}_i)(mt) \in KS(\mathsf{Know}(\mathcal{G}_i(j))) \\
\langle \pi, \, i \rangle \models_A \mathbf{Acts}_\lambda(j, mt) & \textit{iff} & \alpha_{i+1} = j.\lambda(M), \ \textit{where } M = \mathbb{M}(\mathcal{G}_i)(t)
\end{array}
$$

Informally, the proposition $mt_1 = mt_2$ is true if and only if the interpretation, over the global state $\mathcal{G}_i$, where $q_i = \langle \mathcal{G}_i, \, Z_i \rangle$, of the two message terms is equal. The proposition $\mathbf{Knows}(j, mt)$ is true if and only if the message M which is the interpretation of the message term mt over the global state $\mathcal{G}_i$, belongs to $KS(W_j)$, where W_j is the set of messages in the local state of the process whose identifier is j. Finally the proposition $\mathbf{Acts}_\lambda(j, mt)$ is true if and only if the action α_{i+1} is $j.\lambda(M)$ where M is the message obtained interpreting the term mt over the global state $\mathcal{G}_i$.

Definition 3 (Formulae Interpretation) *Given a formula* f *and a trace* $\pi = q_0 \cdot \alpha_1 \cdot \ldots \cdot \alpha_n \cdot q_n$, *we have that for* $1 \leq i \leq n$*:*

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

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

4 Some Running Examples

In this section we will present two examples of protocol specification in our spi-calculus dialect and security properties formalization via the BRUTUS logic.

The first example, quite general, is prevalently aimed to show the use of the dialect and of the BRUTUS logics. The second one is a more specific example showing how the BRUTUS logics can be used to express anonymity.

4.1 A Key-Exchange Protocol

Let us suppose to have the following key-exchange protocol where two principals A and B and a trusted server S are involved:

Protocol 1 (Key Exchange)

$$\begin{array}{lll} 1. & A & \rightarrow S : A, B \\ 2. & S & \rightarrow A : \{K_{AB}\}_{K_{AS}}, \{A, K_{AB}\}_{K_{BS}} \\ 3. & A & \rightarrow B : \{A, K_{AB}\}_{K_{BS}} \end{array}$$

Informally, in the first step A sends to S a message composed of its name and the name of B. With that message A intends to request a new session key to communicate with B. The server S generates the session key K_{AB} and encrypts two copies of it: one using the key K_{AS}, shared with A, and the other using the key K_{BS}, shared with B. In the second copy, which is reserved for B, S encrypts also information about A's identity. Then, S sends (step 2) both copies to A, which in turn forwards (step 3) to B the message reserved to it. Once received, B can determine that A wants to start a confidential communication with it, and retrieve the session key it wants to use.

The protocol can be written in our spi-calculus dialect as in the following: [6]

$$\begin{array}{rcl} KE & \stackrel{def}{=} & (1, S) \| (2, A) \| (3, B) \\ A & \stackrel{def}{=} & \overline{c_{as}}(\langle A,\ B\rangle).c_{as}(\langle\{x_1\}_{K_{AS}}, y_1\rangle).\overline{c_{ab}}(y_1).\mathbf{0} \\ B & \stackrel{def}{=} & c_{ab}(\{\langle A,\ x_2\rangle\}_{K_{BS}}).\mathbf{0} \\ S & \stackrel{def}{=} & (\mathbf{new}\ K_{AB})c_{as}(\langle A,\ B\rangle). \\ & & \overline{c_{as}}(\langle\{K_{AB}\}_{K_{AS}}, \{\langle A, K_{AB}\rangle\}_{K_{BS}}\rangle).\mathbf{0} \end{array}$$

The protocol specification shows up three process instances, one for each process running the protocol. Three communication channels, c_{ab}, c_{as} and c_{bs}, one for each couple of processes, are used as media to exchange messages: indeed, because in our model all the channel are public, c_{ab}, c_{as} and c_{bs} must be considered more as labels with which to distinguish between different input or output actions than real channel names.

Input message terms are written in such a way to explicit *as much as possible* of the structure of the message expected because, as explained in Section 2

[6] All the local variables are named in such a way to be distinct

(item three) input terms act as filters on the messages coming from the environment's knowledge. For example, in the definition of process B, the input term $\{\langle A, x_2\rangle\}_{K_{BS}}$ makes explicit that the message expected is a pair $\langle A, x_2\rangle$, encrypted with the shared key K_{BS}; moreover the first item of the pair must be the name A, of process A. From the semantics of our calculus this will mean that the process B will accept only messages with that structure, exactly as the process B tried to decrypt and split each input message as soon as the input action is performed, going on if and only if the message results to be in fact encrypted with key K_{BS} and composed by a pair whose first component is the name A. A different observation must be done for the input term $\langle\{x_1\}_{K_{AS}}, y_1\rangle$, in the definition of process A. In this case not all the part can be make explicit. In particular A cannot "see inside" in the variable y_1, because the message expected in it (from the informal description of the protocol), is encrypted with a key, K_{BS} that A does not know. To explicit the structure of input messages "as much as possible" should be intended just in this meaning: messages in clear, or encrypted with known keys should be received into a message term reflecting the message structure, while messages encrypted with unknown keys must be received into a single variable.

The BRUTUS logic can be used to express security properties, for example the following authenticity property:

$$
\begin{aligned}
f \;\stackrel{def}{=}\;\; & \forall b.\exists a.\mathbf{name}(b) = B \;\wedge\; \mathbf{name}(a) = A \;\wedge \\
& \mathbf{Acts}_{c_{ab}}(b, \{\langle A,\, x_2\rangle\}_{K_{BS}}) \rightarrow \Diamond_P \mathbf{Acts}_{\overline{c_{ab}}}(a, y_1) \wedge \\
& a.y_1 = b.\{\langle A,\, x_2\rangle\}_{K_{BS}}
\end{aligned}
\tag{1}
$$

Informally the formula (1) says that whenever a process instance, of the principal B, receives the message $\{\langle A, x_2\rangle\}_{K_{BS}}$ then there exist a process instance, of principal A, which has previously sent a message y_1 to B, and y_1 is exactly the same message as $\{\langle A, x_2\rangle\}_{K_{BS}}$ received by B.

4.2 An Anonymity Protocol

In this section we will show how BRUTUS logic can be used to express an anonymity property. It is not our intention to deeply investigate in the expressive power of BRUTUS logic, but we should give just an example of its expressiveness and of its usefulness in the spi-calculus framework.

As a simple anonymous protocol we will refer to a modified version of the one used within the work of Schneider [27], where a charity accepting donations is considered. The charity S, accepts donation from two donors, A and B, which want their donations to remain anonymous to the environment. For this reason they encrypt their donation using a key shared with the charity[7]. The charity always announces its thanks after a donation, declaring that a donation of so many dollars has been just received, but omitting the name of the donor.

[7] One can think that the donation is closed inside an envelop.

In the following we formalize, just for sake of example, only two runs of the protocol in our spi-calculus dialect, also if two runs are probably insufficient for really analyzing anonymity.

Protocol 2 (Donation One)

$$
\begin{aligned}
DO &\stackrel{def}{=} (\nu K)(1, A) \| (2, B) \| (3, S) \\
&\quad \| (4, A) \| (5, B) \| (6, S) \\
A &\stackrel{def}{=} \overline{gives}(\langle \{5\}_K,\, A \rangle).\mathbf{0} \\
B &\stackrel{def}{=} \overline{gives}(\langle \{10\}_K,\, B \rangle).\mathbf{0} \\
S &\stackrel{def}{=} gives(\langle \{x\}_K,\, y \rangle).\overline{thanks}(x).\mathbf{0}
\end{aligned}
$$

In the meaning of Schneider [27], anonymity is concerned with protecting the identity of processes respect to particular actions or messages. Informally a message that could have originated from one anonymous process could equally have originated from any other process.

In the case of Protocol 2 anonymity would require that it is not possible, to the environment, to distinguish A and B when a charity thank for a donation. Protocol 2 does not guarantee anonymity. In fact, an external observer could observe that whenever the charity thanks to have received 5 dollars, donor A but not B, has previously donated some money, and then infer that A have indeed given 5 dollars. Using BRUTUS logic we can express anonymity with the following formulas:

$$
\begin{aligned}
f_A = \; & \forall s\, \mathbf{name}(s) = S \wedge \mathbf{Acts}_{\overline{thanks}}(s, 5) \rightarrow \\
& (\exists a\, \mathbf{name}(a) = A \wedge \\
& \mathbf{Knows}(\Omega, A) \wedge \neg \mathbf{Knows}(\Omega, K) \wedge \\
& \Diamond_P(\mathbf{Acts}_{\overline{gives}}(a, \langle \{5\}_K,\, A \rangle)))
\end{aligned} \tag{2}
$$

$$
\begin{aligned}
f_B = \; & \forall s\, \mathbf{name}(s) = S \wedge \mathbf{Acts}_{\overline{thanks}}(s, 5) \rightarrow \\
& (\exists b\, \mathbf{name}(b) = B \wedge \\
& \mathbf{Knows}(\Omega, B) \wedge \neg \mathbf{Knows}(\Omega, K) \wedge \\
& \Diamond_P(\mathbf{Acts}_{\overline{gives}}(b, \langle \{5\}_K,\, B \rangle)))
\end{aligned} \tag{3}
$$

Formula (2) affirms that whenever an instance of the charity S thanks publicly for having received a donation of 5 dollars, then an instance of A have effectively donated some dollars. In addition the environment knows the process name A, but does not knows the encryption key K. This assure that the environment does not know the amount of money given by A. Formula (3) affirms the same for B.

Informally one can be convinced that $\models f_A$, and $\models \neg f_B$, and as a consequence the environment may infer that indeed was the donor A that gives 5 dollars. Let us now consider a different version of the protocol.

Protocol 3 (Donation Two)

$$
\begin{aligned}
DO &\stackrel{def}{=} (\nu K)(1, A) \| (2, B) \| (3, S) \\
&\quad \| (4, A) \| (5, B) \| (6, S) \\
A &\stackrel{def}{=} (\overline{gives}(\langle \{5\}_K, A \rangle) + \overline{gives}(\langle \{10\}_K, A \rangle)).\mathbf{0} \\
B &\stackrel{def}{=} (\overline{gives}(\langle \{10\}_K, B \rangle) + \overline{gives}(.\langle \{5\}_K, B \rangle)).\mathbf{0} \\
S &\stackrel{def}{=} gives(\langle \{x\}_K, y \rangle).\overline{thanks}(x).\mathbf{0}
\end{aligned}
$$

In this case anonymity is guaranteed. In fact it easy to be agree with the fact $\models f_A$ and $\models f_B$, indicating that the environment is not able to distinguish between the donors A and B whenever the charity thanks to have received 5 dollars.

5 Conclusion

In this work a spi-calculus dialect and its BRUTUS logic has been presented. The BRUTUS logic is the temporal first order logic defined by Clarke, Jha and Marrero [10] in their BRUTUS model checker, and its use can help in expressing security properties in spi-calculus like languages. In fact in BRUTUS logic we are able to formalize not only secrecy and authenticity, usually considered in this calculus, but also other properties such as, for example, anonymity.

The spi-calculus dialect introduced is then provided with a labeled transition systems semantics, and the BRUTUS logic satisfiability relation is then defined over the set of traces coming from the semantics. In this way we intend perform a first step towards the definition of a theoretical framework for model checking logic properties on spi-calculus.

The model checking approach is effective only in case of finite state systems: in fact the *model checker* algorithm checks the satisfiability of the given formula with time complexity which is a function of the size of the model in number of states and transitions, and of the length of the formula [6]. In our definition the labeled transition systems are infinite branching and model checking techniques cannot directly applied. In BRUTUS model checker this problem has been solved limiting the size of derivation that the environment is allowed to perform when generating messages (see [8]) in its knowledge and we could do the same, but at the moment our main intention is to investigate the usefulness of a logic as a mean for expressing properties in spi-calculus framework. Further research efforts will be oriented to introduce formalized strategies to make finite state the semantics, in order to make the model checking effective.

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. A Calculus for Cryptographic Protocols. The Spi Calculus. Technical Report 149, Digital Equipment Corporation Systems Research Center, Palo Alto, California, 1998.

[3] M. Abadi and A. D. Gordon. Reasoning about the Cryptographic Protocols in the Spi Calculus. In *Proc. of the International Conference on Concurrency Theory (CONCUR'98)*, 1998.

[4] M. Boreale. Symbolic trace analysis of cryptographic protocols in the spi-calculus. In *Proc. of ICALP 2001*, 2001.

[5] M. Burrows, M. Abadi, and R. Needham. A logic of authentication. In *Proc. of the Royal Society of London*, volume 426 of *Lecture Notes in Computer Science*, pages 233–271. Springer-Verlag, 1989.

[6] 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.

[7] 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.

[8] 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.

[9] 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, pages 503–518, 2000.

[10] E. M. Clarke, S. Jha, and W. Marrero. Verifying security protocols with Brutus. *ACM Transactions on Software Engineering and Methodology*, 9(4):443–487, 2000.

[11] D. E. Denning and M. S. Sacco. Timestamps in key distribution protocols. *Communications of the ACM*, 24(8):533–536, 1981.

[12] A. Durante, R. Focardi, and R. Gorrieri. A Compiler for Analyzing Cryptographic Protocols Using Noninterference. *ACM Transactions on Software Engineering and Methodology*, 9, 2000.

[13] 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.

[14] R. Focardi, R. Gorrieri, and F. Martinelli. NonInterference for the Analysis of Cryyptographic Protocols. In *Proc. of the ICALP'00*. Springer-Verlag, 2000.

[15] 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.

[16] J. A. Goguen and J. Meseguer. Security policies and security models. In *Proceedings of the IEEE Symposium on Research in Security and Privacy*, pages 11–20. IEEE Computer Society Press, 1982.

[17] A. D. Gordon and A. Jeffrey. Authenticity by Typing for Security Protocols. In *Proc. 14th IEEE Computer Security Foundations Workshop (CSFW 2001)*, pages 145–159. IEEE Computer Society, 2001.

[18] G. Lowe. Breaking and fixing the Needham-Schroeder public-key protocol using FDR. In *Lecture Notes in Computer Science*, volume 1055, pages 147–166. Springer-Verlag, 1996.

[19] Formal System (Europe) Ltd. *Failures-Divergence Refinement, FDR2.* 2000.

[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. M. Needham and M. D. Schroeder. Using Encryption for Authentication in Large Network of Computer. *Communication of the ACM*, 21(12):993–999, 1978.

[25] R. De Nicola and M. C. B. Hennessy. Testing Equivalences for Processes. *Theoretical Computer Science*, 34:83–133, 1984.

[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.

[29] S. Schneider and A. Sidiropoulos. CSP and Anonymity. In *Proc. of the European Symposium on Research in Computer Security (ESORIC'96)*, volume 1146 of *Lecture Notes in Computer Science*, pages 198–218. Springer-Verlag, 1996.

[30] J. Thayer, J. Herzog, and J. Guttman. Strand spaces: Why is a security protocol correct? In *Proc. of the 19th IEEE Computer Society Symposium on Research in Security and Privacy*, 1998.

[31] J. Zhou and D. Gollmann. Observations on Non-repudiation. In *Proc. of ASIACRYPT'96: Advances in Cryptology*, volume 1163 of *Lecture Notes in Computer Science*, pages 133–147. Springer-Verlag, 1996.