A STATE-EXPLORATION TECHNIQUE FOR SPI-CALCULUS TESTING-EQUIVALENCE VERIFICATION

Luca Durante, Riccardo Sisto
Politecnico di Torino
Dipartimento di Automatica e Informatica
corso Duca degli Abruzzi 24
I-10129 Torino
Italy
{luca.durante, riccardo.sisto}@polito.it

Adriano Valenzano
Istituto di Ricerca sull'Ingegneria
delle Telecomunicazioni e dell'Informazione
Politecnico di Torino
corso Duca degli Abruzzi 24
I-10129 Torino
Italy
adriano.valenzano@polito.it

Abstract Several verification techniques based on theorem proving have been developed for the verification of security properties of cryptographic protocols specified by means of the spi calculus. However, to be used successfully, such powerful techniques require skilled users. Here we introduce a different technique which can overcome this drawback by allowing users to carry out the verification task in a completely automatic way. It is based on the definition of an extended labeled transition system, where transitions are labeled by means of the new knowledge acquired by the external environment as the result of the related events. By means of bounding the replication of parallel processes to a finite number, and by using an *abstract* representation of all explicitly allowed values in interactions between the spi process and the environment, the number of states and transitions remains finite and tractable, thus enabling the use of state-space exploration techniques for performing verification automatically.

Keywords: Spi Calculus, Cryptographic Protocols, Testing Equivalence.

The original version of this chapter was revised: The copyright line was incorrect. This has been corrected. The Erratum to this chapter is available at DOI: 10.1007/978-0-387-35533-7_26

Tommaso Bolognesi and Diego Latella (Eds.), *Formal Methods for Distributed System Development.*

1. INTRODUCTION

Due to the increasing importance of secure distributed applications such as electronic commerce, formal verification of cryptographic protocols is being extensively studied by several researchers. Some of them have investigated proof techniques, based on various proof systems and description formalisms [10, 11]. Although partial automation of proofs is possible using theorem provers, this approach is generally highly time consuming and requires a lot of expertise. An alternative simpler and quicker approach is to use state exploration methods, such as model checking [4, 5, 6, 7, 8]. This requires modeling protocol behaviors as reasonably sized finite state systems, which generally entails introducing simplifying assumptions that can reduce the accuracy of the analysis. Nevertheless, this kind of verification has the invaluable advantage of being fully automatic.

Both theorem proving and state exploration have been used with several description formalisms. In this paper attention is focused on spi calculus [1], a process algebra derived from the π-calculus [9] with some simplifications and the addition of cryptographic operations. The main strength of spi calculus with respect to other similar formalisms stands in its simplicity in describing cryptographic protocols and their security requirements. In particular, in [1] it is shown how security properties such as authentication and secrecy can easily be expressed by means of a testing equivalence notion. For example, if $P(M)$ is the description of a cryptographic protocol to exchange a secret message M, secrecy can be simply expressed saying that for any M', $P(M)$ and $P(M')$ must be testing equivalent, i.e. any tester process must be unable to distinguish their behaviors. Instead, when other specification formalisms are used, it is generally required that both the protocol and the attacker behavior be specified. The attacker specification is not only extra work, but also a potential weak point, because it is somewhat arbitrary and might not include some possible attacks on the protocol. Expressing properties in terms of testing equivalence means implicitly considering any attacker that can be specified in spi calculus. This is because the tester process actually represents the protocol environment, i.e. the attacker.

The main problem that remains with the spi calculus approach is the checking of testing equivalence in an efficient and easy way. This is difficult because of universal quantification over testers: checking equivalence means checking that two processes are indistinguishable for *any* tester process, and there are infinitely many such processes. This problem has been addressed in [2] and [3], where tractable proof methods aimed at checking the testing equivalence of spi calculus processes are

introduced. In [2], the proof method is based on a bisimulation relation that is a sufficient but not necessary condition for testing equivalence. In [3], a more accurate method is proposed to check may-testing, an equivalence very similar to the testing equivalence defined in [1]. The approach presented in [3] starts from the definition of a contextual labeled transition system which represents the protocol behavior constrained by the knowledge which the environment has about names and keys. The proof method exploits the fact that the trace equivalence defined on this model is a necessary and sufficient condition for may-testing equivalence.

Differently from the above two approaches, this paper investigates the possibility of checking the spi calculus may-testing equivalence using state exploration instead of theorem proving. The quantification over contexts problem is solved in a way similar to the one reported in [3], i.e. by defining a labeled transition system such that trace equivalence is necessary and sufficient for may-testing equivalence. The other new problem that has to be solved in order to make state exploration possible and effective is to keep the size of the trace sets to be explored within finite and reasonable bounds. This objective is achieved mainly in two different ways. First of all, to keep the state set finite, only spi calculus processes having a finite number of parallel instances are dealt with. In practice, our approach is to substitute any replication expression of the form $!P$, which is interpreted in spi calculus as an infinite number of copies of P running in parallel, with a finite number n of parallel copies of P. Since the replication operator is generally used to represent parallel sessions of a cryptographic protocol, this restriction is equivalent to considering up to n parallel runs of the protocol. Consequently, attacks that are possible only with more than n parallel sessions cannot be detected in this way. A similar restriction is adopted in the literature whenever state exploration methods are used and is generally considered fairly acceptable, since bugs tend to show up with few numbers of parallel sessions.

A second way to reduce the size of the trace sets consists in avoiding the explicit representation of transitions corresponding to inputs from the environment. Since the environment can in principle send at any time any data that is part of its knowledge as well as any fresh name or integer, the number of such transitions can be huge if not infinite. Our approach is to represent a set of such transitions as a single transition with an abstract label called generic term or generic value. This kind of reduction does not imply any loss in accuracy and is a key means to make traces enumerable and to limit their number.

The paper is organized as follows: section 2 briefly introduces the language and our conventions about symbols, section 3 deals with the

$\sigma, \rho, \theta, \eta$ ::= terms		P, Q, R ::=	processes
m	name	$\overline{\sigma}\langle\rho\rangle.P$	output
(σ, ρ)	pair	$\sigma(x).P$	input
0	zero	$P \mid Q$	composition
$suc(\sigma)$	successor	$(\nu b)\ P$	restriction
x	variable	$!P$	replication
$\{\sigma\}_\rho$	shared-key encryption	0	nil
		$[\sigma\ is\ \rho]\ P$	match
		$let\ (x, y) = \sigma\ in\ P$	pair splitting
		$case\ \sigma\ of\ 0 : P\ suc(x) : Q$	integer case
		$case\ \eta\ of\ \{x\}_\rho\ in\ P$	shared-key decryption

Table 1 Syntax of spi calculus

definition of our labeled transition system and section 4 explains the derivation rules for it and gives an example. In section 5 the concept of *trace equivalence* for our labeled transition system is defined while section 6 contains some final remarks.

2. THE LANGUAGE

The spi-calculus is a process algebraic language defined in [1] as an extension of the π-calculus, specifically designed for the specification of cryptographic protocols. No explicit types exist in spi calculus and spi entities are divided into processes and terms only. However, since spi calculus terms are extended here with generic terms, we prefer to use the following partially different naming convention:

- m ranges over names;
- n ranges over natural numbers;
- x and y range over variables;
- b is a fresh name, and $\tilde{b}$ is a tuple of fresh names;
- P, Q and R are spi calculus processes;
- σ, ρ, η, θ and ψ denote terms in the most general sense (i.e. generic terms also);
- Σ is a set of terms;
- Γ is the set containing all generic terms;
- γ ranges over Γ;
- λ is a substitution list i.e. $\lambda = \sigma_1/\rho_1, \sigma_2/\rho_2, \cdots \sigma_n/\rho_n$. $\theta[\lambda]$, $P[\lambda]$ and $\Sigma[\lambda]$ mean that each term ρ_i must be replaced by σ_i in θ, P and Σ respectively.

Tab.1 shows the language syntax. Informally, the meaning of each construct of the language is the following:

- Term $\{\sigma\}_\rho$ is the encrypted message obtained by encrypting σ under key ρ using a shared-key cryptosystem.

- $\overline{\sigma}\langle\rho\rangle.P$ is an *output process*, ready to output ρ on channel σ when a synchronization occurs. After the synchronization, P starts. σ should be a name.
- $\sigma(x).P$ is an *input process*, ready to perform an input from channel σ when a synchronization occurs. After the synchronization, $P[\eta/x]$ starts, where η is the input message. σ should be a name.
- $P \mid Q$ is a *parallel composition* where P and Q run in parallel. They may either synchronize between themselves or with the external environment separately.
- $(\nu b)P$ is a *restriction* where a fresh, private name b is made and then P starts.
- $!P$ is a *replication* where an unbounded number of instances of P run in parallel.
- $[\sigma \; is \; \rho]P$ is a *match.* It means that P starts when σ and ρ are the same; it stucks otherwise.
- 0 is the *nil* process: it does nothing.
- *let* $(x, y) = \sigma$ *in* P is a *pair splitting* and it behaves as $P[\rho/x, \theta/y]$, provided that term σ is a pair (ρ, θ); it stucks otherwise.
- *case* σ *of* $0 : P$ $suc(x) : Q$ is an *integer case.* If σ is 0, it behaves as P; if σ is $suc(\rho)$, $Q[\rho/x]$ starts. It stucks otherwise.
- *case* η *of* $\{x\}_\rho$ *in* P is a *shared-key decryption* and behaves as $P[\sigma/x]$ if η is a cyphertext as $\{\sigma\}_\rho$. Otherwise it stucks.

About encryption the following implicit assumptions hold:

- an encrypted message can be decrypted by means of the corresponding key only;
- the encryption key cannot be detected from the encrypted message;
- an encrypted message is sufficiently redundant so that the decryption algorithm can detect whether its task succeeded in.

All the above assumptions are formally defined at the end of section 4.

The spi calculus operational semantics is defined in [1]. For lack of space, they are not recalled here. In this paper we adhere to such definitions, with the exception of the replication operator $!P$, which is interpreted here as a syntactical shortcut for the parallel composition of n parallel copies of P (i.e. $!P = \underbrace{P|P|\cdots|P}_{n}$), where n is a predetermined finite natural number.

3. THE ENVIRONMENT-SENSITIVE LTS

Our first objective is the definition of an *environment-sensitive labeled transition system* (ES-LTS) describing all the possible interactions of a given spi calculus process with its environment. Although the concept

is borrowed from [3], our specific ES-LTS differs substantially from the one described in [3].

The environment that is considered in building the ES-LTS is the most powerful one, power being measured by the knowledge the environment acquires by interacting with the spi process. For this reason, in analogy with [3], each state of our ES-LTS is made up of a spi calculus process P and an environment's knowledge Σ, and is denoted $\Sigma \rhd P$. Σ can be informally defined as the *minimal* set of spi calculus terms that allows the environment to build everything it has learned during the preceding interactions with P.

Transitions are described by the following syntactical form, already introduced in [3]:

$$\Sigma \rhd P \quad \overset{\mu}{\underset{\epsilon}{\longmapsto}} \quad \Sigma' \rhd P' \tag{1}$$

where μ is the action performed by process P, and ϵ is a complementary action used to represent additional information that must be recorded in the ES-LTS traces. While in [3] such complementary actions represent environment actions, in our model they are also used for other purposes.

Transitions are categorized into 4 different types, according to their meaning:

$$\Sigma \rhd P \quad \overset{\tau}{\underset{-}{\longmapsto}} \quad \Sigma \rhd P' \tag{2}$$

$$\Sigma \rhd P \quad \overset{\overline{\sigma}}{\underset{\delta_\Sigma}{\longmapsto}} \quad \Sigma' \rhd P' \tag{3}$$

$$\Sigma \rhd P \quad \overset{\sigma}{\underset{\gamma}{\longmapsto}} \quad \Sigma \rhd P' \tag{4}$$

$$\Sigma \rhd P \quad \overset{\overline{\ }}{\underset{\lambda}{\longmapsto}} \quad \Sigma' \rhd P' \tag{5}$$

Transitions taking form (2) are related to synchronization events occurring inside the spi process and for this reason they leave the environment unaware of what has happened, and do not involve any *knowledge migration* (Σ does not change).

Transitions taking the second and third form instead are related to synchronization events between the spi process and the environment.

Form (3) represents an output on channel σ, and implies a data transfer from the process to the environment. In this case there is also a complementary action representing an increment in the environment knowledge. This is denoted as δ_Σ and represents the set of new terms added to Σ to yield the new knowledge Σ'.

Form (4) represents an input on channel σ, and implies a data transfer from the environment to the process. Thus, no modification in the environment knowledge takes place. In this case, the complementary action represents the data sent by the environment to the process. In order to limit the number of transitions, such data is not represented explicitly, but it is represented in an abstract way by means of a generic value

denoted γ, symbolizing each value the environment is able to build at that time. Formally, γ is a pair $\langle m_\gamma, \Sigma_\gamma \rangle$ where m_γ is a fresh name that identifies the term and Σ_γ is the set of terms representing the knowledge of the environment at the time the data represented by the term was generated (i.e. Σ). The behavior that follows the input of a *generic value* is indeed the abstract representation of a set of behaviours, each one corresponding to a different actual value of the generic term. Such behaviors are *indistinguishable* from one another (in the sense they are not affected by the particular value of the input) until the generic value that has been received is *filtered* in some way by means of a *let*, *case* or *is* operator. Whenever this happens, the contents of the environment's knowledge at the time the *generic value* had been generated is analyzed, in order to find the satisfying value(s), i.e. the actual values of γ which enable the behavior to continue after the filtering operation.

To represent the fact that only some of the behaviors continue after a filtering operation, we use another kind of transition, which takes the form (5). This transition is labeled by a substitution list λ representing the substitution of a generic term with a corresponding set of satisfying values. In this case, P' and Σ' are obtained actualizing P and Σ by λ ($P' = P[\lambda]$, $\Sigma' = \Sigma[\lambda]$).

4. DERIVATION RULES FOR THE ES-LTS

The spi calculus semantics is defined in [1] by means of the reaction relation $\rightarrow$, where $P \rightarrow P'$ means that process P can evolve into P' by performing an (internal) action. To formally define our environment sensitive LTS we use a set of derivation rules based upon the derivation system defined in [1] for the reaction relation. In practice, the reaction relation definition rules are assumed implicitly. Although generic terms are new with respect to the original spi calculus, we assume that derivation rules as defined in [1] apply to generic terms as to any other term. This does not introduce any technical problem, because all terms are treated in the same way in [1].

In order to minimize the number of derivation rules needed, we exploit the following property:

$$\forall R \begin{cases} \exists P, Q, \tilde{b} \mid R \equiv (\nu \tilde{b})(\overline{\sigma}\langle\rho\rangle.P \mid Q) \\ \qquad\qquad \vee \\ \exists P, Q, \tilde{b} \mid R \equiv (\nu \tilde{b})(\sigma(x).P \mid Q) \\ \qquad\qquad \vee \\ R \equiv 0 \end{cases} \tag{6}$$

where symbol $\equiv$ means *structural equivalence* as defined in [1]. Informally this property means that each spi expression R can always be

reduced to a structural equivalent expression taking one of the three syntactical forms shown in (6), i.e. input and output action prefix and inaction. Consequently, it is enough only to define derivation rules for such syntactical forms (indeed, no rule is needed for inaction because it represents a stuck process).

The following rules specify the conditions for transitions of types (2), (3) and (4):

$$\frac{(\nu\,\tilde{b})\,(\overline{\sigma}\langle\rho\rangle.P \mid \sigma(x).Q) \longrightarrow (\nu\,\tilde{b})\,(P \mid Q[\rho/x])}{\Sigma \rhd (\nu\,\tilde{b})\,(\overline{\sigma}\langle\rho\rangle.P \mid \sigma(x).Q) \xmapsto[-]{\tau} \Sigma \rhd (\nu\,\tilde{b})\,(P \mid Q[\rho/x])} \tag{7}$$

$$\frac{\Sigma \vdash \sigma}{\Sigma \rhd (\nu\,\tilde{b})\,(\overline{\sigma}\langle\rho\rangle.P \mid Q) \xmapsto[\overline{\Sigma\cup\{\rho\}}\setminus(\overline{\Sigma\cup\{\rho\}}\cap\Sigma)]{\overline{\sigma}} \overline{\Sigma\cup\{\rho\}} \rhd (\nu\,\tilde{b})\,(P \mid Q)} \tag{8}$$

$$\frac{\Sigma \vdash \sigma}{\Sigma \rhd (\nu\,\tilde{b})\,(\sigma(x).P \mid Q) \xmapsto[\gamma]{\sigma} \Sigma \rhd (\nu\,\tilde{b})\,(P[\gamma/x] \mid Q)} \tag{9}$$

Rule (7) describes interactions on channel σ, internal to the spi process itself. Each of them corresponds exactly to a reaction step as defined in [1]. The environment is not involved in such an event, thus it is unaware of what has happened, and it's knowledge Σ remains unchanged after the event. Label τ emphasizes the silence of the event.

Rule (8) describes what happens when the spi process is ready to perform an output on a certain channel σ, provided that σ belongs to the environment's knowledge (this precondition is expressed by $\Sigma \vdash \sigma$, formally defined at the end of this section). The transition is labeled with the output channel $\overline{\sigma}$ and with the new knowledge that message ρ brings to the environment's knowledge. Such new knowledge can be computed as $\overline{\Sigma \cup \{\rho\}} \setminus (\overline{\Sigma \cup \{\rho\}} \cap \Sigma)$, where $\overline{\Sigma \cup \{\rho\}}$ represents the final knowledge reached after the addition of ρ to Σ.

It is worth noting that the update of the environment's knowledge is not simply $\Sigma \cup \{\rho\}$, because the addition of ρ to Σ could enable a further decoding of terms already present in Σ. The notation $\overline{\Sigma}$ informally represents the minimal set of terms needed to generate all the elements that can be generated by Σ. In practice, we can imagine that $\overline{\Sigma}$ is the result of two operations: first, Σ is expanded by adding to it all the terms that derive from all possible decoding operations. It is then minimized, considering that, when an element of Σ has been decoded into simpler parts (already added to Σ too), it is no longer necessary to continue to keep it in Σ, since in can be built by means of its parts. A formal definition of this operation is given at the end of this section.

For example, let us consider $P(M,\, k) ::= \overline{a}\langle\{M\}_k\rangle.\overline{b}\langle k\rangle.Q$ with $\Sigma = \{a,\, b\}$. By means of rule (8) we have the following behavior:

$$\{a, b\} \rhd \overline{a}\langle\{M\}_k\rangle.\overline{b}\langle k\rangle.Q \xrightarrow[\{\{M\}_k\}]{\overline{a}}$$

$$\{a, b, \{M\}_k\} \rhd \overline{b}\langle k\rangle.Q \xrightarrow[\{M,k\}]{\overline{b}} \{a, b, M, k\} \rhd Q$$

The example shows clearly how the environment's knowledge is rearranged after k has been received from b and that $\{M, k\}$ is the *new* knowledge added to the environment's knowledge, instead of only $\{k\}$. In particular, looking at event $\xrightarrow[\{M,k\}]{\overline{b}}$, we start with $\Sigma = \{a, b, \{M\}_k\}$ and, with $\rho = k$, $\Sigma \cup \{\rho\} = \{a, b, \{M\}_k, k\}$. By means of k, now $\{M\}_k$ can be decrypted and replaced by M. Such an operation is performed by operator $\overline{\ }$, thus $\overline{\Sigma \cup \{\rho\}} = \{a, b, M, k\}$. $\overline{\Sigma \cup \{\rho\}} \cap \Sigma = \{a, b\}$ describes the knowledge that is present both in Σ and in $\overline{\Sigma \cup \{\rho\}}$, i.e. knowledge not affected by the reception of ρ, thus $\overline{\Sigma \cup \{\rho\}} \setminus (\overline{\Sigma \cup \{\rho\}} \cap \Sigma)$ represents the knowledge brought, directly and indirectly, by ρ, i.e. the *new* knowledge used to label the transition.

Rule (9) describes what happens when the spi process is ready to perform an input from a certain channel, provided that the channel belongs to the environment's knowledge ($\Sigma \vdash \sigma$). The transition is labeled with the input channel σ and with the generic term γ which describes *everything the environment can build by using elements of its knowledge* Σ. After synchronization, the input variable x is bound to γ. The environment's knowledge Σ does not need to be updated, as it is the action of transferring data from the environment to the spi process.

Rule (9) empasizes the moment when a behavior comprising all possible behaviors for each acceptable value of γ starts. Thus semantic rules must be introduced which are able to emphasize when a certain abstract behavior, representing all possible behaviors for a generic term γ, continues only for some particular values of γ. Such rules are the ones needed to define type (5) transitions and are applied whenever two terms must be compared. It is worth noting that after a comparison operation each generic term is not necessarily substituted with a non-generic value, but it can be substituted with another more specialized term, made up of other generic terms. For example, rule (12) replaces a generic term with a pair of generic terms.

$$\frac{\Omega(\sigma \bullet \rho) \wedge \sigma \bullet \rho = \langle\psi, \langle\lambda\rangle\rangle \wedge \lambda \neq \emptyset}{\Sigma \rhd (\nu \tilde{b})\,(\overline{\sigma}\langle\theta\rangle.P \mid \rho(x).Q \mid R) \xrightarrow[\lambda]{\tau} \overline{\Sigma[\lambda]} \rhd (\nu \tilde{b})\left(\overline{\psi}\langle\theta\rangle.P[\lambda] \mid \psi(x).Q[\lambda] \mid R[\lambda]\right)} \quad (10)$$

Rule (10) must be used whenever the spi process becomes ready on both one input and one output channel, in order to check whether the channels can be the same or not.

$\Omega(\sigma \bullet \rho)$ indicates whether the match exists, and, if it does, ψ is the matching value, and λ is the substitution list which must be applied to

turn σ and/or ρ into ψ. In practice ψ and λ make it possible to satisfy the following: $\sigma[\lambda] = \rho[\lambda] = \psi$, and the transition labeled with λ means that future behavior is allowed only when $\sigma[\lambda] = \rho[\lambda] = \psi$. $\sigma \bullet \rho$ and $\Omega(\sigma \bullet \rho)$ are formally defined in [12].

The case $\lambda = \emptyset$ means that the match exists and no substitution list is needed i.e. σ and ρ are the same term; but this case is already handled in [1].

$$\frac{\Omega(\sigma \bullet \rho) \wedge \sigma \bullet \rho = \langle \psi, \langle \lambda \rangle \rangle \wedge \lambda \neq \emptyset}{\Sigma \rhd ([\sigma \; is \; \rho] \; P) \mid Q \xrightarrow[\lambda]{} \overline{\Sigma[\lambda]} \rhd P[\lambda] \mid Q[\lambda]} \tag{11}$$

Rule (11) tries to equate terms σ and ρ. In case of success, the behavior expressions P and Q are updated with the substitution list λ and then enabled. The environment's knowledge Σ is also updated with λ. The matching value ψ is not explicitly used, but λ contains all the information needed to turn σ and/or ρ into ψ in P and Q.

$$\Sigma \rhd (let \; (x,y) = \gamma \; in \; P) \mid Q \xrightarrow[\frac{(\gamma', \gamma'')}{\gamma}]{} \overline{\Sigma\left[\frac{(\gamma', \gamma'')}{\gamma}\right]} \rhd P\left[\frac{(\gamma', \gamma'')}{\gamma}, \frac{\gamma'}{x}, \frac{\gamma''}{y}\right] \mid Q\left[\frac{(\gamma', \gamma'')}{\gamma}\right] \tag{12}$$

where $\gamma' = \langle m_{\gamma'}, \Sigma_\gamma \rangle$ and $\gamma'' = \langle m_{\gamma''}, \Sigma_\gamma \rangle$; $m_{\gamma'}$ and $m_{\gamma''}$ are fresh names.

Rule (12) equates the pair of variables (x, y) with a generic term which is turned into a pair of generic terms whose names are fresh and whose knowledge is inherited from γ. As shown by the substitution list, this new pair replaces γ in P and its components are respectively assigned to x and y.

$$\Sigma \rhd (case \; \gamma \; of \; 0 : P \; suc(x) : Q) \mid R \xrightarrow[0/\gamma]{} \overline{\Sigma[0/\gamma]} \rhd P[0/\gamma] \mid R[0/\gamma] \tag{13}$$

$$\Sigma \rhd (case \; \gamma \; of \; 0 : P \; suc(x) : Q) \mid R \xrightarrow[\gamma'/\gamma]{} \overline{\Sigma[\gamma'/\gamma]} \rhd Q[\gamma'/\gamma, \gamma'/x] \mid R[\gamma'/\gamma] \tag{14}$$

where $\gamma' = \langle m_{\gamma'}, \emptyset \rangle$ and $m_{\gamma'}$ is a fresh name. Such a restriction allows to constrain the generic data to be an integer number only (due to rule (17), an empty Σ can always produce integer numbers).

When a generic term is involved in such a construction, the behaviour forks. In fact, if the value 0 is assumed, then behavior P is enabled, otherwise Q. In the former case, the generic term γ has been replaced by 0, while in the latter one all integer values are still allowed, thus the domain of the new generic term γ' has been restricted to the set of natural numbers (17).

$$\frac{\Omega(\eta \circ \rho) \wedge \eta \circ \rho = \langle \psi, \langle \lambda \rangle \rangle \wedge \lambda \neq \emptyset}{\Sigma \rhd (case \; \eta \; of \; \{x\}_\rho \; in \; P) \mid Q \xrightarrow[\lambda]{} \overline{\Sigma[\lambda]} \rhd P[\lambda, \psi/x] \mid Q[\lambda]} \tag{15}$$

Behavior P is enabled when term η can be considered as *something encrypted by means of* ρ. In this case $\Omega(\eta \circ \rho)$ returns true, ψ is the

value of the encrypted message, and λ is the substitution list that gives the equivalence $\eta[\lambda] = \{\psi\}_{\rho[\lambda]}$. The environment's knowledge Σ is accordingly updated. $\eta \circ \rho$ and $\Omega(\eta \circ \rho)$ are formally defined in [12].

The following rules allow us to verify whether a certain term can be generated by means of the contents of Σ:

$$\Sigma_\gamma \sqsubseteq \Sigma \Leftrightarrow \Sigma \vdash \gamma \quad (16)$$

$$\Sigma \vdash n \quad (17)$$

$$\rho \in \Sigma \Rightarrow \Sigma \vdash \rho \quad (18)$$

$$\Sigma \vdash suc(\rho) \Leftrightarrow \Sigma \vdash \rho \quad (19)$$

$$\Sigma \vdash \rho' \wedge \Sigma \vdash \rho'' \Leftrightarrow \Sigma \vdash (\rho', \rho'') \quad (20)$$

$$\Sigma \vdash \{\rho\}_\sigma \wedge \Sigma \vdash \sigma \Rightarrow \Sigma \vdash \rho \quad (21)$$

$$\Sigma \vdash \rho \wedge \Sigma \vdash \sigma \Rightarrow \Sigma \vdash \{\rho\}_\sigma \quad (22)$$

where

$$\Sigma' \sqsubseteq \Sigma'' \triangleq \Sigma'' \vdash \sigma' \ \forall \sigma' \mid \Sigma' \vdash \sigma' \quad (23)$$

Definition (23) does not introduce circularity in (16) since $\gamma \notin \Sigma_\gamma$.

In the following, we introduce a set of rules that should allow to compute $\overline{\Sigma}$ from Σ:

$$\frac{\Sigma \vdash \gamma}{\Sigma \twoheadrightarrow \Sigma \setminus \{\gamma\}} \quad (24)$$

$$\frac{\rho \in \Sigma \wedge \Sigma \setminus \{\rho\} \vdash \rho}{\Sigma \twoheadrightarrow \Sigma \setminus \{\rho\}} \quad (25)$$

$$\frac{suc(\rho) \in \Sigma}{\Sigma \twoheadrightarrow \Sigma \setminus \{suc(\rho)\} \cup \{\rho\}} \quad (26)$$

$$\frac{(\rho', \rho'') \in \Sigma}{\Sigma \twoheadrightarrow \Sigma \setminus \{(\rho', \rho'')\} \cup \{\rho'\} \cup \{\rho''\}} \quad (27)$$

$$\frac{\{\rho\}_\sigma \in \Sigma \wedge \Sigma \vdash \sigma}{\Sigma \twoheadrightarrow \Sigma \setminus \{\{\rho\}_\sigma\} \cup \{\rho\}} \quad (28)$$

It can be proven [12] that the number of all allowed evolutions, starting from Σ, is finite and each one has a finite number of steps. In particular the last set of each evolution is $\overline{\Sigma}$. The proof can be made by underlining that each rule tends to replace a composite term with its components (i.e. no rule does insert in Σ any element previously extracted by another one) and, given two or more rules, their application (evaluation) order does not affect the obtained set.

4.1. AN EXAMPLE OF ES-LTS

Fig.1 shows the ES-LTS of a simple spi process. We have two processes: $(\nu k)(\overline{c}\langle k\rangle.c(y).[C(y, \{z\}_k)]\ P(z))$ and $(\nu M)(c(x).\overline{c}\langle\{M\}_x\rangle.Q)$. The former generates a fresh key which is sent to the latter which encrypts M with the received key and sends back the encrypted message to the former. The former checks the received message ($[C(y, \{z\}_k)]$ stands for *case* y *of* $\{z\}_k$ *in*) and enters P. In fig. 1 each behavior is bounded by a box, and the related environment's knowledge is enclosed in braces

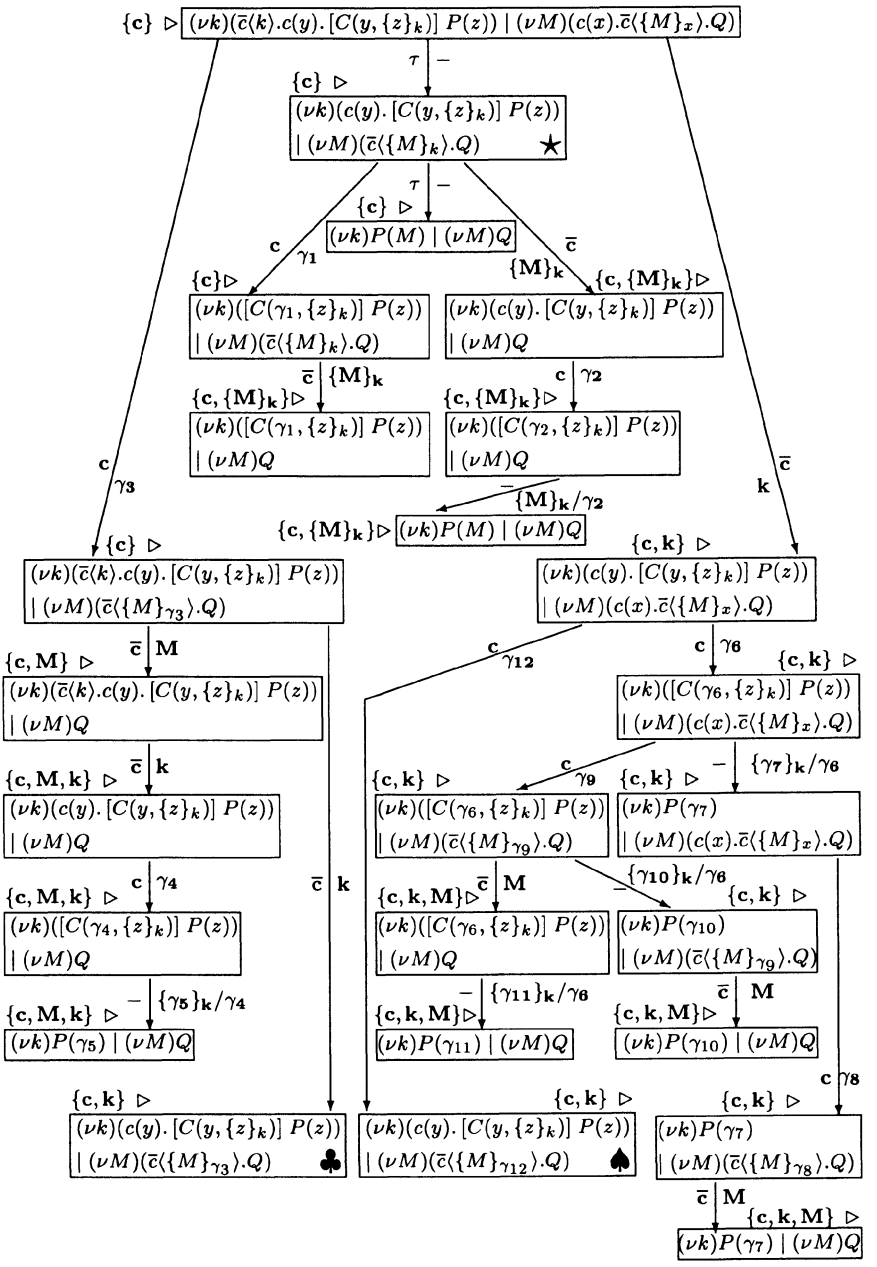

Figure 1 Example of ES-LTS

near to the box itself. The initial environment's knowledge contains only the unrestricted name c. The initial specification can evolve along the three following paths:

a) an internal synchronization occurs ($\overset{\tau}{\longmapsto}$). The behavior expression is reduced accordingly.

   If a second internal synchronization occurs, a final state (P and Q are not specified) is reached without any interaction with the environment. The two alternative paths concern interactions with the environment; in particular they are the two interleavings of the allowed input and output interaction with the environment. In case of the input interaction first ($\underset{\gamma_1}{\overset{c}{\longmapsto}}$), the environment knows only c, thus in the next state it is not possible to find a specializing value for γ_1 that matches the constraint ($[C(\gamma_1, \{z\}_k)]$). By contrast, when the process output precedes the input, the environment receives $\{M\}_k$ (even if it cannot decrypt it) before generating γ_2, which can be turned into $\{M\}_k$ ($\underset{\{M\}_k/\gamma_2}{\overset{-}{\longmapsto}}$)in the next state.

b) an input interaction occurs ($\underset{\gamma_3}{\overset{c}{\longmapsto}}$): in this case a path leads to a final state, moreover a second path leads to a behavior labeled with ♣. Due to lack of space, this behavior has not been expanded further; however it is similar to the one labeled with $\star$, where γ_3 replaces k and the environment also knows k. Thus the set of allowed evolutions is richer than the one obtained from $\star$.

c) an output interaction occurs ($\underset{k}{\overset{\bar{c}}{\longmapsto}}$): in this case, a path leads to a behavior labeled with ♠, similar to the one with ♣. Another path leads to a little more complex path, which takes into account all allowed interleavings between actions.

Now let us analyze the leftmost path in more detail: the environment knows c, thus it sends a generic term γ_3 to the spi process over channel c. γ_3 represents each message the environment is able to build at this point. The spi process uses γ_3 to encrypt M; when it outputs $\{M\}_{\gamma_3}$ on channel c, the environment is able to capture M since it knows c and M has been encrypted by means of *something* it knows. Thus the transition has been labeled with M, which has also been added to the environment's knowledge. The spi process is now ready to output k on channel c; k is captured by the environment (which enlarges its knowledge) and it is used to label the transition. At the next input event, the environment gives a generic γ_4 to the spi process. The last transition means that the last behavior is reached only when γ_4 can be seen as *something encrypted with* k. It is allowed since Σ_{γ_4} includes k.

The meaning of the whole path is the following: the path to the last but one state includes all paths that can be built using each allowed message the environment can build in correspondence to the two input events. The last state is reachable only when the second generic message (γ_4) has been made as *something encrypted with k*. The *something* cannot only be M, but also (M, M), $\{\{M, c\}_k\}_k$ (even if meaningless) and so on.

5. TRACE SEMANTICS

As seen above, transitions take the forms (2), (3), (4) and (5). By introducing $\Longrightarrow$ as the reflexive and transitive closure of $\underset{-}{\overset{\tau}{\longmapsto}}$, we inductively define $\underset{b}{\overset{a}{\Longrightarrow}}$ as $\Longrightarrow \underset{\beta}{\overset{\alpha}{\longmapsto}} \underset{b'}{\overset{a'}{\Longrightarrow}}$ where $a = \alpha \cdot a'$ and $b = \beta \cdot b'$ and the pair (α, β) can assume the forms $(\overline{\sigma}, \delta\Sigma)$, (σ, γ) and $(-, \lambda)$ in the cases of (3), (4) and (5) respectively.

In order to check whether two processes are trace equivalent, for a process P, we define:

$$traces(P) = \{(a, b) \mid \exists\, P' \mid P \underset{b}{\overset{a}{\Longrightarrow}} P'\} \tag{29}$$

where P is a spi process and, inductively, the trace equivalence:

$$(\alpha'.a', \beta'.b') \simeq (\alpha''.a'', \beta''.b'') \Leftrightarrow (\alpha', \beta') \sim (\alpha'', \beta'') \wedge (a', b') \simeq (a'', b'') \tag{30}$$

where $(\alpha', \beta') \sim (\alpha'', \beta'')$ is defined as it follows:

$$(c', \gamma') \sim (c'', \gamma'') \iff c' \equiv c'' \wedge m_{\gamma'} \equiv m_{\gamma''} \tag{31}$$

$$(\overline{c'}, \delta\Sigma') \sim (\overline{c''}, \delta\Sigma'') \iff \overline{c'} \equiv \overline{c''} \wedge R(\delta\Sigma', \delta\Sigma'') \tag{32}$$

$$(-, \lambda') \sim (-, \lambda'') \iff S(\lambda', \lambda'') \tag{33}$$

$R(\delta\Sigma', \delta\Sigma'')$ holds true if a bijective mapping exists between $\delta\Sigma'$ and $\delta\Sigma''$ so that each pair of the mapping (σ', σ'') satisfies the following:

$$\sigma' \equiv \sigma'' \vee \left(\sigma' \equiv \{\eta'\}_{\theta'} \wedge \sigma'' \equiv \{\eta''\}_{\theta''}\right) \tag{34}$$

$S(\lambda', \lambda'')$ holds true if a bijective mapping exists between λ' and λ'' so that each pair of the mapping $(\sigma'/\gamma', \sigma''/\gamma'')$ satisfies the following:

$$m_{\gamma'} \equiv m_{\gamma''} \wedge \left(\sigma' \equiv \sigma'' \vee \left(\begin{array}{l} \sigma' \equiv \{\eta'\}_{\theta'} \\ \wedge \\ \sigma'' \equiv \{\eta''\}_{\theta''} \\ \wedge \\ \Sigma_{\gamma'} \setminus \{\sigma'\} \not\vdash \sigma' \\ \wedge \\ \Sigma_{\gamma''} \setminus \{\sigma''\} \not\vdash \sigma'' \end{array}\right)\right) \tag{35}$$

In practice, in the case of input transitions (31), we require that the names of the two channels are the same, and that the two generic terms are also the same; in the case of output transitions (32), we require that the new data added to the environment's knowledge are the same or are indistinguishable from the environment's point of view (34). It must be remembered that when an encrypted term is present in $\delta\Sigma$, it means that the environment cannot decrypt it, thus it cannot distinguish among such kinds of terms.

In the case of *specializing* transitions (33), two corresponding substitution items must replace the same generic term ($m_{\gamma'} \equiv m_{\gamma''}$) and the replacing terms must be the same, or must be indistinguishable, in the sense of the environment's knowledge. In fact two different replacing terms can be considered equivalent when they are encrypted terms that the environment's knowledge cannot decrypt (last sentence in (35)).

Everything works well since the following assumptions holds: the same criteria must be used in generating the name of generic terms in both processes (rules (31) and (35)), and the initial environment's knowledge must be the same for both processes. In particular, it is assumed that the initial environment's knowledge of a process explicitly contains each unrestricted name of the process specification.

The fact that two trace equivalent processes in our definition are also testing equivalent, and vice versa, will be proven in a future work.

6. CONCLUSIONS

A tractable method to check the spi calculus may-testing equivalence by state space exploration has been presented. It deals with the spi calculus as defined in [1], with the only exception of the replication operator. As far as we know, this is the first attempt in this direction. Previous related work, reported in [2] and [3], investigated only the theorem proving approach.

Further research efforts are oriented to formally prove soundness and completeness of the method and to develop a tool that can implement the verification task automatically.

References

[1] M. Abadi, and A. D. Gordon, "A Calculus for Cryptographic Protocols The Spi Calculus", *Digital Research Report*, vol. 149, January 1998, pp. 1-110.

[2] M. Abadi, and A. D. Gordon, "A bisimulation method for cryptographic protocols", *Nordic Journal of Computing*, Vol. 5, pp. 267-303, 1998.

[3] M. Boreale, R. De Nicola, and R. Pugliese, "Proof Techniques for Cryptographic Processes", *Proc. of the 14th IEEE Symposium Logic In Computer Science (LICS'99)*, IEEE Computer Society Press, pp. 157-166, 1999.

[4] G. Lowe, "Breaking and fixing the Needham-Schroeder public-key protocol using FDR", *Proc. of TACAS'97*, Springer LNCS 1055, 1996.

[5] G.Lowe, "Casper: a compiler for the analysis of security protocols", *Proc. of 1996 IEEE Computer Security Foundations Workshop*, IEEE Computer Society Press, 1996.

[6] G. Lowe, B. Roscoe, "Using CSP to Detect Errors in the TMN Protocol", *IEEE Transactions on Software Engineering*, Vol. SE-23, No. 10, pp. 659-669, October 1997.

[7] J. K. Millen, S. C. Clark, and S. B. Freedman, "The Interrogator: Protocol Security Analysis", *IEEE Transactions on Software Engineering*, Vol. SE-13, No. 2, pp. 274-288, February 1987.

[8] G. Leduc, O. Bonaventure, L. Léonard, E. Koerner, and C. Pecheur, "Model-Based Verification of a Security Protocol for Conditional Access to Services", *Formal Methods in System Design*, Vol. 14, No. 2, pp. 171-191, March 1999.

[9] R. Milner, J. Parrow, and D. Walker, "A Calculus of mobile processes, parts I and II", *Information and Computation*, pages 1-40 and 41-77, September 1992.

[10] L. C. Paulson, "The inductive approach to verifying cryptographic protocols", *Journal of Computer Security*, Vol. 6, pp. 85-128, 1998.

[11] S. Schneider, "Verifying Authentication Protocols in CSP", *IEEE Transactions on Software Engineering*, Vol. SE-24, No. 9, pp. 741-758, September 1998.

[12] L. Durante, R. Sisto, and A. Valenzano, "A state-exploration technique for spi-calculus testing equivalence verification", *Technical Report DAI/ARC 1-00*, Politecnico di Torino, Italy, 2000.